



ANDROID

application development

Created By:

**Gara Anil
Lolugu Sivendra
Kalluri Chandu
Kottubadi Jagadeesh
Maddili Ganesh**

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
1. ANDROID – OVERVIEW	1
What is Android?	1
Features of Android	1
Android Applications	2
2. ANDROID – ENVIRONMENT SETUP	3
Step 1 - Setup Java Development Kit (JDK).....	3
Step 2 - Setup Android SDK	4
Step 3 - Setup Eclipse IDE.....	5
Step 4 - Setup Android Development Tools (ADT) Plugin	6
Step 5 - Create Android Virtual Device.....	8
3. ANDROID – ARCHITECTURE	10
Linux kernel	10
Libraries.....	10
Android Runtime	11
Application Framework	11
Applications.....	11
4. ANDROID – APPLICATIONS COMPONENT	12
Activities.....	12
Services.....	12
Broadcast Receivers.....	13

Content Providers	13
Additional Components	13
5. ANDROID – HELLO WORLD EXAMPLE	15
Create Android Application.....	15
Anatomy of Android Application	16
The Main Activity File	18
The Manifest File	19
The Strings File.....	20
The R File	21
The Layout File.....	22
Running the Application	23
6. ANDROID – ORGANIZING & ACCESSING THE RESOURCES	24
Organize Resources.....	24
Alternative Resources.....	25
Accessing Resources	27
Accessing Resources in Code.....	27
Accessing Resources in XML.....	28
7. ANDROID – ACTIVITIES	30
8. ANDROID – SERVICES	36
9. ANDROID – BROADCAST RECEIVERS	46
Creating the Broadcast Receiver	46
Registering Broadcast Receiver.....	46
Broadcasting Custom Intents	48
10. ANDROID – CONTENT PROVIDERS	55
Content URIs.....	55

Create Content Provider	56
11. ANDROID – FRAGMENTS	70
Fragment Life Cycle.....	71
How to use Fragments?	72
12. ANDROID – INTENTS & FILTERS	80
Intent Objects	80
Action	81
Data.....	81
Category	82
Extras.....	82
Flags	82
Component Name.....	82
Types of Intents	82
Explicit Intents.....	82
Implicit Intents.....	83
Intent Filters	88
13. ANDROID – UI LAYOUTS.....	98
Android Layout Types	99
Layout Attributes.....	99
View Identification	102
14. ANDROID – UI CONTROLS.....	103
Android UI Controls	103
Create UI Controls.....	105
15. ANDROID – EVENT HANDLING.....	106
Event Listeners & Event Handlers	106

Event Listeners Registration.....	107
Event Handling Examples.....	107
Registration Using the Activity Implements Listener Interface.....	112
Registration Using Layout file activity_main.xml	115
Exercise:	118
 17. ANDROID – STYLES & THEMES.....	119
Defining Styles	119
Using Styles	120
Style Inheritance.....	120
Android Themes	122
Default Styles & Themes	122
 18. ANDROID – CUSTOM COMPONENTS	123
Creating a Simple Custom Component.....	123
Instantiate using code inside activity class.....	124
Instantiate using Layout XML file	124
Custom Component with Custom Attributes	125
Step 1	126
Step 2	126
Step 3	127
 19. ANDROID – DRAG & DROP	129
The Drag/Drop Process	129
The DragEvent Class.....	130
Listening for Drag Event.....	131
Starting a Drag Event	131
 20. ANDROID – NOTIFICATIONS	138
Create and Send Notifications.....	139

The NotificationCompat.Builder Class.....	140
Big View Notification	151
21. ANDROID – LOCATION-BASED SERVICES	154
The Location Object	154
Get the Current Location	156
Get the Updated Location	157
Location Quality of Service	157
Displaying a Location Address.....	158
Install the Google Play Services SDK	158
Create Android Application.....	159
22. ANDROID – SENDING EMAIL.....	170
Intent Object - Action to send Email	170
Intent Object - Data/Type to send Email.....	170
Intent Object - Extra to send Email	170
23. ANDROID – SENDING SMS	178
Using SmsManager to send SMS.....	178
Using Built-in Intent to send SMS.....	185
Intent Object - Action to send SMS	185
Intent Object - Data/Type to send SMS.....	185
Intent Object - Extra to send SMS	185
24. ANDROID – PHONE CALLS.....	192
Intent Object - Action to make Phone Call	192
Intent Object - Data/Type to make Phone Call.....	192
25. ANDROID – PUBLISHING ANDROID APPLICATION	199
Export Android Application.....	200

Google Play Registration.....	204
26. ANDROID – ALERT DIALOG TUTORIAL	206
27. ANDROID – ANIMATIONS	220
Tween Animation	220
Zoom in animation.....	221
28. ANDROID – AUDIO CAPTURE	235
29. ANDROID – AUDIO MANAGER	248
30. ANDROID – AUTOCOMPLETE	261
31. ANDROID – BEST PRACTICES.....	271
Best Practices - User input	271
Best Practices - Performance	272
Best Practices - Security and privacy	272
32. ANDROID – BLUETOOTH	283
33. ANDROID – CAMERA	297
Using existing android camera application in our application	297
Directly using Camera API Provided by Android in our Application	306
34. ANDROID – CLIPBOARD	317
Copying data.....	317
Pasting data	318
35. ANDROID – CUSTOM FONTS.....	328
36. ANDROID – DATA BACKUP	333
Test your BackupAgent	337
37. ANDROID – DEVELOPER TOOLS	339



SDK tools	339
Android.....	340
DDMS	340
Capturing ScreenShot	344
Sqlite3	344
Platform tools.....	345
 38. ANDROID – EMULATOR	 346
Creating AVD	346
Creating Snapshots	346
Changing Orientation.....	346
Emulator Commands.	347
Emulator - Sending SMS.....	348
Emulator - Making Call.....	350
Emulator - Transferring files	351
 39. ANDROID – FACEBOOK INTEGRATION	 352
Integrating Facebook SDK	352
Intent share	354
 40. ANDROID – GESTURES	 364
Handling Pinch Gesture.....	364
 41. ANDROID – GOOGLE MAPS	 374
Adding Google Map	374
Customizing Google Map	375
Integrating Google Maps	377
Download and configure. Google Play Services SDK.....	377
Obtaining the API key	378
Specify Android Manifest Settings	380



Adding Google Maps to your application	380
42. ANDROID – IMAGE EFFECTS	386
43. ANDROID – IMAGE SWITCHER.....	399
44. ANDROID – INTERNAL STORAGE	409
Writing file.....	409
Reading file.....	409
45. ANDROID – JETPLAYER	420
Using JetCreator	421
Verifying Results.....	424
46. ANDROID – JSON PARSER	426
JSON - Elements.....	426
JSON - Parsing.....	427
47. ANDROID – LINKEDIN INTEGRATION	440
Integrating LinkedIn SDK.....	440
Intent share	441
48. ANDROID – LOADING SPINNER.....	451
49. ANDROID – LOCALIZATION	458
Localizing Strings.....	458
50. ANDROID – LOGIN SCREEN	466
51. ANDROID – MEDIA PLAYER	478
52. ANDROID – MULTITOUCH.....	493
53. ANDROID – NAVIGATION	506
Providing Up Navigation	506



Handling device back button.....	506
54. ANDROID – NETWORK CONNECTION	518
Checking Network Connection	518
Performing Network Operations.....	519
55. ANDROID – NFC GUIDE	531
How It Works:.....	531
How it works with Android	531
Future Applications.....	533
56. ANDROID – PHP/MYSQL	534
PHP - MYSQL.....	534
Android - Connecting MYSQL.....	536
57. ANDROID – PROGRESS CIRCLE.....	553
58. ANDROID – PROGRESS BAR USING PROGRESS DIALOG	562
59. ANDROID – PUSH NOTIFICATION	572
60. ANDROID – RENDERSCRIPT	584
How RenderScript Works:.....	584
How to Begin	584
61. ANDROID – RSS READER	587
RSS Example	587
RSS Elements	587
Parsing RSS	588
62. ANDROID – SCREEN CAST	600
Screen Cast Steps.....	600
63. ANDROID – SDK MANAGER.....	604

