UNIT-II

Managing Application Resources: **What Are Resources? -** Storing Application Resources, Resource Value Types, Accessing Resources Programmatically, **Working with Different Types of Resources**-Working with String Resources, Using String Resources as Format Strings, Working with Quantity Strings, Working with String Arrays, Working with Boolean Resources, Working with Integer Resources, Working with Colors, Working with Dimensions, Drawable Resources, Working with Images, Working with Color State Lists, Working with Animation, Working with Menus, Working with XML Files, Working with Raw Files, References to Resources, **Working with Layouts**, Exploring User Interface Building Blocks: **Introducing Android Views and Layouts**-The Android View, The Android Controls, The Android Layout, **Displaying Text to Users with TextView -** Configuring Layout and Sizing, Creating Contextual Links in Text, Retrieving Data from Users with Text Fields-Retrieving Text Input Using EditText Controls, Constraining User Input with Input Filters, Helping the User with Autocompletion, Giving Users Choices Using Spinner Controls, Allowing Simple User Selections with Buttons and Switches - Using Basic Buttons, Using CheckBox and ToggleButton Controls, Using RadioGroup and RadioButton, Retrieving Dates, Times, and Numbers from Users with Pickers, Using Indicators to Display Progress and Activity to Users-Indicating Progress with ProgressBar, Indicating Activity with Activity Bars and Activity Circles, Adjusting Progress with Seek Bars.

Managing Application Resources

The well-written application accesses its resources programmatically instead of the developer hard-coding them into the source code. This is done for a variety of reasons. Storing application resources in a single place is a more organized approach to development and makes the code more readable and maintainable. Externalizing resources such as strings makes it easier to localize applications for different languages and geographic regions. Finally, different resources may be necessary for different devices.

What Are Resources?

All Android applications are composed of two things: functionality (code instructions) and data (resources). The functionality is the code that determines how your application behaves. This includes any algorithms that make the application run. Resources include text strings, styles and themes, dimensions, images and icons, audio files, videos, and other data used by the application.

Storing Application Resources

Android resource files are stored separately from the `.java` class files in the Android project. Most common resource types are stored in XML. You can also store raw data files and graphics as resources. Resources are organized in a strict directory hierarchy. All resources must be stored under the `/res` project directory in specially named subdirectories whose names must be lowercase.

Table 6.1 **Default Android Resource Directories**

Resource Subdirectory	Purpose
/res/drawable-*/	Graphics resources
/res/layout/	User interface resources
/res/menu/	Menu resources for showing options or actions in activities
/res/values/	Simple data such as strings, styles and themes, and dimensions
/res/values-sw*/	Dimension resources for overriding defaults
/res/values-v*/	Style and theme resources for newer API customizations

Resource types are stored in different directories in Android projects, with each type corresponding to a specific subdirectory name. Resources can be further organized using directory qualifiers, such as /res/drawable-hdpi for high-density screens, /res/drawable-ldpi for low-density screens, /res/drawable-mdpi for medium-density screens, /res/drawable-xhdpi for extra-high-density screens, and /res/drawable-xxhdpi for extra-extra-high-density screens. The Android IDE simplifies adding resources to the project, generating the R.java source file.

Resource Value Types

Android applications rely on many different types of resources—such as text strings, graphics, and color schemes—for user interface design. These resources are stored in the `/res` directory of your Android project in a strict (but reasonably flexible) set of directories and files. All resource filenames must be lowercase and simple (letters, numbers, and underscores only).
The resource types supported by the Android SDK and how they are stored withinthe project are shown in Table 6.2.

Table 6.2 **How Common Resource Types Are Stored in the Project File Hierarchy**

Resource Type	Required Directory	Suggested Filenames	XML Tag
Strings	/res/values/	strings.xml	<string>
String pluralization	/res/values/	strings.xml	<plurals>, <item>
Arrays of strings	/res/values/	strings.xml or arrays.xml	<string-array>, <item>
Booleans	/res/values/	bools.xml	<bool>
Colors	/res/values/	colors.xml	<color>
Color state lists	/res/color/	Examples include buttonstates.xml, indicators.xml	<selector>, <item>
Dimensions	/res/values/	dimens.xml	<dimen>
IDs	/res/values/	ids.xml	<item>
Integers	/res/values/	integers.xml	<integer>
Arrays of integers	/res/values/	integers.xml	<integer-array>
Mixed-type arrays	/res/values/	arrays.xml	<array>, <item>
Simple drawables (paintables)	/res/values/	drawables.xml	<drawable>
Graphics definition XML files such as shapes	/res/drawable/	Examples include icon.png, logo.jpg	Supported graphics files or drawables
Tweened animations	/res/anim/	Examples include fadesequence.xml, spinsequence.xml	<set>, <alpha>, <scale>, <translate>, <rotate>
Property animations	/res/animator/	mypropanims.xml	<set>, <objectAnimator>, <valueAnimator>
Frame-by-frame animations	/res/drawable/	Examples include sequence1.xml, sequence2.xml	<animation-list>, <item>
Menus	/res/menu/	Examples include mainmenu.xml, helpmenu.xml	<menu>
XML files	/res/xml/	Examples include data.xml, data2.xml	Defined by the developer

