**UNIT-II**

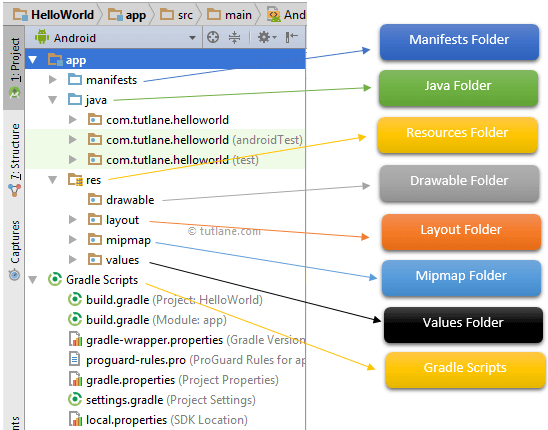
Android Application Design Essentials:

1. Anatomy of an Android applications,
2. Android terminologies,
3. Creating User Interfaces with basic views-
4. Application Context, Activities, Services, Intents, linking activities with Intents,, Receiving and Broadcasting Intents,
5. Android Manifest File and its common settings,
6. Using Intent Filter, Permissions.

* **Anatomy of an Android**

To implement android apps, Android Studio is the official IDE (Integrated Development Environment) which is freely provided by Google for android app development.

Once we setup android development environment using android studio and if we create a sample application using android studio, our project folder structure will be like as shown below. In case if you are not aware of creating an application using an android studio please check this Android Hello World App.

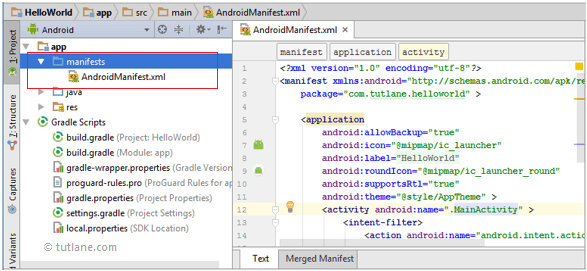


The Android project structure on the disk might be differs from the above representation. To see the actual file structure of the project, select Project from the Project dropdown instead of Android.

The android app project will contain different types of app modules, source code files, and resource files. We will explore all the folders and files in the android app.

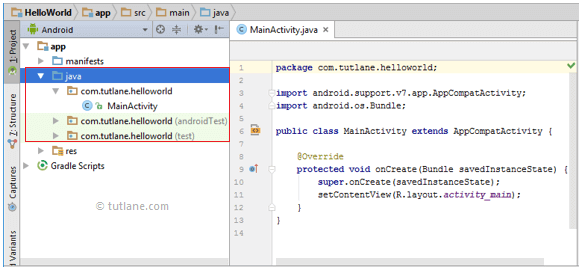
1. **Manifests Folder**

This folder will contain a manifest file (AndroidManifest.xml) for our android application. This manifest file will contain information about our application such as android version, access permissions, metadata, etc. of our application and its components. The manifest file will act as an intermediate between android OS and our application.Following is the structure of the manifests folder in the android application.



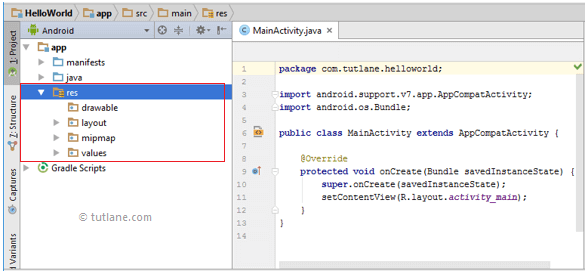
1. **Java Folder**

This folder will contain all the java source code (.java) files which we’ll create during the application development, including JUnit test code. Whenever we create any new project/application, by default the class file MainActivity.java will create automatically under the package name “com.tutlane.helloworld” like as shown below.



1. **res (Resources) Folder**

It’s an important folder that will contain all non-code resources, such as bitmap images, UI strings, XML layouts like as shown below.

