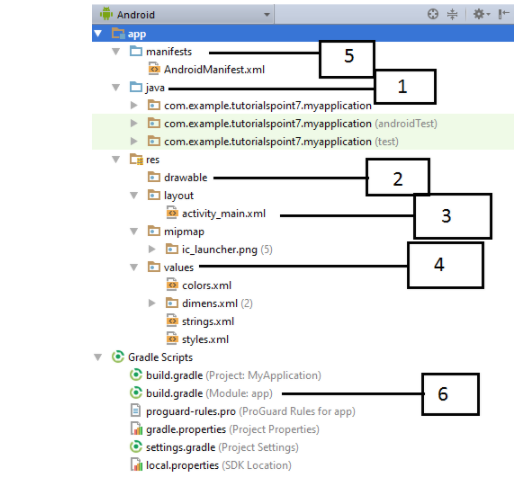
**Android Runtim**e

A key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language.

The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.



By default, Android Studio displays our project files in the Android project view, as shown in the above image. This view is formed by modules to provide quick access to our project's key source files.

These build files are visible to the top-level under Gradle Scripts. And the app module contains the following folders:

* **manifests:** It contains the AndroidManifest.xml file.
* **java:** It contains the source code of Java files, including the JUnit test code.
* **res:** It contains all non-code resources, UI strings, XML layouts, and bitmap images.

We will see the actual file structure of the project by selecting the **Project** from the **Project dropdown**.

|  |
| --- |
| **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 colours 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 |

## 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

**Every Android app must include an AndroidManifest.xml file describing functionality**

• **The manifest specifies:**

– App’s Activities, Services, etc.

– Permissions requested by app

– Minimum API required

– Hardware features required, e.g., camera with Autofocus

Activity: key building block of Android apps

• Extend Activity class, override onCreate(), onPause(), onResume() methods

• Dalvik VM can stop any Activity without warning, so saving state is important!

• Activities need to be “responsive”, otherwise Android shows user “App Not Responsive” warning:

– Place lengthy operations in Runnable Threads, AsyncTasks

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.

<?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>

Here <application>...</application> tags enclosed the components related to the application. Attribute android:icon will point to the application icon available under res/drawable-hdpi. The application uses the image named ic\_launcher.png located in the drawable folders

The <activity> tag is used to specify an activity and android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

The action for the intent filter is named android.intent.action.MAIN to indicate that this activity serves as the entry point for the application. The category for the intent-filter is named android.intent.category.LAUNCHER to indicate that the application can be launched from the device's launcher icon.

The @string refers to the strings.xml file explained below. Hence, @string/app\_name refers to the app\_name string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application.

**Apps use four main components:**

– **Activity:** A “single screen” that’s visible to user

– **Service:** Long-running background “part” of app (not

separate process or thread)

– **ContentProvider:** Manages app data (usually stored in

database) and data access for queries

– **BroadcastReceiver:** Component that listens for

particular Android system “events”, e.g., “found

wireless device”, and responds accordingly

Following is the list of tags which you will use in your manifest file to specify different Android application components −

<activity>elements for activities

<service> elements for services

<receiver> elements for broadcast receivers

<provider> elements for content providers

## Organize resource in Android Studio



**VIEW**

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.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main\_layout.xml** which is located in the res/layout folder of your project.

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"**

Following is a brief description of @ and + signs −

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);**

**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.

**Example**

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);**

**Android UI Controls**

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

TextView

[EditText](https://www.tutorialspoint.com/android/android_edittext_control.htm)

[AutoCompleteTextView](https://www.tutorialspoint.com/android/android_autocompletetextview_control.htm)

[Button](https://www.tutorialspoint.com/android/android_button_control.htm)

[ImageButton](https://www.tutorialspoint.com/android/android_imagebutton_control.htm)

[ToggleButton](https://www.tutorialspoint.com/android/android_togglebutton_control.htm)

[CheckBox](https://www.tutorialspoint.com/android/android_checkbox_control.htm)

[RadioButton](https://www.tutorialspoint.com/android/android_radiobutton_control.htm)

[RadioGroup](https://www.tutorialspoint.com/android/android_radiogroup_control.htm)

[Spinner](https://www.tutorialspoint.com/android/android_spinner_control.htm)

[DatePicker](https://www.tutorialspoint.com/android/android_datepicker_control.htm)

[TimePicker](https://www.tutorialspoint.com/android/android_timepicker_control.htm)

|  |  |
| --- | --- |
| **Sr.No.** | **UI Control & Description** |
| 1 | [TextView](https://www.tutorialspoint.com/android/android_textview_control.htm)  This control is used to display text to the user. |
| 2 | [EditText](https://www.tutorialspoint.com/android/android_edittext_control.htm)  EditText is a predefined subclass of TextView that includes rich editing capabilities. |
| 3 | [AutoCompleteTextView](https://www.tutorialspoint.com/android/android_autocompletetextview_control.htm)  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. |
| 4 | [Button](https://www.tutorialspoint.com/android/android_button_control.htm)  A push-button that can be pressed, or clicked, by the user to perform an action. |
| 5 | [ImageButton](https://www.tutorialspoint.com/android/android_imagebutton_control.htm)  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. |
| 6 | [CheckBox](https://www.tutorialspoint.com/android/android_checkbox_control.htm)  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. |
| 7 | [ToggleButton](https://www.tutorialspoint.com/android/android_togglebutton_control.htm)  An on/off button with a light indicator. |
| 8 | [RadioButton](https://www.tutorialspoint.com/android/android_radiobutton_control.htm)  The RadioButton has two states: either checked or unchecked. |
| 9 | [RadioGroup](https://www.tutorialspoint.com/android/android_radiogroup_control.htm)  A RadioGroup is used to group together one or more RadioButtons. |
| 10 | [ProgressBar](https://www.tutorialspoint.com/android/android_progressbar.htm)  The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background. |
| 11 | [Spinner](https://www.tutorialspoint.com/android/android_spinner_control.htm)  A drop-down list that allows users to select one value from a set. |
| 12 | [TimePicker](https://www.tutorialspoint.com/android/android_timepicker_control.htm)  The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. |
| 13 | [DatePicker](https://www.tutorialspoint.com/android/android_datepicker_control.htm)  The DatePicker view enables users to select a date of the day. |