Chapter two
Android App User Interface development

In this chapter we will talk about the following main points.

- Anatomy of android application
- How to add and access resources in android studio
- Android Layouts and attributes
- Android - UI Controls, create and access UI controls.

Anatomy of Android Application

Folder, File & Description

- **Java**: This contains the **.java** source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
- **res/drawable-hdpi**: This is a directory for drawable objects that are designed for high-density screens.
- **res/layout**: This is a directory for files that define your app's user interface.
- **res/values**: This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.

- **AndroidManifest.xml**: This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
- **Build.gradle**: This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName
- **anim/:** XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class.
- **color/:** XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.

The Main Activity File

- The main activity code is a Java file **MainActivity.java**.
- This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application.
- Following is the default code generated by the application wizard for Hello World! application −package com.example.helloworld;

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

The Manifest File

- Whatever component you develop as a part of your application, you must declare all its components in a *manifest.xml* which resides at the root of the application project directory.
- This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS.
- For example, a default manifest file will look like as following file −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tutorialspoint7.myapplication">
   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      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>
```

The Strings File

- The strings.xml file is located in the res/values folder and it contains all the text that your application uses.
- For example, the names of buttons, labels, default text, and similar types of strings go into this file.
- This file is responsible for their textual content.
- For example, a default strings file will look like as following file −

```
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="hello_world">Hello world!</string>
   <string name="menu_settings">Settings</string>
   <string name="title_activity_main">MainActivity</string>
</resources>
```

The Layout File

- The **activity_main.xml** is a layout file available in *res/layout* directory, that is referenced by your application when building its interface.
- You will modify this file very frequently to change the layout of your application.
- For your "Hello World!" application, this file will have following content related to default layout

```
<RelativeLayout 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" >
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />
    </RelativeLayout>
```

Accessing Resources

- During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios.
- Accessing Resources in Code: When your Android application is compiled, a **R** class gets generated, which contains resource IDs for all the resources available in your **res/** directory.
- You can use R class to access that resource using sub-directory and resource name or directly resource ID.

To access *res/drawable/myimage.png* and set an ImageView you will use following code

- ImageView imageView = (ImageView) findViewById(R.id.myimageview);
- imageView.setImageResource(R.drawable.myimage);
- Here first line of the code make use of R.id.myimageview to get ImageView defined with id myimageview in a Layout file.
- Second line of code makes use of R.drawable.myimage to get an image with name myimage available in drawable sub-directory under /res.

How to add string resource and access it?

- Consider next example where *res/values/strings.xml* has following definition.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string  name="hello">Hello, World!</string>
</resources>
```

Now you can set the text on a TextView object with ID msg using a resource ID as follows

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Consider the following resource XML res/values/strings.xml file that includes a color resource and a string resource .

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

Now you can use these resources in the following layout file to set the text color and text string as follows

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

Android - UI Layouts

- The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling.
- View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.
- The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.
- A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.
- A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
    <TextView android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a TextView" />
      <Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a Button" />
      <!-- More GUI components go here  -->
  </LinearLayout>
```

- Once your layout has created, you can load the layout resource from your application code, in your Activity.onCreate() callback implementation as shown below

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
}
```

Android Layout Types

- There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.
- Linear Layout: LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
- Relative Layout: RelativeLayout is a view group that displays child views in relative positions.
- Table Layout: TableLayout is a view that groups views into rows and columns.
- Absolute Layout: AbsoluteLayout enables you to specify the exact location of its children.
- Frame Layout: The FrameLayout is a placeholder on screen that you can use to display a single view.
- List View: ListView is a view group that displays a list of scrollable items.
- Grid View: GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid

Layout Attributes

- Each layout has a set of attributes which define the visual properties of that layout.
- **android:id**: This is the ID which uniquely identifies the view.
- **android:layout_width**: This is the width of the layout.
- **android:layout_height**: This is the height of the layout
- **android:layout_marginTop**: This is the extra space on the top side of the layout.
- **android:layout_marginBottom**: This is the extra space on the bottom side of the layout.
- **android:layout_marginRight:** This is the extra space on the right side of the layout.
- **android:paddingLeft**: This is the left padding filled for the layout.
- width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

- **android:layout_marginRight:** This is the extra space on the right side of the layout.
- **android:paddingLeft**: This is the left padding filled for the layout.
- width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).
- You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −
- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

View Identification

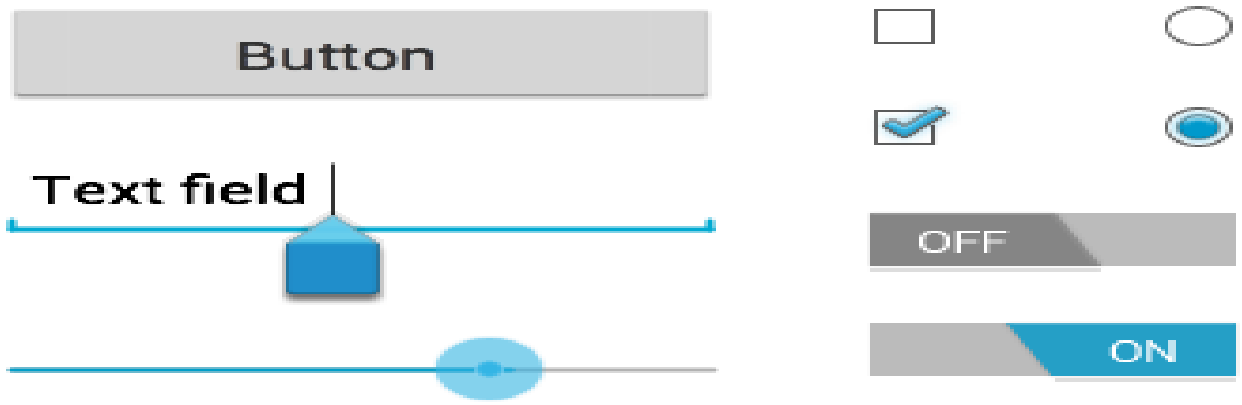
- A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is

  android:id="@+id/my_button“

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

Android - UI Controls

- Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.



UI Elements

Android UI Controls

- TextView: This control is used to display text to the user.
- EditText: EditText is a predefined subclass of TextView that includes rich editing capabilities.
- AutoCompleteTextView: The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
- Button : A push-button that can be pressed, or clicked, by the user to perform an action.
- ImageButton: An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.

- **CheckBox**: An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
- **ToggleButton**: An on/off button with a light indicator.
- **RadioButton**: The RadioButton has two states: either checked or unchecked.
- **RadioGroup** : A RadioGroup is used to group together one or more RadioButtons.
- **ProgressBarThe** ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
- **Spinner**: A drop-down list that allows users to select one value from a set.
- **TimePicker**: The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
- **DatePicker**: The DatePicker view enables users to select a date of the day.

Create UI Controls

- To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
    <TextView android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following −

```
TextView myText = (TextView) findViewById(R.id.text_id);
```