Alt Text

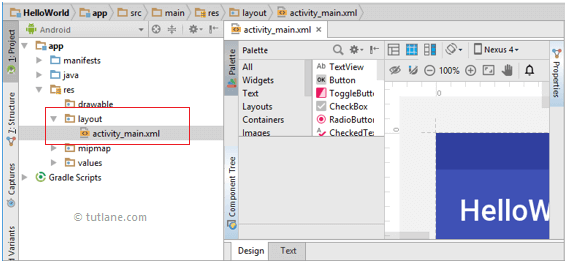
The res (Resources) will contain a different type of folders that are

**3.1. Drawable Folder (res/drawable)**

It will contain the different types of images as per the requirement of application. It’s a best practice to add all the images in a drawable folder other than app/launcher icons for the application development.

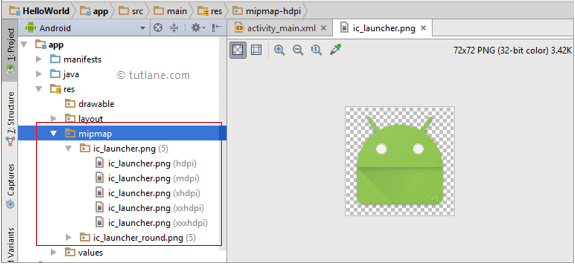
**3.2. Layout Folder (res/layout)**

This folder will contain all XML layout files which we used to define the user interface of our application. Following is the structure of the layout folder in the android application.



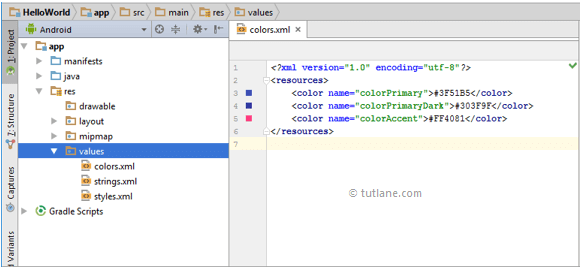
**3.3. Mipmap Folder (res/mipmap)**

This folder will contain app / launcher icons that are used to show on the home screen. It will contain different density type of icons such as hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi, to use different icons based on the size of the device.

Following is the structure of the mipmap folder in the android application. 

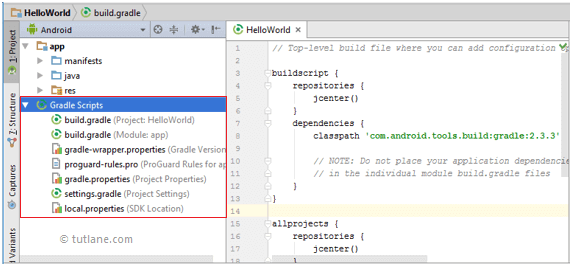
**3.4. Values Folder (res/values)**

This folder will contain various XML files, such as strings, colors, style definitions and a static array of strings or integers. Following is the structure of the values folder in android application.



**4. Gradle Scripts**

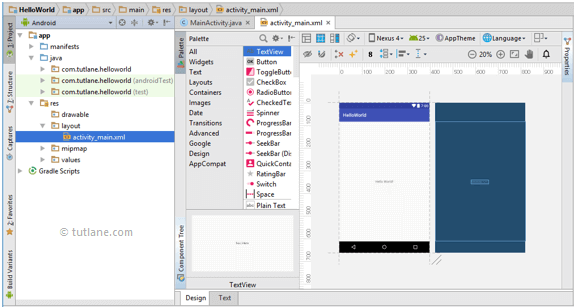
In android, Gradle means automated build system and by using this we can define a build configuration that applies to all modules in our application. In Gradle build.gradle (Project), and build.gradle (Module) files are useful to build configurations that apply to all our app modules or specific to one app module.Following is the structure of Gradle Scripts in the android application.

Alt Text

Following are the important files which we need to implement an app in android studio.

**5. Android Layout File (activity\_main.xml)**

The UI of our application will be designed in this file and it will contain Design and Text modes. It will exists in the layouts folder and the structure of activity\_main.xml file in Design mode like as shown below.



We can make required design modifications in activity\_main.xml file either using Design or Text modes. If we switch to Text mode activity\_main.xml file will contain a code like as shown below.

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

**<android.support.constraint.ConstraintLayout** xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.tutlane.helloworld.MainActivity"**>**

**<TextView**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello World!"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent" **/>**

**</android.support.constraint.ConstraintLayout>**

**6. Android Main Activity File (MainActivity.java)**

The main activity file in the android application is MainActivity.java and it will exist in the java folder. The MainActivity.java file will contain the java code to handle all the activities related to our app.

