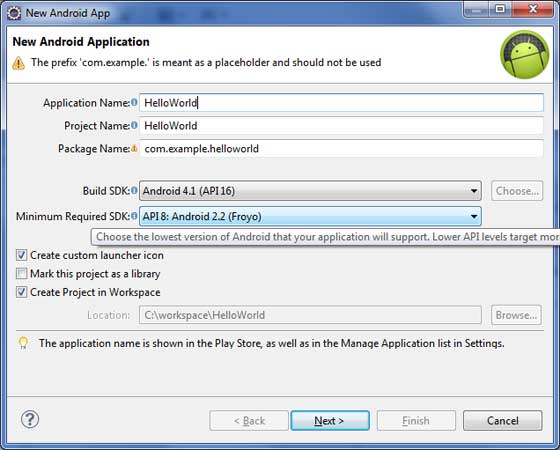
# Android Hello World Example

Let us start actual programming with Android Framework. Before you start writing your first example using Android SDK, you have to make sure that you have setup your Android development environment properly as explained in [Android - Environment Setup](http://www.tutorialspoint.com/android/android_environment_setup.htm) tutorial. I also assume that you have a little bit working knowledge with Eclipse IDE.

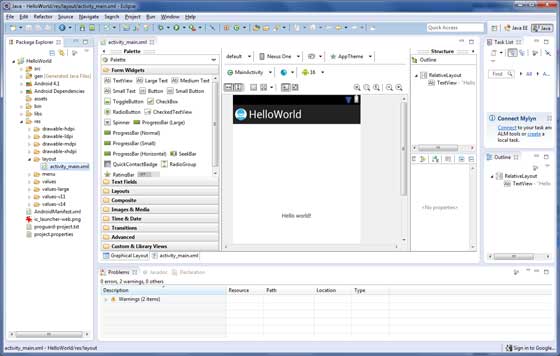
So let us proceed to write a simple Android Application which will print "Hello World!".

Create Android Application

The first step is to create a simple Android Application using Eclipse IDE. Follow the option **File -> New -> Project** and finally select **Android New Application** wizard from the wizard list. Now name your application as **HelloWorld** using the wizard window as follows:

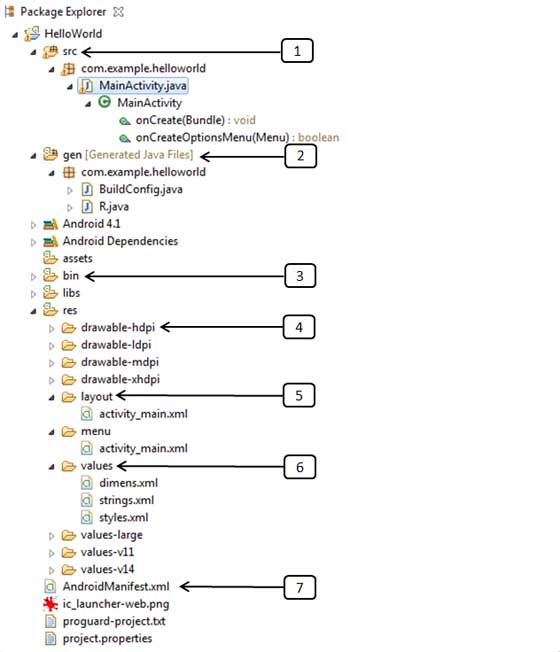


Next, follow the instructions provided and keep all other entries as default till the final step. Once your project is created successfully, you will have following project screen:



Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project:



|  |  |
| --- | --- |
| **S.N.** | **Folder, File & Description** |
| 1 | **src** 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. |
| 2 | **gen** This contains the **.R** file, a compiler-generated file that references all the resources found in your project. You should not modify this file. |
| 3 | **bin** This folder contains the Android package files **.apk** built by the ADT during the build process and everything else needed to run an Android application. |
| 4 | **res/drawable-hdpi** This is a directory for drawable objects that are designed for high-density screens. |
| 5 | **res/layout** This is a directory for files that define your app's user interface. |
| 6 | **res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions. |
| 7 | **AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components. |

Following section will give a brief overview few of the important application files.

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.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.view.MenuItem;

import android.support.v4.app.NavUtils;

public class MainActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.activity\_main, menu);

return true;

}

}

Here, *R.layout.activity\_main* refers to the *activity\_main.xml* file located in the *res/layout* folder. The*onCreate()* method is one of many methods that are fi red when an activity is loaded.

The Manifest File

Whatever component you develop as a part of your application, you must declare all its components in a *manifest* file called **AndroidManifest.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:

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

package="com.example.helloworld"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk

android:minSdkVersion="8"

android:targetSdkVersion="15" />

<application

android:icon="@drawable/ic\_launcher"

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

android:theme="@style/AppTheme" >

<activity

android:name=".MainActivity"

android:label="@string/title\_activity\_main" >

<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 fi le, which is "HelloWorld". Similar way, other strings get populated in the application.

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

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 R File

The **gen/com.example.helloworld/R.java** file is the glue between the activity Java files like*MainActivity.java* and the resources like *strings.xml*. It is an automatically generated file and you should not modify the content of the R.java file. Following is a sample of R.java file:

/\* AUTO-GENERATED FILE. DO NOT MODIFY.

\*

\* This class was automatically generated by the

\* aapt tool from the resource data it found. It

\* should not be modified by hand.

\*/

package com.example.helloworld;

public final class R {

public static final class attr {

}

public static final class dimen {

public static final int padding\_large=0x7f040002;

public static final int padding\_medium=0x7f040001;

public static final int padding\_small=0x7f040000;

}

public static final class drawable {

public static final int ic\_action\_search=0x7f020000;

public static final int ic\_launcher=0x7f020001;

}

public static final class id {

public static final int menu\_settings=0x7f080000;

}

public static final class layout {

public static final int activity\_main=0x7f030000;

}

public static final class menu {

public static final int activity\_main=0x7f070000;

}

public static final class string {

public static final int app\_name=0x7f050000;

public static final int hello\_world=0x7f050001;

public static final int menu\_settings=0x7f050002;

public static final int title\_activity\_main=0x7f050003;

}

public static final class style {

public static final int AppTheme=0x7f060000;

}

}

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>

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The *TextView* is an Android control used to build the GUI and it have various attribuites like *android:layout\_width*,*android:layout\_height* etc which are being used to set its width and height etc. The *@string* refers to the strings.xml file located in the res/values folder. Hence, @string/hello\_world refers to the hello string defined in the strings.xml fi le, which is "Hello World!".

Running the Application

Let's try to run our **Hello World!** application we just created. I assume you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:

