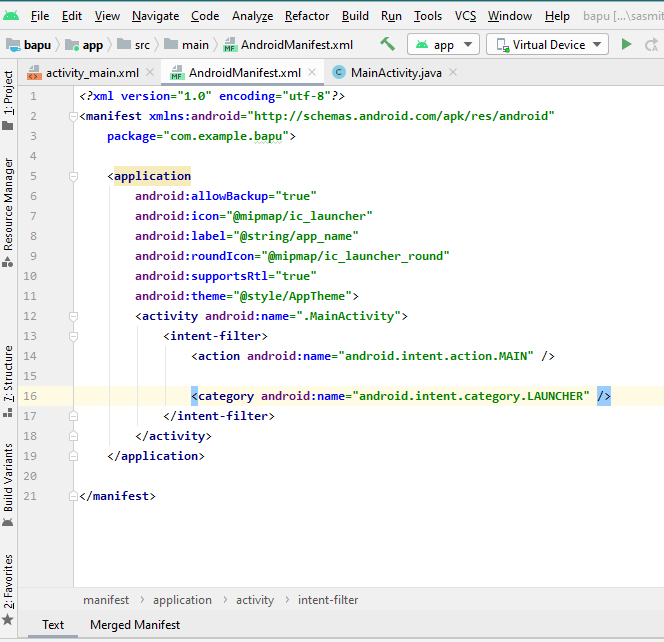
Android:

* Manifest: XML file that describes the structure of an Android app. This file also records permission settings (where applicable) and other details about the app.



An XML declaration is made up of as many as three name/value pairs, syntactically identical to attributes. The three attributes are a mandatory version attribute and optional encoding and standalone attributes. The order of these attributes within an XML declaration is fixed.

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

The XML declaration begins with the character sequence <?xml and ends with the character sequence ?>.

**UTF** stands for UCS Transformation Format, and UCS itself **means** Universal Character Set. The number **8** or 16 refers to the number of bits used to represent a character. They are either **8**(1 to 4 bytes) or 16(2 or 4 bytes). For the documents without **encoding** information, **UTF**-**8** is set by default.

<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.myapplication"**>

In XML, xmlns declares a Namespace. This is just the XML Name Space declaration. **We use this Name Space in order to specify that the attributes listed below, belongs to Android.** Thus they starts with "android:" Instead of calling android:id, the xml will use [http://schemas.android.com/apk/res/android:id](http://schemas.android.com/apk/res/android%3Aid) to be unique.

**package="com.example.myapplication"**>

Every Android app has a unique application ID that looks like a Java package name, such as com.example.myapp. This ID uniquely identifies your app on the device and in Google Play Store.

There are two concepts of backup for the manifest:

android:allowBackup="true"

"android:allowBackup" allows to backup and restore via adb(Your application should be backed up)

**android:icon="@mipmap/ic\_launcher"**

The default application that android provides…res-mipmap---Ic\_launcher(6)

android:label="@string/app\_name"

Name of the application coming with the icon.Values-Strings.xml then u ll get the app name as

<**resources**>  
 <**string name="app\_name"**>My 2nd Project</**string**>  
</**resources**>

**android:roundIcon="@mipmap/ic\_launcher\_round"**

You ll get round icon through res folder.