Running Android SDK Manager.....	604
Recommended.....	605
Enabling Proxy in Android SDK Manager.....	605
Adding New Third Party Sites.....	606
64. ANDROID – SENSORS	608
Getting list of sensors supported	608
65. ANDROID – SESSION MANAGEMENT	615
Shared Preferences.....	615
Session Management through Shared Preferences.....	616
66. ANDROID – SIP PROTOCOL	631
Applications.....	631
Requirements	631
SIP Classes	631
Functions of SIP	632
Components of SIP.....	632
UAC.....	632
UAS.....	632
SipManager	632
67. ANDROID – SPELLING CHECKER.....	634
68. ANDROID – SQLITE DATABASE.....	644
Database - Package.....	644
Database - Creation	644
Database - Insertion.....	645
Database - Fetching	645
Database - Helper class.....	646

69. ANDROID – SUPPORT LIBRARY	673
Support Library Features	673
Downloading the Support Libraries.....	674
70. ANDROID – TESTING	676
Test Structure	676
Testing Tools in Android	676
JUnit	677
Monkey	678
71. ANDROID – TEXT TO SPEECH	687
72. ANDROID – TEXTURE VIEW	698
73. ANDROID – TWITTER INTEGRATION	708
Integrating Twitter SDK	708
Intent share	710
74. ANDROID – UI DESIGN	720
UI screen components	720
Understanding Screen Components.....	721
Units of Measurement	724
Screen Densities	724
Optimizing layouts	725
75. ANDROID – UI PATTERNS	726
UI Patterns components	726
Action Bar	726
Confirming and Acknowledging	727
Settings.....	728
Help	728



Selection.....	728
76. ANDROID – UI TESTING	730
uiautomatorviewer.....	730
uiautomator	734
77. ANDROID – WEBVIEW	742
78. ANDROID – WI-FI.....	751
79. ANDROID – WIDGETS	759
Widget - XML file	759
Widget - Layout file.....	759
Widget - Java file	759
Widget - Manifest file	760
80. ANDROID – XML PARSER.....	768
XML - Elements.....	768
XML - Parsing.....	769

1. ANDROID – Overview

What is Android?

Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need to develop only for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007, whereas the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 **Jelly Bean**. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

Features of Android

Android is a powerful operating system competing with Apple 4GS and support great features. Few of them are listed below:

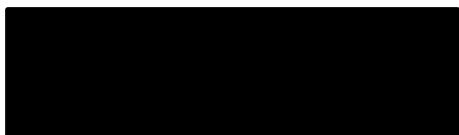
Feature	Description
Beautiful UI	Android OS basic screen provides a beautiful and intuitive user interface.
Connectivity	GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage	SQLite, a lightweight relational database, is used for data storage purposes.

Media support	H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
Messaging	SMS and MMS
Web browser	Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
Multi-touch	Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
Multi-tasking	User can jump from one task to another and same time various application can run simultaneously.
Resizable widgets	Widgets are resizable, so users can expand them to show more content or shrink them to save space
Multi-Language	Support single direction and bi-directional text.
GCM	Google Cloud Messaging (GCM) is a service that let developers send short message data to their users on Android devices, without needing a proprietary sync solution.
Wi-Fi Direct	A technology that let apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
Android Beam	A popular NFC-based technology that let users instantly share, just by touching two NFC-enabled phones together.

Android Applications

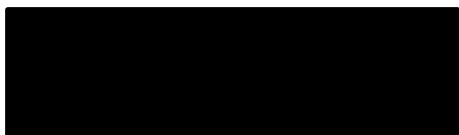
Android applications are usually developed in the Java language using the Android Software Development Kit.

Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play** or the **Amazon Appstore**.



Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and is growing fast. Every day more than 1 million new Android devices are activated worldwide.

This tutorial has been written with an aim to teach you how to develop and package Android application. We will start from environment setup for Android application programming and then drill down to look into various aspects of Android applications.



2. ANDROID – Environment Setup

You will be glad to know that you can start your Android application development on either of the following operating systems:

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or JDK6
- Android SDK
- Eclipse IDE for Java Developers (optional)
- Android Development Tools (ADT) Eclipse Plugin (optional)

Here last two components are optional and if you are working on Windows machine then these components make your life easy while doing Java based application development. So let us have a look at how to proceed to set the required environment.

Step 1 - Setup Java Development Kit (JDK)

You can download the latest version of Java JDK from Oracle's Java site: [Java SE Downloads](#). You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally, set PATH and JAVA_HOME environment variables to refer to the directory that contains **java** and **javac**, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and have installed the JDK in C:\jdk1.6.0_15, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.6.0_15\bin;%PATH%  
set JAVA_HOME=C:\jdk1.6.0_15
```

Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.



On Linux, if the SDK is installed in `/usr/local/jdk1.6.0_15` and you use the C shell, you would put the following code into your `.cshrc` file.