Following is the default code of MainActivity.java file which is generated by our HelloWorld application.

**package** **com.tutlane.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);

}

}

**7. Android Manifest File (AndroidManifest.xml)**

Generally, our application will contain multiple activities and we need to define all those activities in the AndroidManifest.xml file. In our manifest file, we need to mention the main activity for our app using the MAIN action and LAUNCHER category attributes in intent filters (). In case if we didn’t mention MAIN action or LAUNCHER category for the main activity, our app icon will not appear in the home screen’s list of apps.

Following is the default code of AndroidManifest.xml file which is generated by our HelloWorld application.

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

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

package="com.tutlane.helloworld" **>**

**<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>**

These are the main folders and files required to implement an application in android studio. If you want to see the actual file structure of the project, select Project from the Project dropdown instead of Android.

* Terminologies Correlated to Android

XML file

The preeminent file is used for the structure of an android project. It has complete information about all the components and packages. It initializes the API that is further used by an application.

View

It is the component of the User Interface that occupies the rectangular area on the screen.

Layout

It properly aligned the views on the screen.

Activity

Activity is a User interface screen through which the user interacts. Users have a right to place the UI elements in any way according to the Users choice.

Emulator

The emulator is the virtual device smartphone provided with an android studio. You can run your created application on the emulator and test its UI and function according to the needs.

Intent

It acts as a communicating object. You can establish a communication between two or more than two components as services, broadcast receivers. It is used to start and end the activity and services components.

Services

It is used to run the process even in the background. There is no defined UI for service. Any component can start the service and end the services. You can easily switch between the applications even if the services are running the background.

Content Provider

It implemented in two ways:

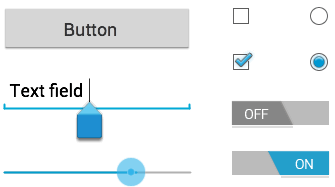
You can use implement the existing content provider in your application.

However, you can also create a new content provider that will provide or share the data with other applications.

* Android UI Controls (Textview, EditText, Radio Button, Checkbox)

In android **UI** or**input** controls are the interactive or View components that are used to design the user interface of an application. In android we have a wide variety of UI or input controls available, those are [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples), [Buttons](https://www.tutlane.com/tutorial/android/android-button-with-examples), [Checkbox](https://www.tutlane.com/tutorial/android/android-checkbox-with-examples), [Progressbar](https://www.tutlane.com/tutorial/android/android-progressbar-with-examples), [Spinners](https://www.tutlane.com/tutorial/android/android-spinner-dropdown-list-with-examples), etc.

Following is the pictorial representation of user interface (UI) or input controls in android application.



Generally, in android the user interface of an app is made with a collection of **View** and **ViewGroup** objects.

The **View** is a base class for all UI components in android and it is used to create interactive UI components such as [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples" \o "Android TextView Control with Examples" \t "_blank), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples" \o "Android EditText Control with Example" \t "_blank), [Checkbox](https://www.tutlane.com/tutorial/android/android-checkbox-with-examples), [Radio Button](https://www.tutlane.com/tutorial/android/android-radiobutton-with-examples), etc. and it is responsible for event handling and drawing.

The **ViewGroup** is a subclass of **View** and it will act as a base class for layouts and layout parameters. The ViewGroup will provide invisible containers to hold other Views or ViewGroups and to define the layout properties.

To know more about View and ViewGroup in android applications, check this [Android View and ViewGroup](https://www.tutlane.com/tutorial/android/android-view-and-viewgroup-with-examples).

In android, we can define a UI or input controls in two ways, those are

* Declare UI elements in XML
* Create UI elements at runtime

The android framework will allow us to use either or both of these methods to define our application’s UI.

Declare UI Elements in XML

In android, we can create layouts same as web pages in HTML by using default **Views** and **ViewGroups** in the XML file. The layout file must contain only one root element, which must be a **View** or **ViewGroup** object. Once we define the root element, then we can add additional layout objects or widgets as a child elements to build View hierarchy that defines our layout.