**android:supportsRtl="true"**

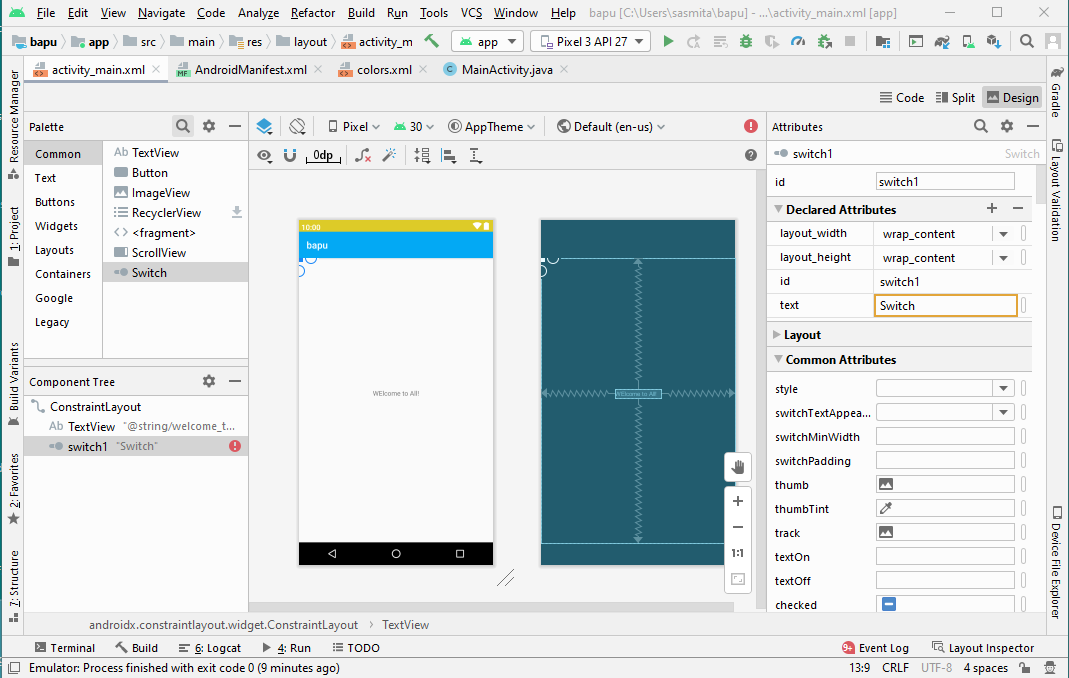
Declares whether your application is willing to support right-to-left (RTL) layouts. If set to true and targetSdkVersion is set to 17 or higher, various RTL APIs will be activated and used by the system so your app can display RTL layouts