```
setenv PATH /usr/local/jdk1.6.0_15/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.6.0_15
```

Alternatively, if you use an Integrated Development Environment (IDE) Eclipse, then it will know automatically where you have installed your Java.

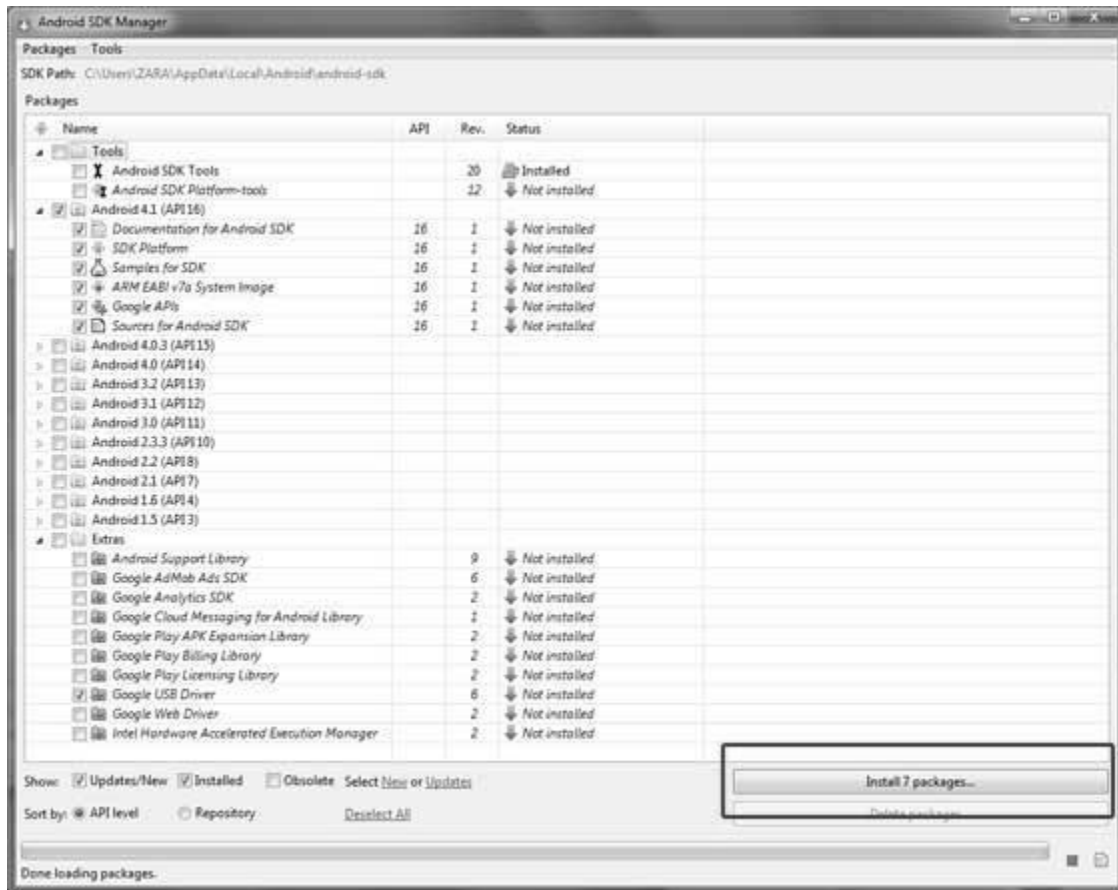
Step 2 - Setup Android SDK

You can download the latest version of Android SDK from Android's official website: <http://developer.android.com/sdk/index.html>. If you are installing SDK on Windows machine, then you will find *ainstaller_rXX-windows.exe*, so just download and run this exe which will launch *Android SDK Tool Setup* wizard to guide you throughout the installation, so just follow the instructions carefully. Finally, you will have *Android SDK Tools* installed on your machine.

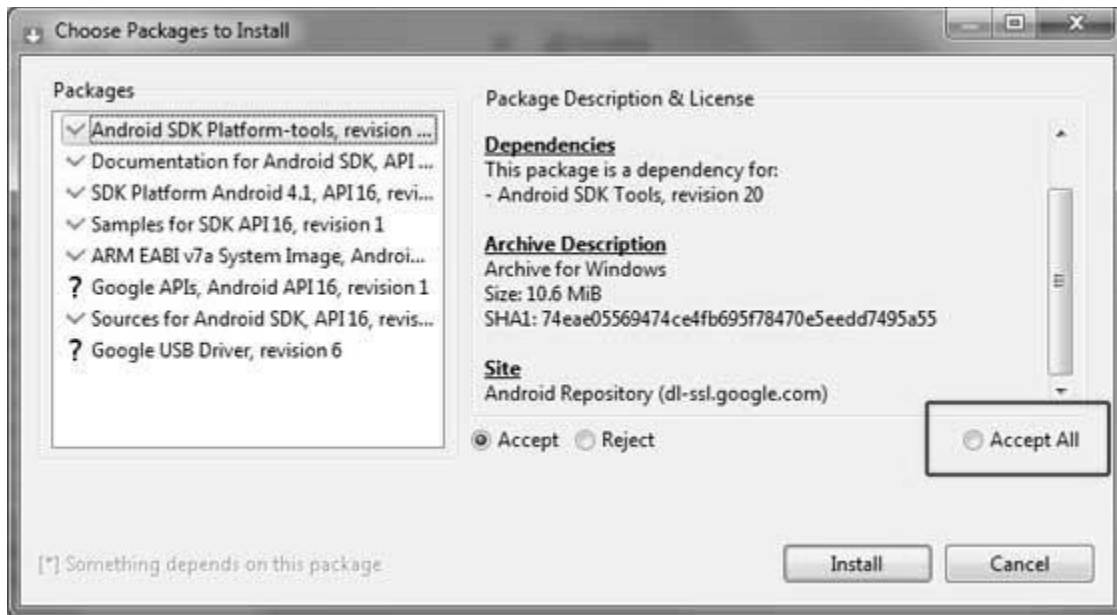
If you are installing SDK either on Mac OS or Linux, check the instructions provided along with the downloaded *android-sdk_rXX-macosx.zip* file for Mac OS and *android-sdk_rXX-linux.tgz* file for Linux. This tutorial will consider that you are going to setup your environment on Windows machine having Windows 7 operating system.

So let's launch *Android SDK Manager* using the option **All Programs > Android SDK Tools > SDK Manager**, this will give you following window:





Once you launched SDK manager, it is time to install other required packages. By default it will list down total 7 packages to be installed, but we will suggest to de-select *Documentation for Android SDK* and *Samples for SDK* packages to reduce installation time. Next click the **Install 7 Packages** button to proceed, which will display following dialogue box:



If you agree to install all the packages, select **Accept All** radio button and proceed by clicking **Install** button. Now let SDK manager do its work and you go, pick up a cup of coffee and wait until all the packages are installed. It may take some time depending on your internet connection. Once all the packages are installed, you can close SDK manager using top-right cross button.

Step 3 - Setup Eclipse IDE

All the examples in this tutorial have been written using Eclipse IDE. So we would suggest you should have latest version of Eclipse installed on your machine.

To install Eclipse IDE, download the latest Eclipse binaries from <http://www.eclipse.org/downloads/>. Once you have downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\eclipse on windows, or /usr/local/eclipse on Linux and finally set PATH variable appropriately.

Eclipse can be started by executing the following commands on windows machine, or you can simply double click on eclipse.exe

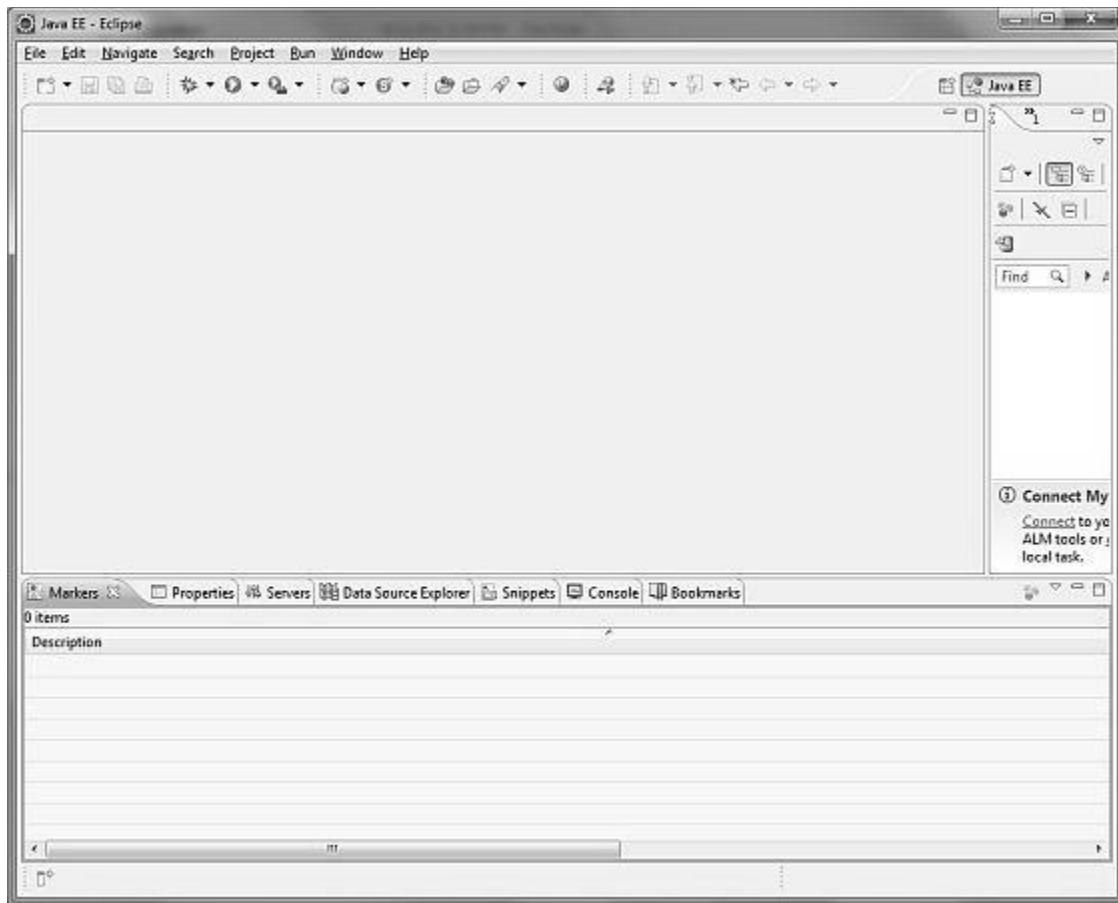
```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following command on Linux machine:

```
$/usr/local/eclipse/eclipse
```