(continues)

Table 6.2 **Continued**

Resource Type	Required Directory	Suggested Filenames	XML Tag
Raw files	`/res/raw/`	Examples include `jingle.mp3`, `somevideo.mp4`, `helptext.txt`	Defined by the developer
Layouts	`/res/layout/`	Examples include `main.xml`, `help.xml`	Varies; must be a layout control
Styles and themes	`/res/values/`	`styles.xml`, `themes.xml`	`<style>`

Accessing Resources Programmatically

Developers access application resources using the R.java class file and its subclasses, which are automatically generated when resources are added to a project using the Android IDE. Resource identifiers in a project can be accessed by their unique name. For example, a String named strHello can be accessed in code as R.string.strHello. This variable is not the actual data associated with the String named hello but is used to retrieve the appropriate String resource from the project resources associated with the application. To retrieve the appropriate resource, the Resources instance for the application Context (android.content.Context) is used, which has helper methods for handling every kind of resource. For example, to retrieve the String text, the getString() method of the Resources class is used.

Working with Different Types of Resources

This section explores the types of resources available for Android applications, their definition in project files, and their programming access to these resources.

Table 6.3 **String Resource Formatting Examples**

String Resource Value	Displays As
`Hello, World`	Hello, World
`"User's Full Name:"`	User's Full Name:
`User\'s Full Name:`	User's Full Name:
`She said, \"Hi.\"`	She said, "Hi."
`She\'s busy but she did say, \"Hi.\"`	She's busy but she did say, "Hi."

The Android IDE resource editor supports certain resource types like string and color, while others like Animation sequences can be managed directly by editing XML files.

Working with String Resources

String resources are a basic type of resource used by developers for text labels, help text, and application names. They are defined in XML and compiled into the application package at build time. String values can be edited using the Resources tab or directly, and resource identifiers are automatically added to the R.java class file. Properly name these resources.
Here is an example of the string resource file /res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">Resource Viewer</string>
```

<string name=”test_string”>Testing 1,2,3</string>
<string name=”test_string2”>Testing 4,5,6</string>
</resources>

Using String Resources as Format Strings

You can create format strings, but you need to escape all bold, italic, and underlining tags if you do so. For example, this text shows a score and the “win” or “lose” string:

<string
name=”winLose”>Score: %1$d of %2$d! You %3$s.</string>
If you want to include bold, italic, or underlining in this format string, you need to escape the format tags. For example, if you want to italicize the “win” or “lose” string at the end, your resource would look like this:

<string name=”winLoseStyled”>
Score: %1$d of %2$d! You <i>%3$s</i>.</string>

Using String Resources Programmatically

There are two primary ways in which you can access a string resource.
The following code accesses your application’s string resource named hello, returning only the String. All HTML-style attributes (bold, italic, and underlining) are stripped from the String.

```
String myStrHello = getResources().getString(R.string.hello);
```

You can also access the String and preserve the formatting by using this other method:

```
CharSequence myBoldStr = getResources().getText(R.string.boldhello);
```

To load a format String, you need to make sure any format variables are properly escaped.

One way you can do this is by using the htmlEncode() method of the TextUtils (android.text.TextUtils) class:

```
String mySimpleWinString;
mySimpleWinString = getResources().getString(R.string.winLose);
String escapedWin = TextUtils.htmlEncode(mySimpleWinString);
String resultText = String.format(mySimpleWinString, 5, 5, escapedWin);
```

The resulting text in the resultText variable is Score: 5 of 5! You Won.
Now if you have styling in this format String like the preceding string resource winLoseStyled, you need to take a few more steps to handle the escaped italic tags. For this, you might want to use the fromHtml() method of the Html class (android.text .Html), as shown here:

```
String myStyledWinString;
myStyledWinString = getResources().getString(R.string.winLoseStyled);
String escapedWin = TextUtils.htmlEncode(myStyledWinString);
String resultText = String.format(myStyledWinString, 5, 5, escapedWin);
CharSequence styledResults = Html.fromHtml(resultText);
```

The resulting text in the styledResults variable is Score: 5 of 5! You Won.
This variable, styledResults, can then be used in user interface controls such as TextView objects, where styled text is displayed correctly.

Working with Quantity Strings

A special resource type called <plurals> can be used to define strings that are useful for changing a word's grammatical quantity form. Here is an example string resource file with the resource path of res/values/strings.xml that defines two different quantity forms of a particular animal name that changes based on the context of the quantity:

```
<resources>
<plurals name="quantityOfGeese">
<item quantity="one">You caught a goose!</item>
<item quantity="other">You caught %d geese!</item>
</plurals>
</resources>
```

The singular form for this particular animal is goose, and the plural form is geese. The %d value is used so we can display the exact quantity of geese to the user. To work with pluralized resources in your code, the method getQuantityString() can be used to retrieve a plural string resource like so:

```
int quantity = getQuantityOfGeese();
Resources plurals = getResources();
```

Table 6.4 String `quantity` Values

Value	Description
`zero`	Used for languages that have words with a zero `quantity` form
`one`	Used for languages that have words with a singular `quantity` form
`two`	Used for languages that have words for specifying two
`few`	Used for languages that have words for specifying a small `quantity` batch
`many`	Used for languages that have words for specifying a large `quantity` batch
`other`	Used for languages that have words that do not have a `quantity` form

```
String geeseFound = plurals.getQuantityString( R.plurals.quantityOfGeese, quantity,
quantity);
```

getQuantityString() takes three variables. The first is the plural resource; the second is the quantity value, which is used to tell the application which grammatical form of the word to display; and the third value is defined only when the actual quantity is to be displayed to the user, substituting %d with the actual integer value. When internationalizing your application, managing the translation of words properly and accounting for quantity in a particular language is very important. Not all languages follow the same rules for quantity, so in order to make this process manageable, using plural string resource files will definitely help. For a particular word, you are able to define many different grammatical forms. To define more than one quantity form of a given word in your string resource file, you simply specify more than one <item> element and provide a value for the quantity attribute for that particular <item> denoting the word's quantity. The values that you can use to specify the <item>'s quantity are shown in Table 6.4.

Working with String Arrays

You can specify lists of strings in resource files. This can be a good way to store menu

options and drop-down list values. String arrays are defined in XML under the /res/ values project directory and compiled into the application package at build time.

String arrays are appropriately tagged with the <string-array> tag and a number of <item> child tags, one for each String in the array. Here is an example of a simple array resource file, /res/values/arrays.xml:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
        <string-array name=”flavors”>
                <item>Vanilla</item>
                <item>Chocolate</item>
                <item>Strawberry</item>
                <item>Coffee</item>
                <item>Sherbet</item>
        </string-array>
        <string-array name=”soups”>
                <item>Vegetable minestrone</item>
                <item>New England clam chowder</item>
                <item>Organic chicken noodle</item>
        </string-array>
</resources>
```

Accessing string array resources is easy. The method getStringArray() retrieves a string array from a resource file, in this case, one named flavors:

```
String[] aFlavors =
        getResources().getStringArray(R.array.flavors);
```

Working with Boolean Resources

Other primitive types are supported by the Android resource hierarchy as well. Boolean resources can be used to store information about application game preferences and default values. Boolean resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

Defining Boolean Resources in XML

Boolean values are appropriately tagged with the <bool> tag and represent a name/value pair. The name attribute is how you refer to the specific Boolean value programmatically, so name these resources wisely.
Here is an example of the Boolean resource file /res/values/bools.xml:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
        <bool name=”onePlusOneEqualsTwo”>true</bool>
        <bool name=”isAdvancedFeaturesEnabled”>false</bool>
</resources>
```

Using Boolean Resources Programmatically

To use a Boolean resource in code, you can load it using the getBoolean() method of the Resources class. The following code accesses your application’s Boolean resource

named bAdvancedFeaturesEnabled:

```
boolean isAdvancedMode =
        getResources().getBoolean(R.bool.isAdvancedFeaturesEnabled);
```

Working with Integer Resources

In addition to strings and Boolean values, you can also store integers as resources. Integer resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

Defining Integer Resources in XML

Integer values are appropriately tagged with the <integer> tag and represent a name/value pair. The name attribute is how you refer to the specific integer programmatically, so name these resources wisely.

Here is an example of the integer resource file /res/values/nums.xml:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
        <integer name=”numTimesToRepeat”>25</integer>
        <integer name=”startingAgeOfCharacter”>3</integer>
</resources>
```

Using Integer Resources Programmatically

To use the integer resource, you must load it using the Resources class. The following code accesses your application’s integer resource named numTimesToRepeat:

```
int repTimes = getResources().getInteger(R.integer.numTimesToRepeat);
```

Working with Colors

Android applications can store RGB color values, which can then be applied to other screen elements. You can use these values to set the color of text or other elements, such as the screen background. Color resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