**android:theme="@style/AppTheme"**>

so we can get it by

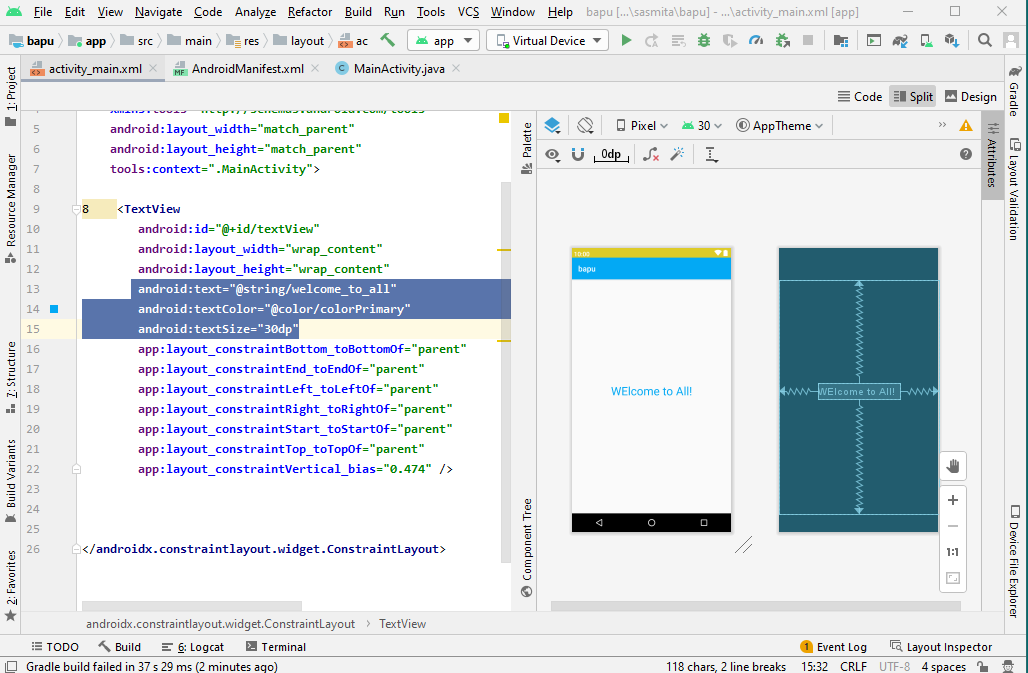
values-style.xml

<**resources**>  
 *<!-- Base application theme. -->* <**style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar"**>  
 *<!-- Customize your theme here. -->* <**item name="colorPrimary"**>@android:color/holo\_orange\_light</**item**>  
 <**item name="colorPrimaryDark"**>#CDDC39</**item**>  
 <**item name="colorAccent"**>@android:color/holo\_red\_dark</**item**>  
 </**style**>  
  
</**resources**>



<**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
</**activity**>

Note:



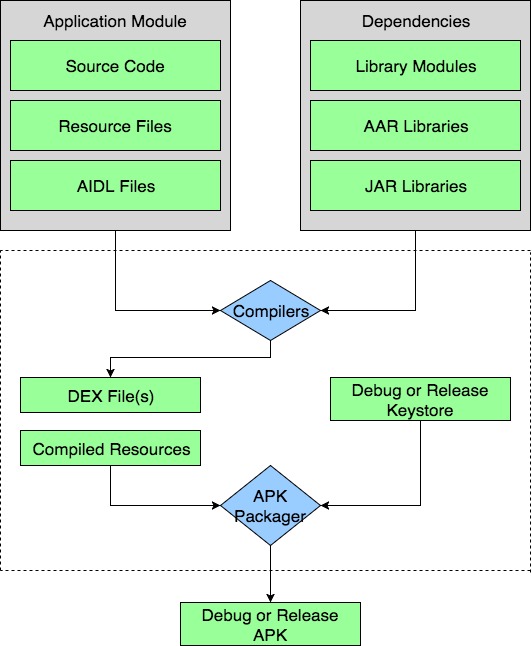
Gradle: **Gradle** is a build system, which is responsible for code compilation, testing, deployment and conversion of the code into .dex files and hence running the app on the device.

As Android Studio comes with Gradle system pre-installed, there is no need to install additional runtime softwares to build our project. Whenever you click on **Run** button in android studio, a gradle task automatically triggers and starts building the project and after gradle completes its task, app starts running in AVD or in the connected device.

A build system like Gradle is not a compiler, linker etc, but it controls and supervises the operation of compilation, linking of files, running test cases, and eventually bundling the code into an apk file for your Android Application.

There are two build.gradle files for every android studio project of which, one is for **application** and other is for **project level(module level) build files**.

The build process works as shown in the below diagram.



In the build process, the compiler takes the source code, resources, external libraries JAR files and AndroidManifest.xml(which contains the meta-data about the application) and convert them into .dex(Dalvik Executable files) files, which includes bytecode. That bytecode is supported by all android devices to run your app. Then **APK Manager** combines the .dex files and all other resources into single **apk** file. **APK Packager** signs debug or release apk using respective debug or release keystore.

**Debug apk** is generally used for testing purpose or we can say that it is used at development stage only. When your app is complete with desired features and you are ready to publish your application for external use then you require a **Release apk** signed using a release keystore.

Now lets shed some light on the gradle files.

**setting.gradle**

The **setting.gradle**(Gradle setting) file is used to **specify all the modules used in your app**.

**build.gradle (project level)**

The *Top level* (module) **build.gradle** file is project level build file, which defines build configurations at project level. This file applies configurations to all the modules in android application project.