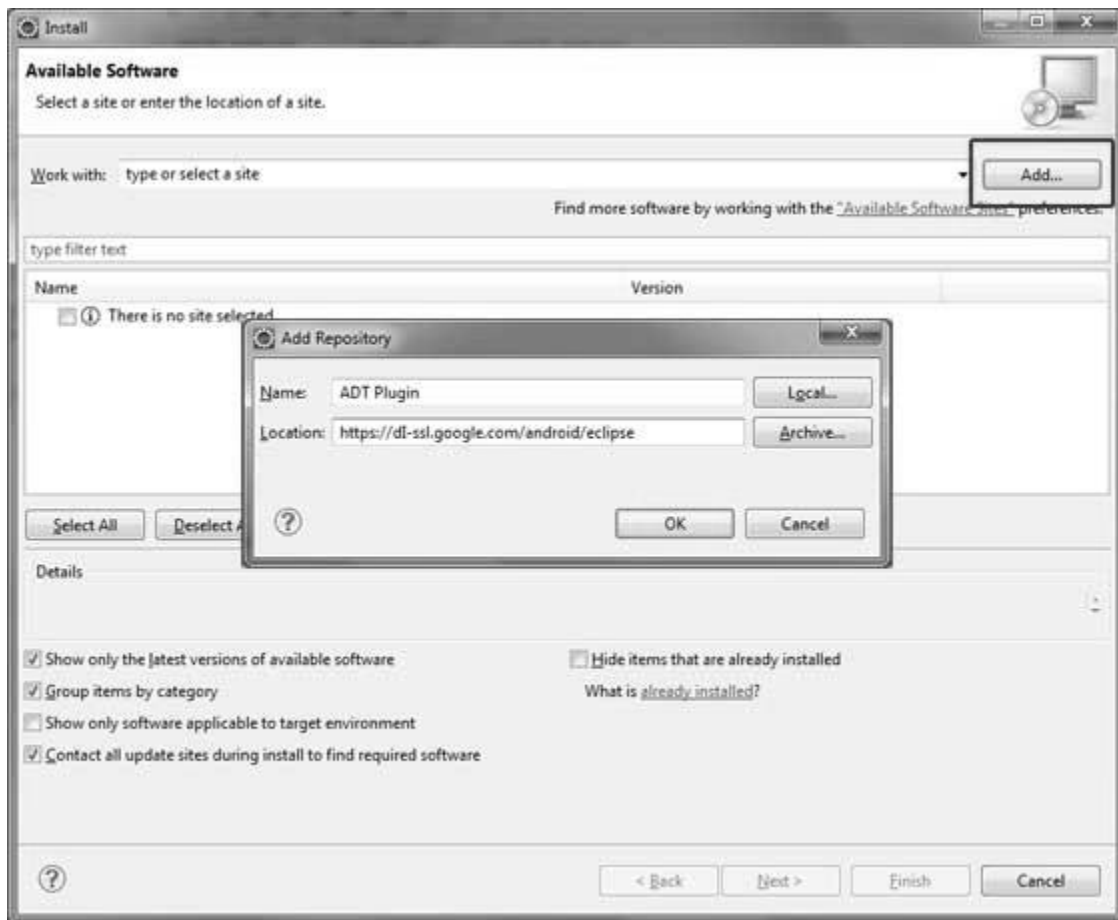
After a successful startup, if everything is fine then it should display the following result:



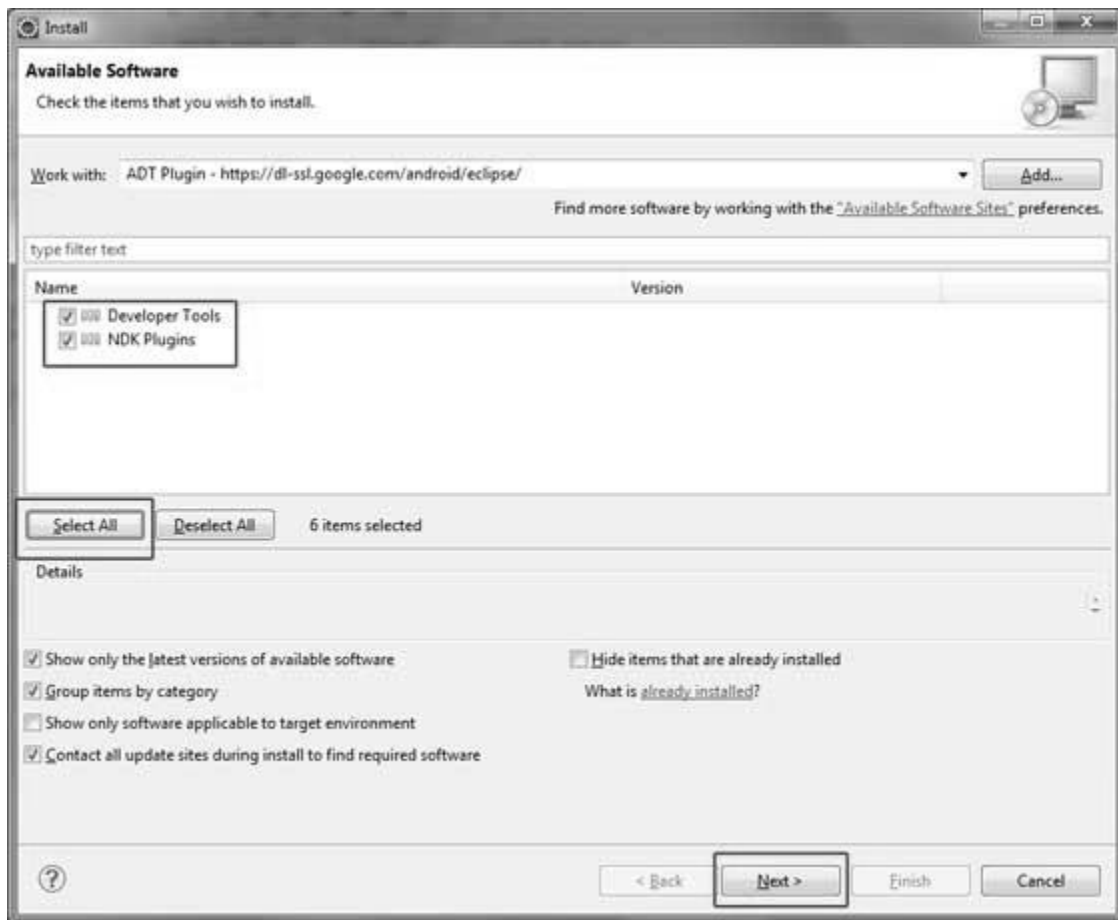


Step 4 - Setup Android Development Tools (ADT) Plugin

This step will help you in setting Android Development Tool plugin for Eclipse. Let's start with launching Eclipse and then, choose **Help > Software Updates > Install New Software**. This will display the following dialogue box.



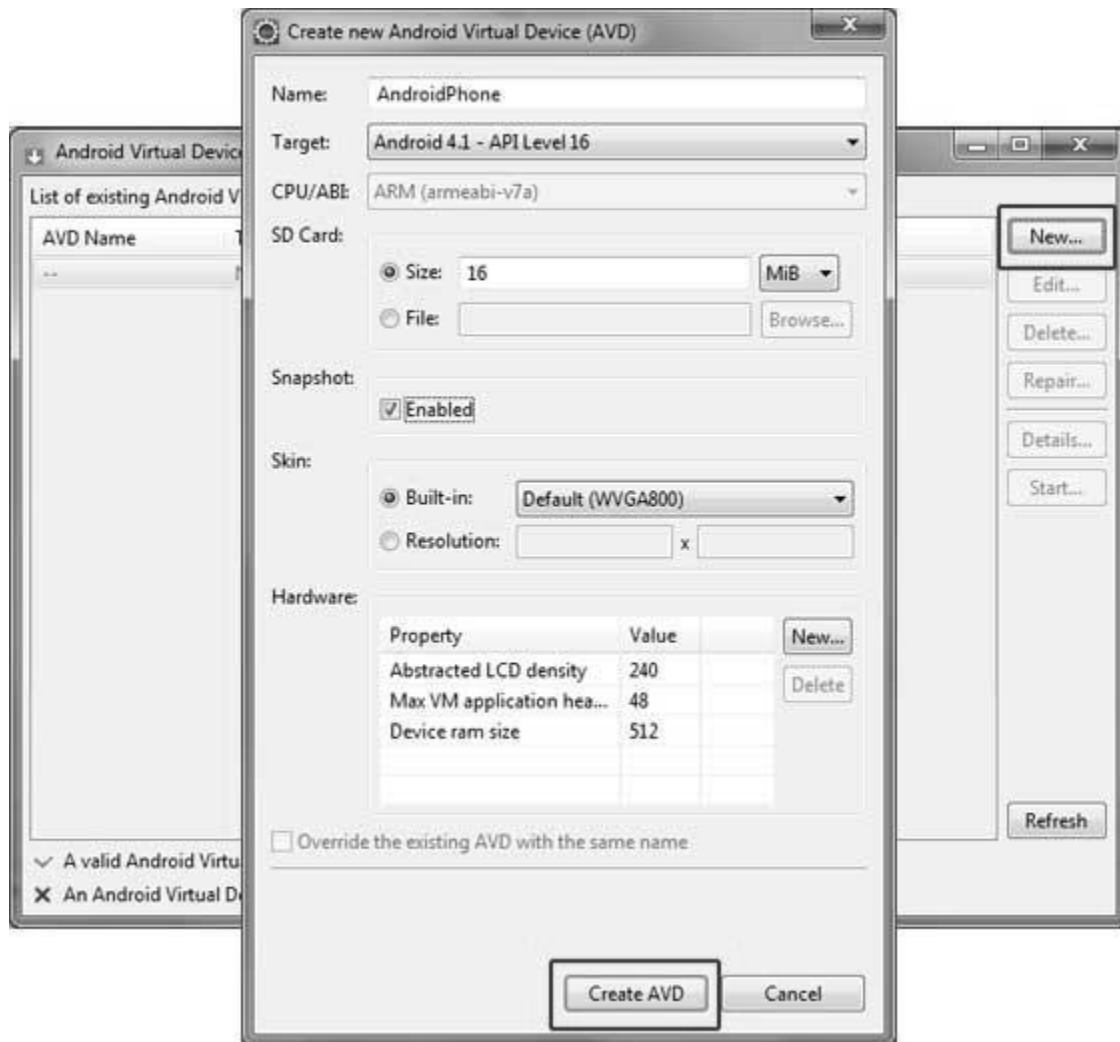
Now use **Add** button to add *ADT Plugin* as name and *https://dl-ssl.google.com/android/eclipse/* as the location. Then click OK to add this location. As soon as you will click OK button to add this location, Eclipse starts searching for the plug-in available in the given location and finally lists down the found plugins.