Defining Color Resources in XML RGB color values always start with the hash symbol (#). The alpha value can be given for transparency control. The following color formats are supported:

- #RGB (for example, #F00 is 12-bit color, red)
- #ARGB (for example, #8F00 is 12-bit color, red with alpha 50%)
- #RRGGBB (for example, #FF00FF is 24-bit color, magenta)
- #AARRGGBB (for example, #80FF00FF is 24-bit color, magenta, with alpha 50%)

Color values are appropriately tagged with the <color> tag and represent a name/value pair. Here is an example of a simple color resource file, /res/values/colors.xml:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
<color name=”background_color”>#006400</color>
<color name=”text_color”>#FFE4C4</color>
</resources>
```

Working with Dimensions

Many user interface layout controls, such as text controls and buttons, are drawn to specific dimensions. These dimensions can be stored as resources. Dimension values always end with a unit of measurement tag.

Defining Dimension Resources in XML

Dimension values are tagged with the <dimen> tag and represent a name/value pair. Dimension resources are defined in XML under the /res/values project directory and compiled into the application package at build time. The dimension units supported are shown in Table 6.5.

Table 6.5 **Dimension Unit Measurements Supported**

Unit of Measurement	Description	Resource Tag Required	Example
Pixels	Actual screen pixels	px	20px
Inches	Physical measurement	in	1in
Millimeters	Physical measurement	mm	1mm
Points	Common font measurement unit	pt	14pt
Screen density—independent pixels	Pixels relative to 160dpi screen (preferable for dimension screen compatibility)	dp	1dp
Scale-independent pixels	Best for scalable font display	sp	14sp

Here is an example of a simple dimension resource file called /res/values/dimens.xml:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
<dimen name=”FourteenPt”>14pt</dimen>
<dimen name=”OneInch”>1in</dimen>
<dimen name=”TenMillimeters”>10mm</dimen>
<dimen name=”TenPixels”>10px</dimen>
</resources>
```

Drawable Resources

The Android SDK supports many different types of drawable resources for managing the different types of graphics files that your project requires. These resource types are also useful for managing the presentation of your project’s drawable files. Table 6.6 presents some of the different types of drawable resources that you can define.

Working with Simple Drawables

You can specify simple colored rectangles by using the drawable resource type, which can then be applied to other screen elements. These drawable resource types are defined in specific paint colors, much as the color resources are defined.

Table 6.6 **Different Drawable Resources**

Drawable Class	Description
`ShapeDrawable`	A geometric shape such as a circle or rectangle
`ScaleDrawable`	Defines the scaling of a `Drawable`
`TransitionDrawable`	Used to cross-fade between drawables
`ClipDrawable`	`Drawable` used to clip a region of a `Drawable`
`StateListDrawable`	Used to define different states of a `Drawable` such as pressed or selected
`LayerDrawable`	An array of drawables
`BitmapDrawable`	Bitmap graphics file
`NinePatchDrawable`	Stretchable PNG file

Working with Images

Applications often include visual elements such as icons and graphics. Android supports several image formats that can be directly included as resources for your application. These image formats are shown in Table 6.7.
These image formats are all well supported by popular graphics editors such as Adobe Photoshop, GIMP, and Microsoft Paint. Adding image resources to your project is easy. Simply drag the image asset into the /res/drawable resource directory hierarchy and it will automatically be included in the application package.

Table 6.7 **Image Formats Supported in Android**

Supported Image Format	Description	Required Extension
Portable Network Graphics (PNG)	Preferred format (lossless)	`.png`
Nine-Patch Stretchable Graphics	Preferred format (lossless)	`.9.png`
Joint Photographic Experts Group (JPEG)	Acceptable format (lossy)	`.jpg, .jpeg`
Graphics Interchange Format (GIF)	Discouraged format	`.gif`
WebP (WEBP)	Android 4.0+	`.webp`

Working with Color State Lists

A special resource type called <selector> can be used to define different colors or drawables to be used depending on a control's state. For example, you could define a color state list for a Button control: gray when the button is disabled, green when it is enabled, and yellow when it is being pressed. Similarly, you could provide different drawables based on the state of an ImageButton control.
The <selector> element can have one or more child <item> elements that define different colors for different states. There are quite a few attributes that you are able to define for the <item> element, and you can define one or more for supporting many different states for your View objects. Table 6.8 shows many of the attributes that you are able to define for the <item> element.