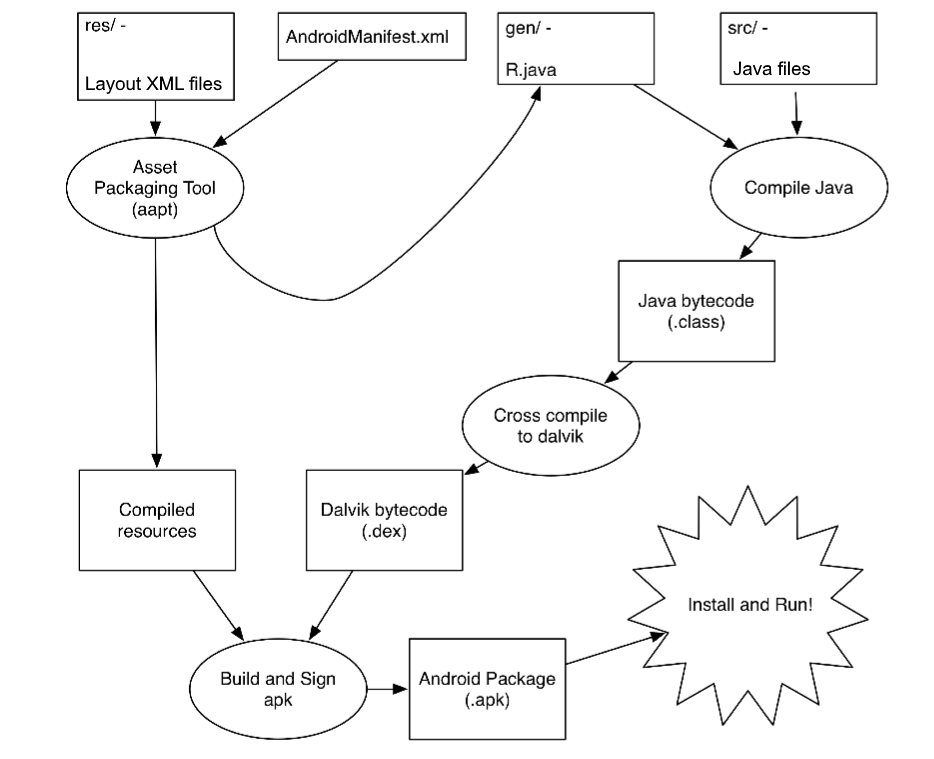
**build.gradle (application level)**

The *Application level* **build.gradle** file is located in each module of the android project. This file includes your package name as applicationID, version name(apk version), version code, minimum and target sdk for a specific application module. When you are including external libraries(not the jar files) then you need to mention it in the app level gradle file to include them in your project as dependencies of the application.

**Note:** If a certain application is developed in variations for individual modules like, Smart Phone, Tablet or TV then separate gradle files must to be created for all.

You can even start your gradle system through **command line tool**. Following commands are used for it:

* ./gradlew build - (build project)
* ./gradlew clean build - (build project complete scratch)
* ./gradlew clean build - (run the test)
* ./gradlew wrapper - (to see all the available tasks)