Now select all the listed plug-ins using **Select All** button and click **Next** button which will guide you ahead to install Android Development Tools and other required plugins.

Step 5 - Create Android Virtual Device

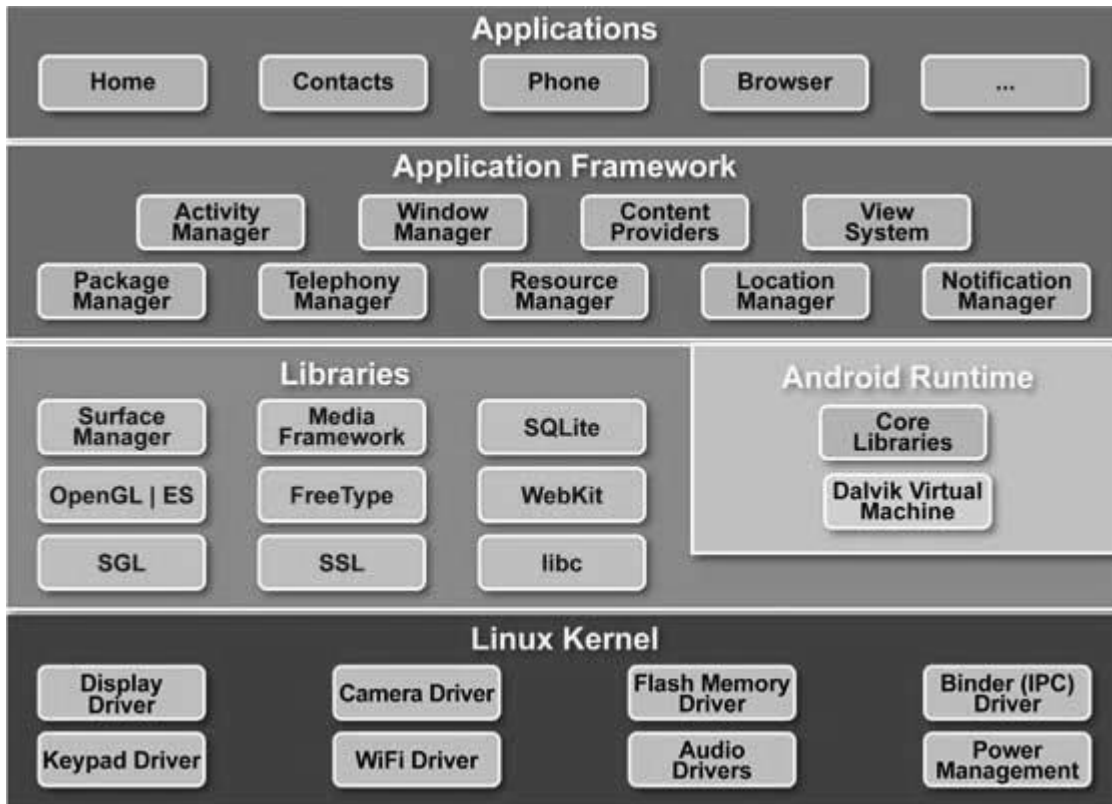
To test your Android applications you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager using Eclipse menu options **Window > AVD Manager** which will launch Android AVD Manager. Use **New** button to create a new Android Virtual Device and enter the following information, before clicking **Create AVD** button.



If your AVD is created successfully it means your environment is ready for Android application development. If you like, you can close this window using top-right cross button. Better you re-start your machine and once you are done with this last step, you are ready to proceed for your first Android example but before that we will see few more important concepts related to Android Application Development.

3. ANDROID – Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

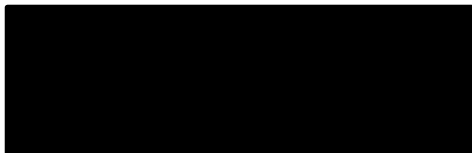


Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at, such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and



sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides 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.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games, etc.



4. ANDROID – Applications Component

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application:

Components	Description
Activities	They dictate the UI and handle the user interaction to the smartphone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and one for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows:

```
public class MainActivity extends Activity
{
}
}
```

Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows:

```
public class MyService extends Service
{

}
```

Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcasted as an **Intent** object.

```
public class MyReceiver extends BroadcastReceiver
{

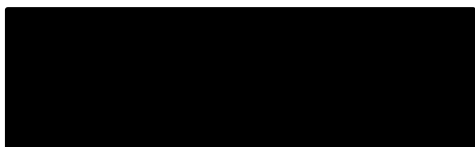
}
```