Table 6.8 **Color State List <item> Attributes**

Attribute	Values
color	Required attribute for specifying a hexadecimal color in one of the following formats: #RGB, #ARGB, #RRGGBB, or #AARRGGBB, where A is alpha, R is red, G is green, and B is blue
state_enabled	Boolean value denoting whether this object is capable of receiving touch or click events, true or false
state_checked	Boolean value denoting whether this object is checked or unchecked, true or false
state_checkable	Boolean value denoting whether this object is checkable or not checkable, true or false
state_selected	Boolean value denoting whether this object is selected or not selected, true or false
state_focused	Boolean value denoting whether this object is focused or not focused, true or false
state_pressed	Boolean value denoting whether this object is pressed or not pressed, true or false

Working with Animation

Android provides two categories of animations. The first category, property animation, allows you to animate an object's properties. The second category of animation is view animation. There are two different types of view animations: *frame-by-frame* animation and *tween* animations. Frame-by-frame animation involves the display of a sequence of images in rapid succession. Tweened animation involves applying standard graphical transformations such as rotations and fades to a single image.

The Android SDK provides some helper utilities for loading and using animation resources. These utilities are found in the `android.view.animation.AnimationUtils` class.

Working with Menus

You can also include menu resources in your project files. Like animation resources, menu resources are not tied to a specific control but can be reused in any menu control.

Defining Menu Resources in XML

Each menu resource (which is a set of individual menu items) is stored as a specially formatted XML file in the `/res/menu` directory and is compiled into the application package at build time.

Here is an example of a simple menu resource file called `/res/menu/speed.xml` that defines a short menu with four items in a specific order:

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
<item
android:id="@+id/start"
```

```
android:title="Start!"
android:orderInCategory="1"></item>
<item
android:id="@+id/stop"
android:title="Stop!"
android:orderInCategory="4"></item>
<item
android:id="@+id/accel"
android:title="Vroom! Accelerate!"
android:orderInCategory="2"></item>
<item
android:id="@+id/decel"
android:title="Decelerate!"
android:orderInCategory="3"></item>
</menu>
```

Working with XML Files

You can include arbitrary XML resource files to your project. You should store these XML files in the `/res/xml` directory, and they are compiled into the application package at build time.

Defining Raw XML Resources

First, put a simple XML file in the `/res/xml` directory. In this case, the file `my_pets.xml` with the following contents can be created:

```
<?xml version="1.0" encoding="utf-8"?>
<pets>
<pet name="Bit" type="Bunny" />
<pet name="Nibble" type="Bunny" />
<pet name="Stack" type="Bunny" />
<pet name="Queue" type="Bunny" />
<pet name="Heap" type="Bunny" />
<pet name="Null" type="Bunny" />
<pet name="Nigiri" type="Fish" />
<pet name="Sashimi II" type="Fish" />
<pet name="Kiwi" type="Lovebird" />
</pets>
```

Using XML Resources Programmatically

Now you can access this XML file as a resource programmatically in the following manner:

```
XmlResourceParser myPets =
getResources().getXml(R.xml.my_pets);
```

You can then use the parser of your choice to parse the XML.

Working with Raw Files

Your application can also include raw files as part of its resources. For example, your application might use raw files such as audio files, video files, and other file formats not supported by the `Android Asset Packaging Tool (aapt)`. The resource filename must be unique to the directory and should be descriptive because the filename (without the extension) becomes the name by which the resource is accessed.

References to Resources

You can reference resources instead of duplicating them. For example, your application might need to reference a single string resource in multiple string arrays. The most common use of resource references is in layout XML files, where layouts can reference any number of resources to specify attributes for layout colors, dimensions, strings, and graphics. Another common use is within style and theme resources.

Resources are referenced using the following format:
`@resource_type/variable_name`

Recall that earlier we had a string array of soup names. If we want to localize the soup listing, a better way to create the array is to create individual string resources for each soup name and then store the references to those string resources in the string array (instead of the text). To do this, we define the string resources in the `/res/strings.xml` file like this:

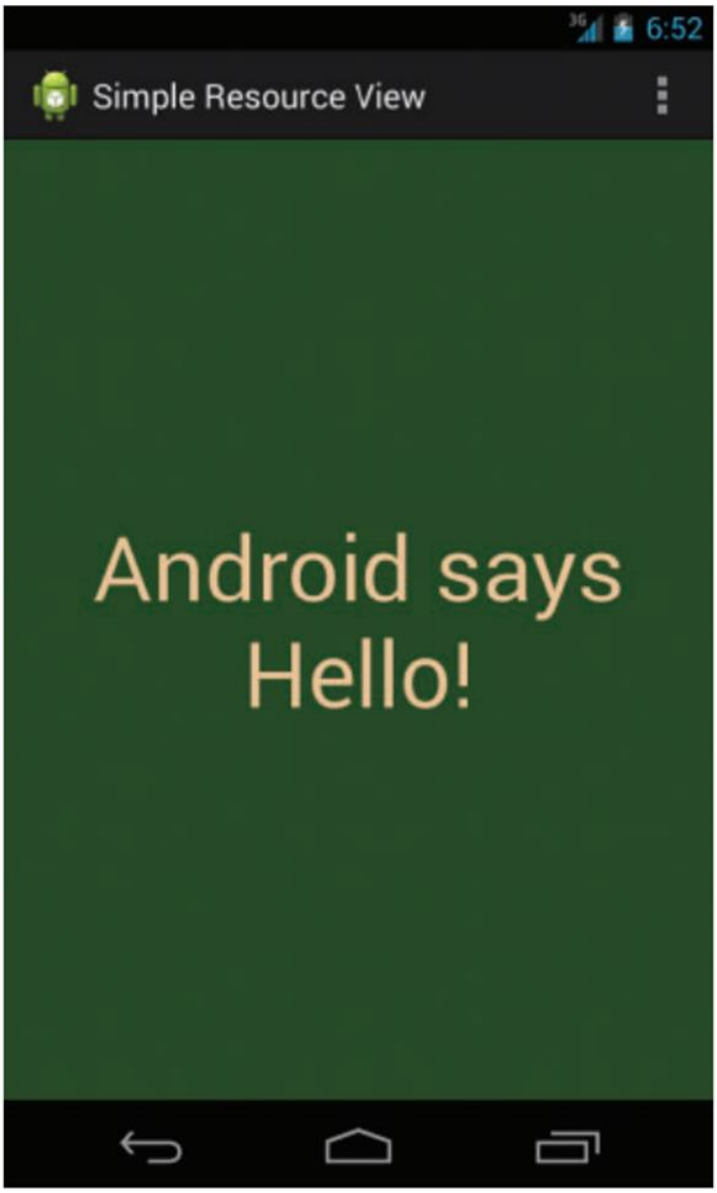
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">Application Name</string>
<string name="chicken_soup">Organic chicken noodle</string>
<string name="minestrone_soup">Veggie minestrone</string>
<string name="chowder_soup">New England clam chowder</string>
</resources>
<string-array name="soups">
<item>@string/minestrone_soup</item>
<item>@string/chowder_soup</item>
<item>@string/chicken_soup</item>
</string-array>
</resources>
```

Working With Layouts:

Much as Web designers use HTML, user interface designers can use XML to define Android application screen elements and layout.

A layout XML resource is where many different resources come together to form the definition of an Android application screen. Layout resource files are included in the **/res/layout/** directory and are compiled into the application package at build time.

Layout files might include many user interface controls and define the layout for an entire screen or describe custom controls used in other layouts.



Activity_simple_resource_view.xml

The **activity_simple_resource_view.xml** layout file that displays this screen references a number of other resources, including colors, strings, and dimension values, all of which were defined in the strings.xml, styles.xml, colors.xml, and dimens.xml resource files. The color resource for the screen background color and resources for a TextView control's color, string, and text size follow:

```
<?xml version="1.0" encoding="utf-8"?>

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

 android:orientation="vertical"

 android:layout_width="match_parent"

 android:layout_height="match_parent"
```