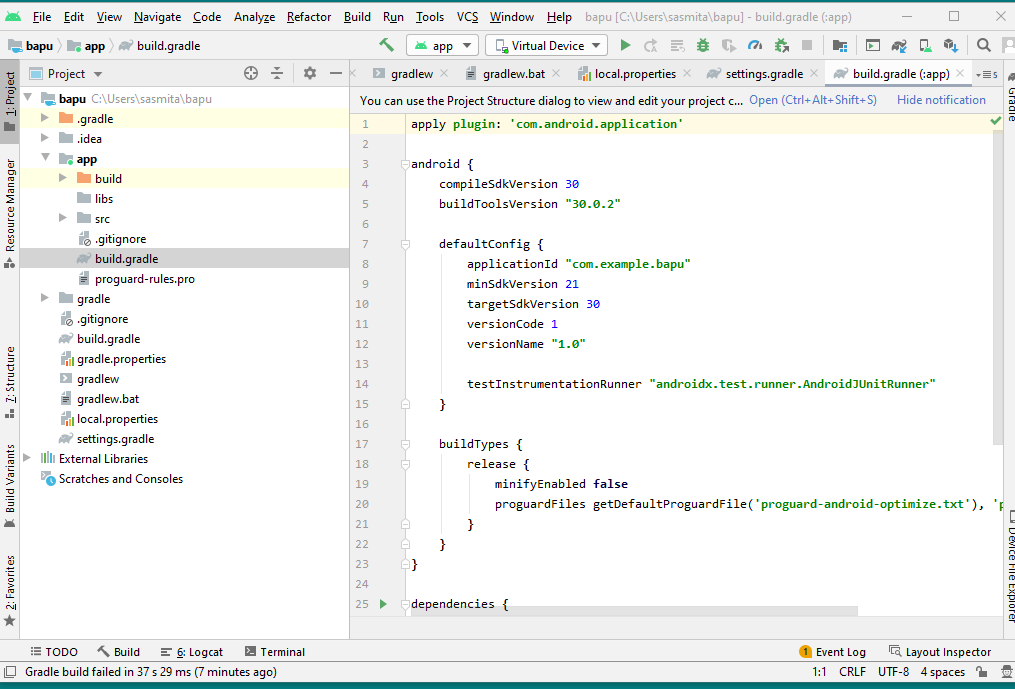


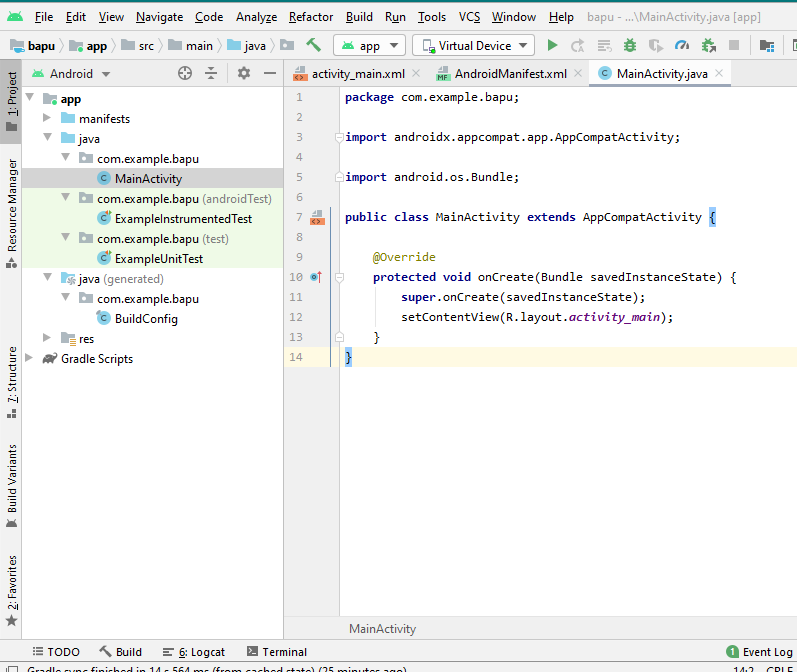
**ART - Android Runtime**

The **Dalvik Virtual Machine is dead**. Yes, Google stopped using it in 2014, although you will find most of the Android tutorials online, still not updated, but please be informed that Dalvik Virtual Machine is not used in Android anymore.

The new runtime is known as ART or Android Runtime which is very well compatible with its predecessor Dalvik, but do comes in with a lot of new features like:

* **Ahead-of-Time compilation**
* **Improved Garbage collection**
* Improved Debugging and diagnostics.





**onCreate(Bundle savedInstanceState) Function in Android:**

* When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.
* When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.
* After Orientation changed then onCreate(Bundle savedInstanceState) will call and recreate the activity and load all data from savedInstanceState.
* Basically Bundle class is used to stored the data of activity whenever above condition occur in app.
* onCreate() is not required for apps. But the reason it is used in app is because that method is the best place to put initialization code.
* You could also put your initialization code in onStart() or onResume() and when you app will load first, it will work same as in onCreate().
* **setContentView( R.layout.activity\_main );**
* This is a Java method called **setContentView**. It sets the XML file you want as your main layout when the app starts. In between round brackets, you need the name and location of your layout file. The letter R in the round brackets is short for **res**. This is the resource folder where those drawable, layout, mipmap, and values folders are. The **layout.activity\_main** part points to the activity\_main XML file, which is in the layout folder of res.