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends ContentProvider
{
```



```
}
```

We will go through these tags in detail while covering application components in individual chapters.

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

Components	Description
Fragments	Represent a behavior or a portion of user interface in an Activity.
Views	UI elements that are drawn onscreen including buttons, lists forms etc.
Layouts	View hierarchies that control screen format and appearance of the views.
Intents	Messages wiring components together.
Resources	External elements, such as strings, constants and drawable pictures.
Manifest	Configuration file for the application.

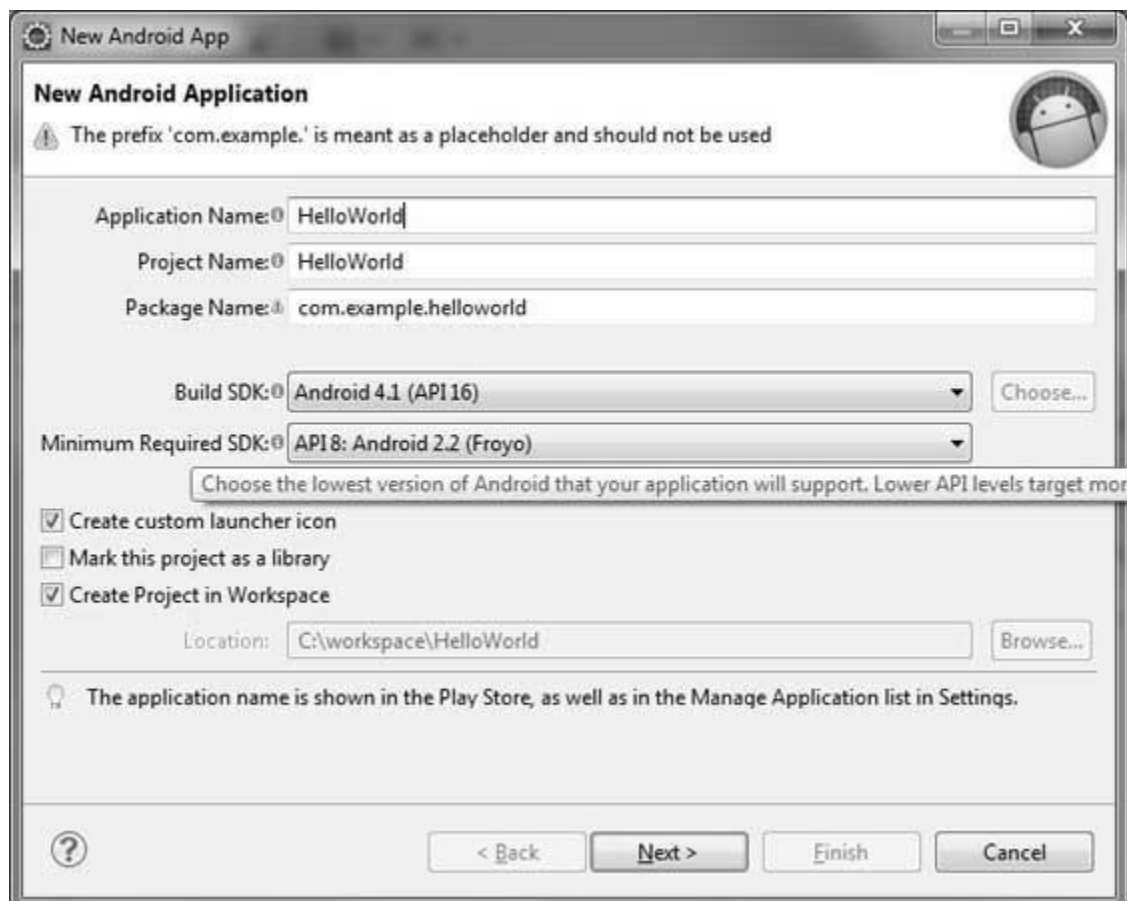
5. ANDROID – Hello World Example

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](#) tutorial. We also assume, that you have a little bit working knowledge with Eclipse IDE.

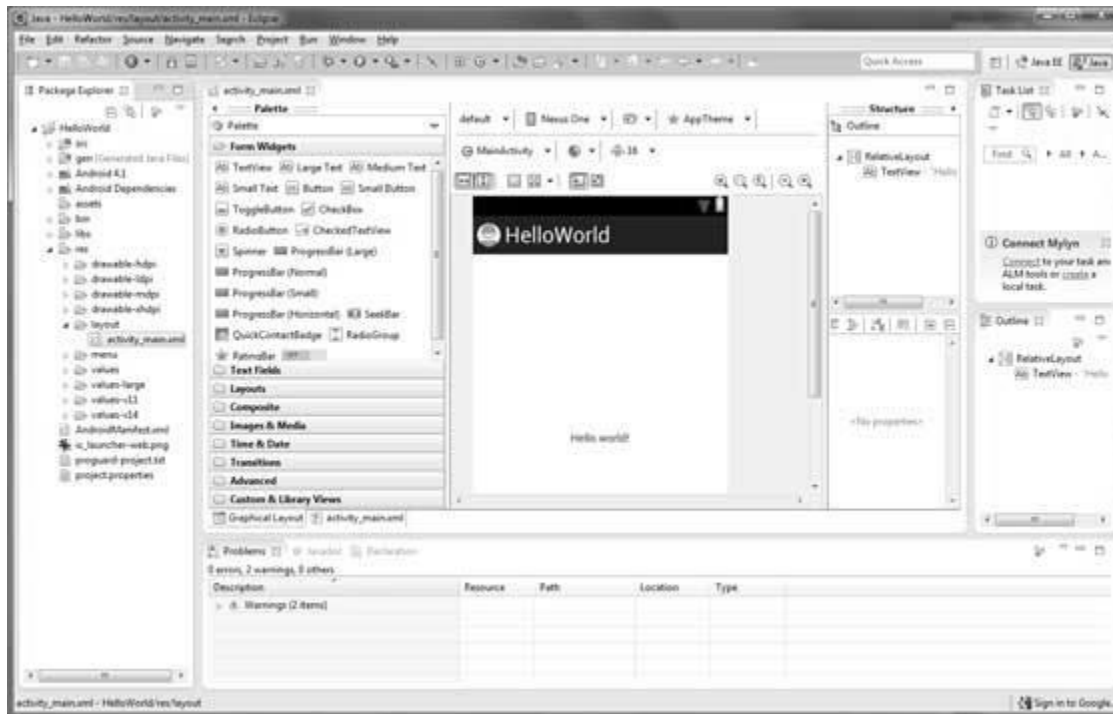
So let us proceed to write a simple Android Application which will print "Hello World!".

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:

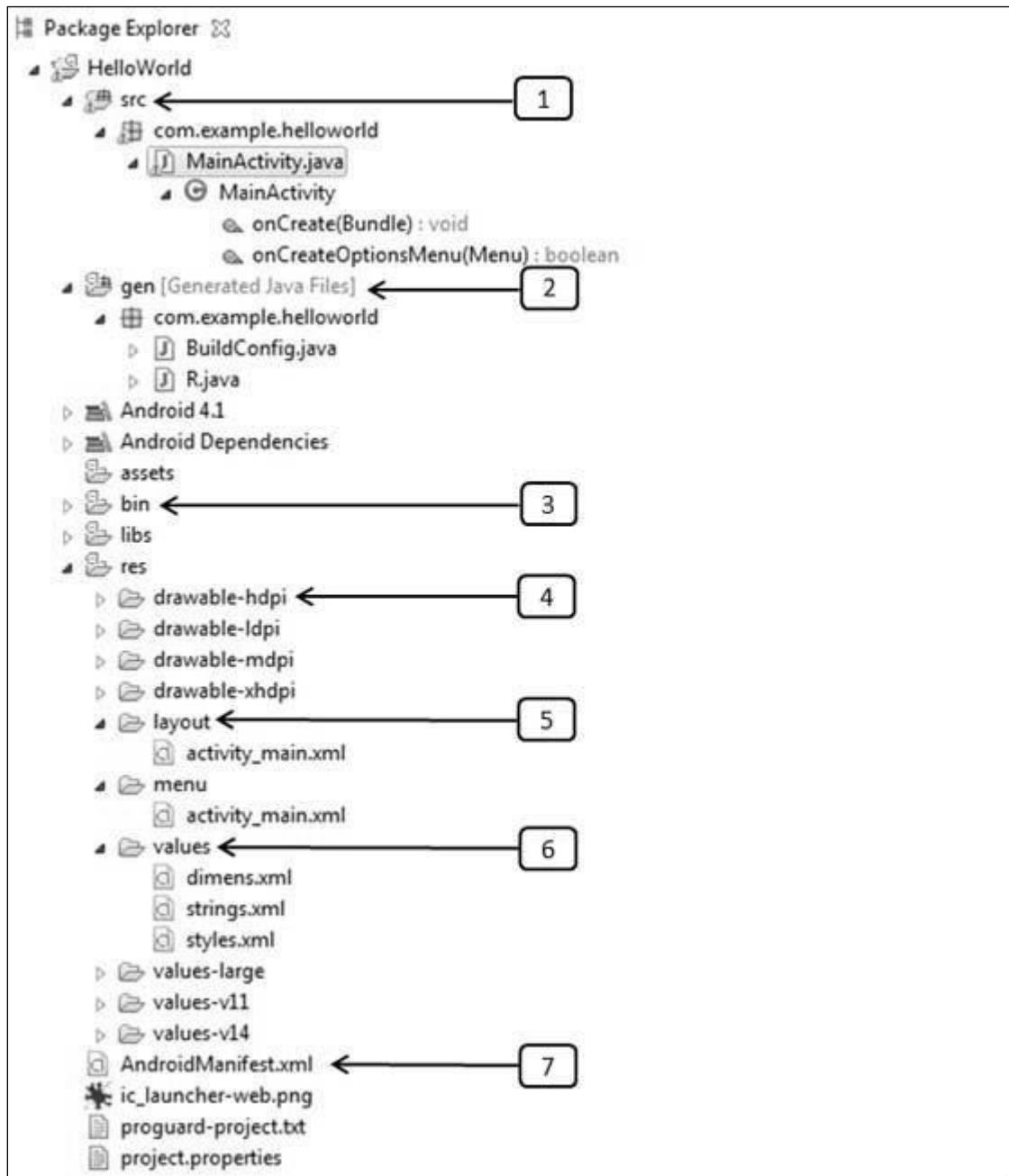


Next, follow the instructions provided and keep all other entries as default till the final step. Once your project is created successfully, you will have the following project screen:



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 <i>MainActivity.java</i> 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 fired 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>

```

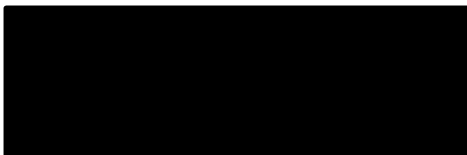
Here `<application>...</application>` tags enclosed the components related to the application. Attribute *android:icon* will point to the application icon available under *res/drawable-hdpi*. The application uses the image named *ic_launcher.png* located in the drawable folders.