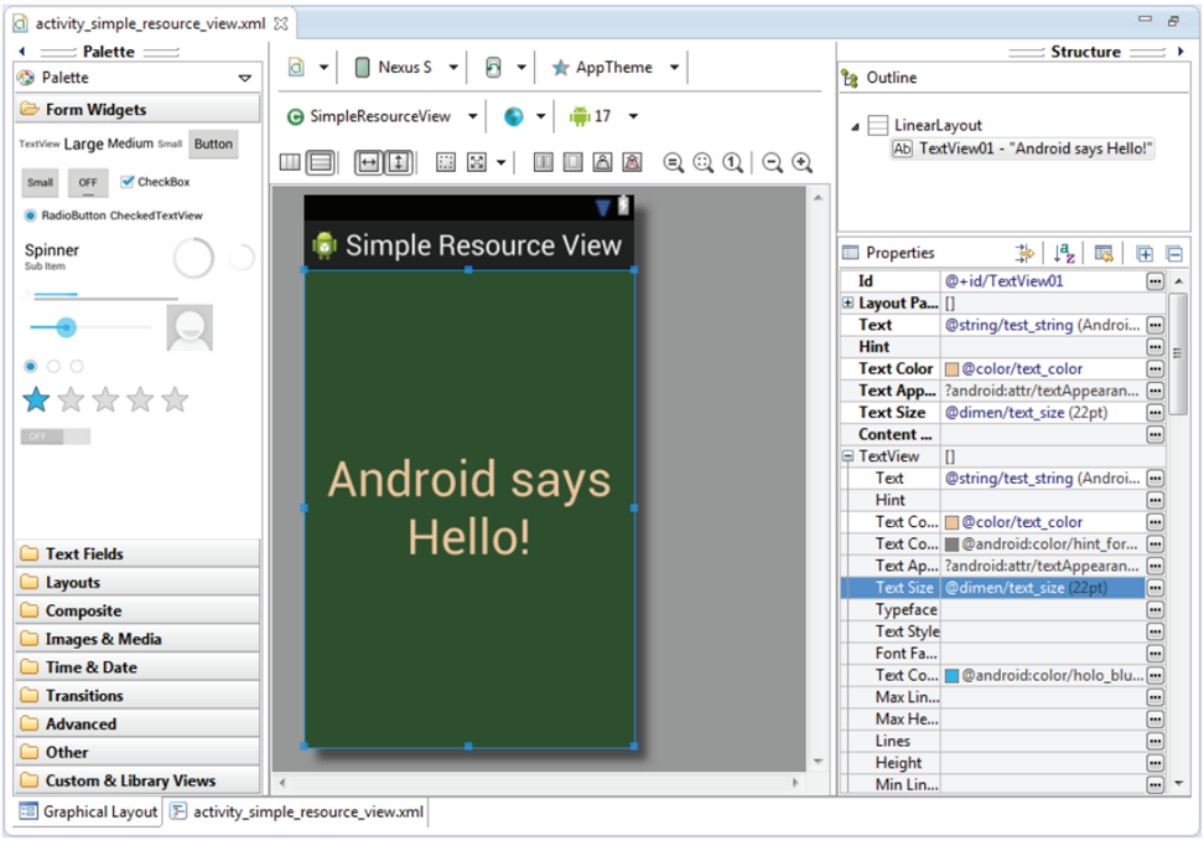
```
android:background="@color/background_color">
<TextView
android:id="@+id/TextView01"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:text="@string/test_string"
android:textColor="@color/text_color"
android:gravity="center"
android:textSize="@dimen/text_size" />
</LinearLayout>
```

Designing Layouts in the Android IDE:

Layouts can be designed and previewed in the Android IDE by using the resource editor functionality (see below figure).

If you click the project file **/res/layout/activity_ simple_resource_view.xml**, you see the Layout tab, which shows a preview of the Graphical Layout, and the

activity_simple_resource_view.xml tab, which shows the raw XML of the layout file. As with most user interface editors, the Android IDE works well for your basic layout needs, enables you to create user interface controls such as TextView and Button controls easily, and enables setting the controls' properties in the Properties pane.



- Now is a great time to get to know the layout resource editor, also known as the layout designer. Try creating a new Android project called ParisView.
- Navigate to the **/res/layout/activity_paris_view.xml** layout file and double-click it to open it in the editor. It's quite simple by default, with only a black (empty) rectangle and a String of text. Below in the Resource pane of the Android IDE perspective, you notice the Outline tab. This outline is the XML hierarchy of this layout file.
- By default, you see a LinearLayout. If you expand it, you see it contains one TextView control. Click the TextView control. You see that the Properties pane of the Android IDE perspective now has all the properties available for that object. If you scroll down to the property called text, you see that it's set to the string resource variable @string/hello_world
- Take a moment to switch to the activity_paris_view.xml tab.

- Notice that the properties you set are now in the XML. If you save and run your project in the emulator now, you will see similar results to what you see in the designer preview.
- Now select Images & Media within the Palette.
- Drag and drop the ImageView object within the preview editor.
- Now you have a new control in your layout.
- Drag two PNG (or JPG) graphics files into your **/res/drawable** project directory, naming them flag.png and background.png.
- Now go to the Outline view, browse the properties of your ImageView control, and then set the src property manually by typing @drawable/flag.
- Now, you see that the graphic shows up in your preview.

While we're at it, select the LinearLayout object and set its background property to the background drawable you added. If you save the layout file and run the application in the emulator (as shown in below Figure) or on the phone, you will see results much like those in the resource designer preview pane

Using Layout Resources Programmatically:

Objects within layouts, whether they are Button or ImageView controls, are all derived from the View class.

Here is how you would retrieve a TextView object named TextView01, called in an Activity class after the call to setContentView():

```
TextView txt = (TextView)findViewById(R.id.TextView01);
```

You can also access the underlying XML of a layout resource much as you would any XML file.

The following code retrieves the main.xml layout file for XML parsing:

```
XmlResourceParser myMainXml = getResources().getLayout(R.layout.activity_paris_view);
```

Developers can also define custom layouts with unique attributes.

Few layouts that we can use are:

1. Relative Layout
2. Linear Layout
3. Table Layout
4. List View
5. Grid View
6. Frame Layout
7. Text View
8. Constraint Layout.