Following is the example of defining UI controls ([TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples" \o "Android TextView Control with Examples" \t "_blank), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples" \o "Android EditText Control with Example" \t "_blank), [Button](https://www.tutlane.com/tutorial/android/android-button-with-examples)) in the XML file (**activity\_main.xml**) using [LinearLayout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples" \o "Android Linear Layout with Examples" \t "_blank).

<?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">  
    <TextView  
        android:id="@+id/fstTxt"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Enter Name" />  
    <EditText  
        android:id="@+id/name"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:ems="10"/>  
    <Button  
        android:id="@+id/getName"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Get Name" />  
</LinearLayout>

In android, each input control is having a specific set of events and these events will be raised when the user performs particular action like, entering the text or touches the button.

Load XML Layout File from an Activity

Once we are done with the creation of layout with UI controls, we need to load the XML layout resource from our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) **onCreate()** callback method like as shown below.

protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);   
}

If you observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

Create UI Element at Runtime

If we want to create UI elements at runtime, we need to create our own custom **View** and **ViewGroup** objects programmatically with required layouts.

Following is the example of creating UI elements ([TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples" \o "Android TextView Control with Examples" \t "_blank), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples" \o "Android EditText Control with Example" \t "_blank), [Button](https://www.tutlane.com/tutorial/android/android-button-with-examples)) in [LinearLayout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples" \o "Android Linear Layout with Examples" \t "_blank) using custom **View** and **ViewGroup** objects in an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) programmatically.

public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        TextView textView1 = new TextView(this);  
        textView1.setText("Name:");  
        EditText editText1 = new EditText(this);  
        editText1.setText("Enter Name");  
        Button button1 = new Button(this);  
        button1.setText("Add Name");  
        LinearLayout linearLayout = new LinearLayout(this);  
        linearLayout.addView(textView1);  
        linearLayout.addView(editText1);  
        linearLayout.addView(button1);  
        setContentView(linearLayout);  
    }  
}

By creating a custom **View** and **ViewGroups** programmatically, we can define UI controls in layouts based on our requirements in android applications.

Width and Height

When we define a UI controls in a layout using an XML file, we need to set width and height for every **View** and **ViewGroup** elements using **layout\_width** and **layout\_height** attributes.

Following is the example of setting width and height for **View** and **ViewGroup** elements in the XML layout file.

<?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">  
    <TextView  
        android:id="@+id/fstTxt"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Enter Name" />  
</LinearLayout>

If you observe above example, we used different values to set layout\_width and layout\_height, those are

* match\_parent
* wrap\_content

If we set value **match\_parent**, then the **View** or **ViewGroup** will try to match with parent width or height.

If we set value **wrap\_content**, then the **View** or **ViewGroup** will try to adjust its width or height based on the content.

Android Different Types of UI Controls

We have a different type of UI controls available in android to implement the user interface for our android applications.

Following are the commonly used UI or input controls in android applications.

* [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples)
* [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples)
* [AutoCompleteTextView](https://www.tutlane.com/tutorial/android/android-autocompletetextview-with-examples)
* [Button](https://www.tutlane.com/tutorial/android/android-button-with-examples)
* [ImageButton](https://www.tutlane.com/tutorial/android/android-imagebutton-with-examples)
* [ToggleButton](https://www.tutlane.com/tutorial/android/android-toggle-button-with-examples)
* [CheckBox](https://www.tutlane.com/tutorial/android/android-checkbox-with-examples)
* [RadioButton](https://www.tutlane.com/tutorial/android/android-radiobutton-with-examples)

Android TextView

In android, **TextView** is a user interface control that is used to display the text to the user.

Android EditText

In android, **EditText** is a user interface control which is used to allow the user to enter or modify the text.

Android AutoCompleteTextView

In android, **AutoCompleteTextView** is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.

Android Button

In android, **Button** is a user interface control that is used to perform an action when the user clicks or tap on it.

Android Image Button

In android, **Image Button** is a user interface control that is used to display a button with an image to perform an action when the user clicks or tap on it.

Generally, the Image button in android looks similar as regular Button and perform the actions same as regular button but only difference is for image button we will add an image instead of text.

Android Toggle Button

In android, **Toggle Button** is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.

Android CheckBox

In android, **Checkbox** is a two-states button that can be either checked or unchecked.

Android Radio Button

In android, **Radio Button** is a two-states button that can be either checked or unchecked and it cannot be unchecked once it is checked.