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* file, 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 string 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" >

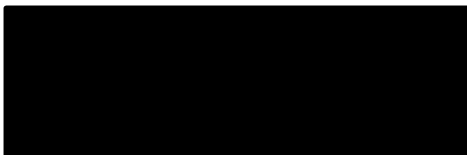
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:padding="@dimen/padding_medium"
        android:text="@string/hello_world"
        tools:context=".MainActivity"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

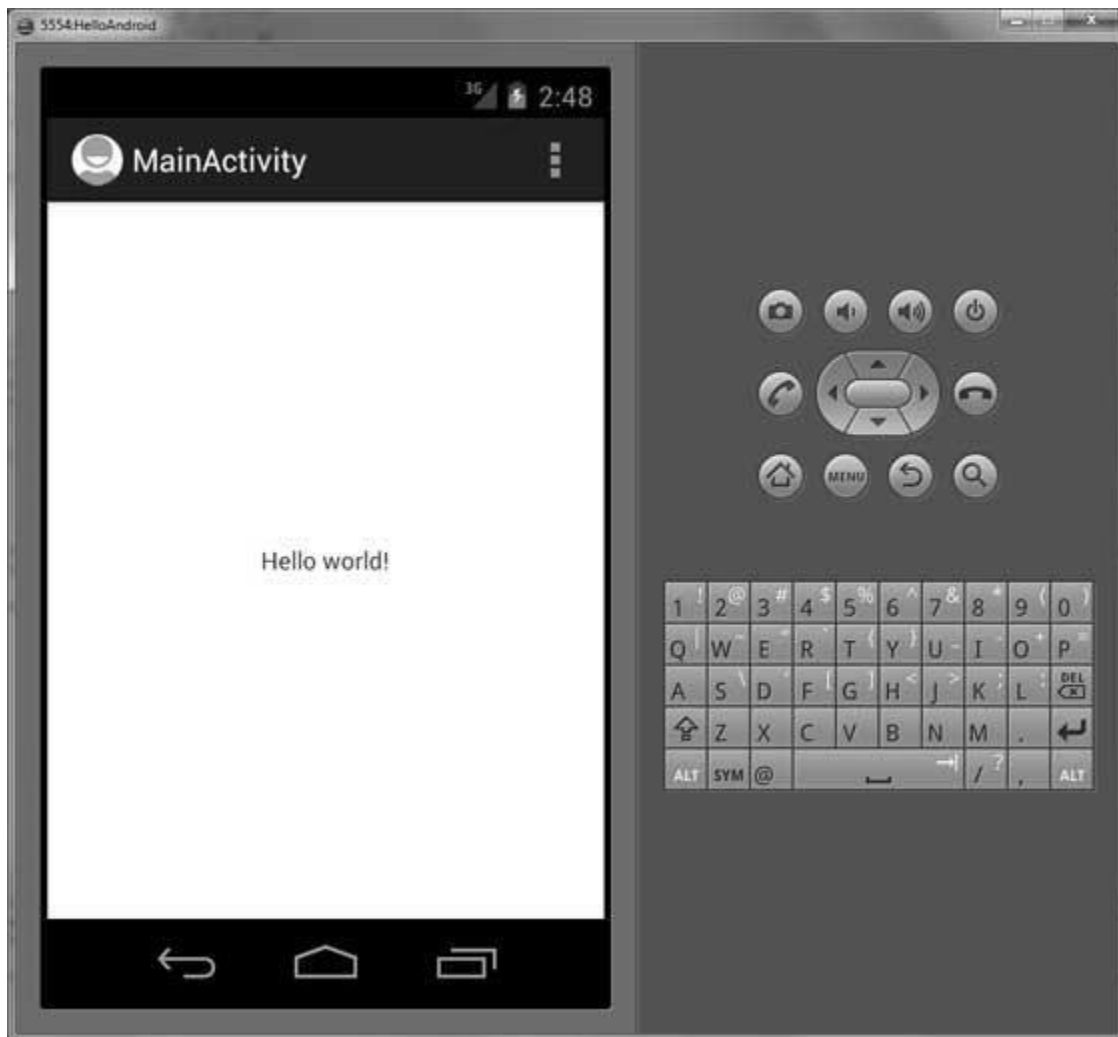
</RelativeLayout>
```

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 has various attributes 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* file, which is "Hello World!".

Running the Application

Let's try to run our **Hello World!** application we just created. We 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  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:





Congratulations! You have developed your first Android Application and now just keep following rest of the tutorial step by step to become a great Android Developer. All the very best!

6. ANDROID – Organizing & Accessing the resources

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other **resources** like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications.

Organize Resources

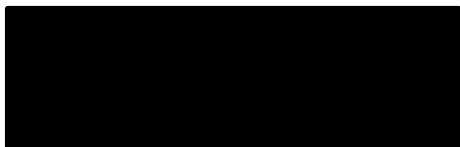
You should place each type of resource in a specific subdirectory of your project's **res/** directory. For example, here's the file hierarchy for a simple project:

```
MyProject/
  src/
    MainActivity.java
  res/
    drawable/
      icon.png
    layout/
      activity_main.xml
      info.xml
    values/
      strings.xml
```

The **res/** directory contains all the resources in various sub-directories. Here we have an image resource, two layout resources, and a string resource file. Following table gives a detail about the resource directories supported inside project **res/** directory.

Directory	Resource Type
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.
drawable/	Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawables. They are saved in res/drawable/ and accessed from the R.drawable class.
layout/	XML files that define a user interface layout. They are saved in res/layout/ and accessed from the R.layout class.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the R.menu class.
raw/	Arbitrary files to save in their raw form. You need to call <i>Resources.openRawResource()</i> with the resource ID, which is <i>R.raw.filename</i> to open such raw files.
values/	<p>XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory:</p> <p>arrays.xml for resource arrays, and accessed from the R.array class.</p> <p>integers.xml for resource integers, and accessed from the R.integer class.</p> <p>bools.xml for resource boolean, and accessed from the R.bool class.</p> <p>colors.xml for color values, and accessed from the R.color class.</p> <p>dimens.xml for dimension values, and accessed from the R.dimen class.</p> <p>strings.xml for string values, and accessed from the R.string class.</p> <p>styles.xml for styles, and accessed from the R.style class.</p>
xml/	Arbitrary XML files that can be read at runtime by calling <i>Resources.getXML()</i> . You can save various configuration files here which will be used at run time.



Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e. images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow these steps:

- Create a new directory in `res/` named in the form **<resources_name>-<config_qualifier>**. Here **resources_name** will be any of the resources mentioned in the above table, like layout, drawable etc. The **qualifier** will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.
- Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screen.

```
MyProject/  
  src/  
    MainActivity.java  
  res/  
    drawable/  
      icon.png  
      background.png  
    drawable-hdpi/  
      icon.png  
      background.png  
    layout/  
      activity_main.xml  
      info.xml  
    values/  
      strings.xml
```

Below is another example which specifies layout for a default language and alternative layout for Arabic language (layout-ar/).

```
MyProject/
  src/
    MainActivity.java
  res/
    drawable/
      icon.png
      background.png
    drawable-hdpi/
      icon.png
      background.png
    layout/
      activity_main.xml
      info.xml
    layout-ar/
      main.xml
    values/
      strings.xml
```

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:

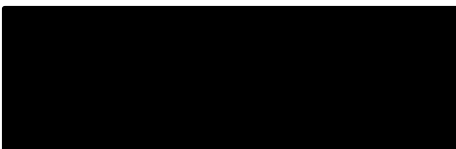
Accessing Resources in Code

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.

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);
```

Here first line of the code uses the *R.id.myimageview* to get *ImageView* defined with *idmyimageview* in a *Layout* file. Second line of code uses the *R.drawable.myimage* to get an image with name **myimage** available in *drawable* sub-directory under **/res**.