Relative Layout: Relative Layout can eliminate nested view groups and keep your layout hierarchy flat, which improves performance.

EX:
```
<?xml version="1.0" encoding="utf-8"?>
 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent">
     <EditText
         android:id="@+id/name"
         android:layout_width="match_parent"
 </RelativeLayout>
```

Frame Layout: Android Frame layout is a View Group subclass that is used to specify the position of multiple views placed on top of each other to represent a single view screen.

EX:
```
<?xml version="1.0" encoding="utf-8"?>

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

    android:layout_height="match_parent"

    android:layout_width="match_parent">

          <TextView android:layout_height="wrap_content"

        android:layout_width="wrap_content"

        android:text="RightBottom"

        android:layout_gravity="right|bottom" />

          <TextView android:layout_height="wrap_content"

        android:layout_width="wrap_content"
```

```
    android:text="CenterBottom"

    android:layout_gravity="center|bottom" />

</FrameLayout>
```

Constraint Layout: Constraint Layout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.

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

Table Layout: Android Table Layout is a View Group subclass which is used to display the child View elements in rows and columns. It will arrange all the children elements into rows and columns and does not display any border lines in between rows, columns or cells.

List Layout: List layout customization provides the user with advanced, comprehensive control over the appearance and positions of columns in a SmartOffice list, as well as the sorting and grouping of list data.

Displaying Text to Users with TextView:

- **Configuring Layout and Sizing:** TextViews are used to display text on the screen. You can control their layout using properties like android:layout_width, android:layout_height, margins, and padding within your XML layout file.

Example (XML):

```
XML
<TextView
   android:id="@+id/myTextView"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="This is some text"
   android:textSize="20sp"
   android:layout_margin="10dp" />
```

- **Creating Contextual Links in Text:** You can make specific words or phrases in your TextView clickable using the android:autoLink property. This allows users to perform actions like opening a web link or dialing a phone number.

Retrieving Data from Users with Text Fields:

- **Retrieving Text Input Using EditText Controls:** EditText is a subclass of TextView that allows users to enter text. You can access the entered text using the getText().toString() method on the EditText object.

Example (Java):

```
EditText editText = (EditText) findViewById(R.id.myEditText);
```

String userInput = editText.getText().toString();

- **Constraining User Input with Input Filters:** You can control the type of text a user can enter using InputFilters. For example, InputType.TYPE_CLASS_NUMBER restricts input to numbers only.

Helping the User with Autocompletion

- **AutoCompleteTextView:** Extends EditText and offers suggestions as the user types. Ideal for:
    - Improving data entry speed and accuracy.
    - Guiding users with valid options.

```
AutoCompleteTextView autoCompleteTextView =
findViewById(R.id.autoCompleteTextView);

String[] suggestions = new String[]{"Apple", "Banana", "Cherry", "Orange"};

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, suggestions);

autoCompleteTextView.setAdapter(adapter);
```

Giving Users Choices Using Spinner Controls:

- **Using Spinner Controls:** A Spinner allows users to select an item from a dropdown list. You need to create an adapter to populate the list with data and set it on the Spinner.

Example (Java):

```
Spinner spinner = (Spinner) findViewById(R.id.mySpinner);
String[] options = {"Option 1", "Option 2", "Option 3"};
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, options);
spinner.setAdapter(adapter);
```

Allowing Simple User Selections with Buttons and Switches:

- **Using Basic Buttons:** Buttons allow users to trigger actions. You can set a click listener on a button to perform an action when clicked.

Example (Java):

```
Button button = (Button) findViewById(R.id.myButton);
button.setOnClickListener(new View.OnClickListener() {
  @Override
```

```
  public void onClick(View v) {
    // Your action here
  }
});
```

- **Using CheckBox and ToggleButton Controls:** CheckBoxes allow users to select multiple options, while ToggleButtons have two states (on/off).
- **Using RadioGroup and RadioButton:** RadioButtons are used within a RadioGroup to create a group where only one option can be selected at a time.

Retrieving Dates, Times, and Numbers from Users with Pickers:

- **Using DatePicker and TimePicker:** These provide dialogs for users to select dates and times.

Using Indicators to Display Progress and Activity to Users:

- **Indicating Progress with ProgressBar:** A ProgressBar is a visual representation of ongoing tasks. You can set its progress programmatically.

Example (Java):

```
ProgressBar progressBar = (ProgressBar) findViewById(R.id.myProgressBar);
progressBar.setProgress(50); // Set progress between 0 and 100
```

- **Indicating Activity with Activity Bars and Activity Circles:** These are visual cues that inform users about ongoing background processes.
- **Adjusting Progress with Seek Bars:** SeekBars allow users to select a value from a continuous range.