Example:

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>
```

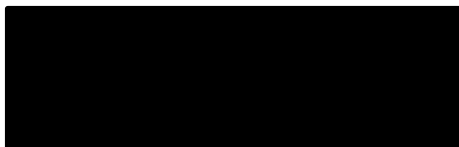
Now you can set the text on a *TextView* object with ID *msg* using a resource ID as follows:

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

Example:

Consider a layout *res/layout/activity_main.xml* with the following definition:

```
<?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="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
```



```
</LinearLayout>
```

This application code will load this layout for an Activity, in the onCreate() method as follows:

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

Accessing Resources in XML

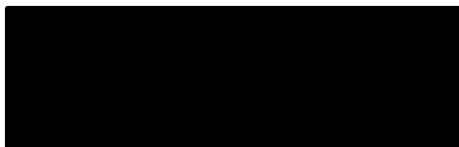
Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource:

```
<?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" />
```

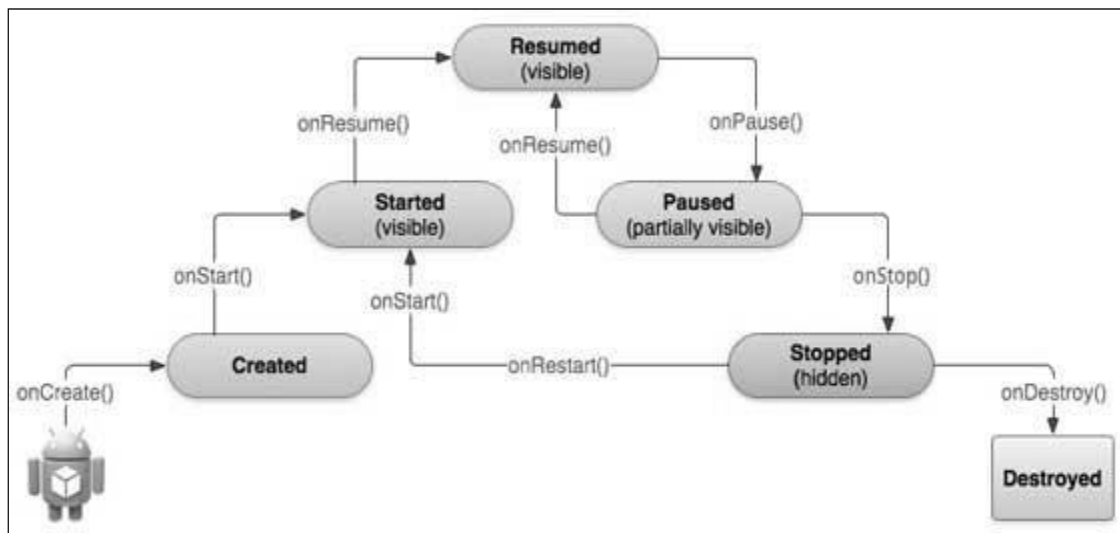
Now if you go through the previous chapter once again where we have explained **Hello World!** example, surely you will have better understanding on all the concepts explained in this chapter. So we highly recommend to check previous chapter for working example and check how we have used various resources at very basic level.



7. ANDROID – Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program within an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity lifecycle diagram: (*image courtesy: android.com*)



The Activity class defines the following callbacks i.e. events. You don't need to implement all the callback methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback	Description
<i>onCreate()</i>	This is the first callback and called when the activity is first created.
<i>onStart()</i>	This callback is called when the activity becomes visible to the user.

onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

Example:

This example will take you through simple steps to show Android application activity life cycle. Follow the below mentioned steps to modify the Android application we created in *Hello World Example* chapter:

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>HelloWorld</i> under a package <i>com.example.helloworld</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify main activity file <i>MainActivity.java</i> as explained below. Keep rest of the files unchanged.
3	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.helloworld/MainActivity.java**. This file includes each of the fundamental lifecycle methods. The **Log.d()** method has been used to generate log messages:

```
package com.example.helloworld;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
    String msg = "Android : ";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "The onCreate() event");
    }

    /** Called when the activity is about to become visible. */
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(msg, "The onStart() event");
    }

    /** Called when the activity has become visible. */
    @Override
    protected void onResume() {
        super.onResume();
        Log.d(msg, "The onResume() event");
    }

    /** Called when another activity is taking focus. */
    @Override
    protected void onPause() {
```



```

        super.onPause();
        Log.d(msg, "The onPause() event");
    }

    /** Called when the activity is no longer visible. */
    @Override
    protected void onStop() {
        super.onStop();
        Log.d(msg, "The onStop() event");
    }

    /** Called just before the activity is destroyed. */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(msg, "The onDestroy() event");
    }
}

```

An activity class loads all the UI component using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity_main.xml* file:

```
setContentView(R.layout.activity_main);
```

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an `<intent-filter>` that includes the MAIN action and LAUNCHER category as follows:

```

<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>

```

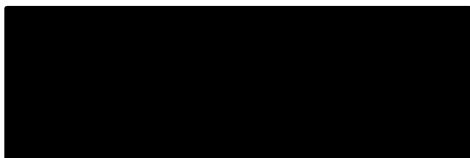
If either the MAIN action or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

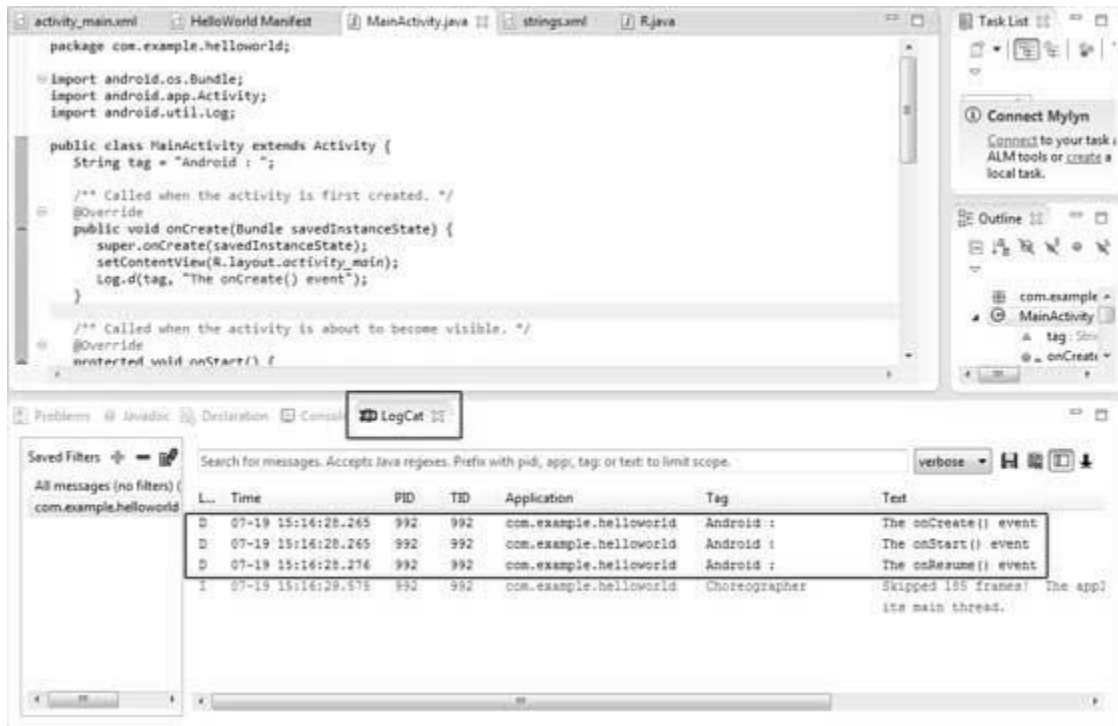
Let's try to run our modified **Hello World!** application we just modified. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts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


```

07-19 15:00:43.405: D/Android :(866): The onCreate() event
07-19 15:00:43.405: D/Android :(866): The onStart() event
07-19 15:00:43.415: D/Android :(866): The onResume() event


```






Let us try to click Red button  on the Android emulator and it will generate following events messages in **LogCat** window in Eclipse IDE:

```
07-19 15:01:10.995: D/Android :(866): The onPause() event
07-19 15:01:12.705: D/Android :(866): The onStop() event
```

Let us again try to click Menu button  on the Android emulator and it will generate following events messages in **LogCat** window in Eclipse IDE:

```
07-19 15:01:13.995: D/Android :(866): The onStart() event
07-19 15:01:14.705: D/Android :(866): The onResume() event
```

Next, let us again try to click Back button  on the Android emulator and it will generate following events messages in **LogCat** window in Eclipse IDE and this completes the Activity Life Cycle for an Android Application.

```
07-19 15:33:15.687: D/Android :(992): The onPause() event
07-19 15:33:15.525: D/Android :(992): The onStop() event
07-19 15:33:15.525: D/Android :(992): The onDestroy() event
```

End of ebook preview
If you liked what you saw...
Buy it from our store @ **<https://store.tutorialspoint.com>**

