



ANDROID

application development

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

 <https://www.facebook.com/tutorialspointindia>

 <https://twitter.com/tutorialspoint>

About the Tutorial

Android is an open-source, 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.

This tutorial will teach you the basic Android programming and will also take you through some advance concepts related to Android application development.

Audience

This tutorial has been prepared for beginners to help them understand basic Android programming. After completing this tutorial, you will find yourself at a moderate level of expertise in Android programming from where you can take yourself to next levels.

Prerequisites

Android programming is based on Java programming language. If you have a basic understanding of Java programming, then it will be fun to learn Android application development.

Copyright & Disclaimer

© Copyright 2014 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents.....	ii
1. OVERVIEW	1
What is Android?	1
Features of Android	1
Android Applications	2
2. ENVIRONMENT SETUP	4
Step 1 - Setup Java Development Kit (JDK).....	4
Step 2 - Setup Android SDK.....	5
Step 3 - Setup Eclipse IDE	6
Step 4 - Setup Android Development Tools (ADT) Plugin	7
Step 5 - Create Android Virtual Device	9
3. ARCHITECTURE.....	11
Linux kernel	11
Libraries	11
Android Runtime	12
Application Framework.....	12
Applications.....	12
4. APPLICATIONS COMPONENT.....	13
Activities	13
Services.....	14

Broadcast Receivers	14
Content Providers	14
Additional Components	15
5. HELLO WORLD EXAMPLE	16
Create Android Application	16
Anatomy of Android Application	17
The Main Activity File	19
The Manifest File	20
The Strings File	21
The R File	22
The Layout File	23
Running the Application	24
6. ORGANIZING & ACCESSING THE RESOURCES	26
Organize Resources	26
Alternative Resources	28
Accessing Resources	29
Accessing Resources in Code	29
Accessing Resources in XML	31
7. ACTIVITIES	32
8. SERVICES	38
9. BROADCAST RECEIVERS	49
Creating the Broadcast Receiver	49
Registering Broadcast Receiver	49
Broadcasting Custom Intents	51
10. CONTENT PROVIDERS	58

Content URIs	58
Create Content Provider	59
11. FRAGMENTS	74
Fragment Life Cycle	75
How to use Fragments?	76
12. INTENTS & FILTERS	85
Intent Objects	86
Action	86
Data	86
Category	87
Extras	87
Flags	87
Component Name	87
Types of Intents	87
Explicit Intents	88
Implicit Intents	88
Intent Filters	93
13. UI LAYOUTS	104
Android Layout Types	105
Layout Attributes	106
View Identification	108
14. UI CONTROLS	110
Android UI Controls	110
Create UI Controls	112
15. EVENT HANDLING	113

Event Listeners & Event Handlers	113
Event Listeners Registration:.....	114
Event Handling Examples	114
Event Listeners Registration Using an Anonymous Inner Class	114
Registration Using the Activity Implements Listener Interface.....	119
Registration Using Layout file activity_main.xml	122
Exercise:.....	126
17. STYLES & THEMES	127
Defining Styles	127
Using Styles.....	128
Style Inheritance	129
Android Themes.....	130
Default Styles & Themes	130
18. CUSTOM COMPONENTS	132
Creating a Simple Custom Component	132
Instantiate using code inside activity class	133
Instantiate using Layout XML file	134
Custom Component with Custom Attributes	135
Step 1.....	135
Step 2.....	135
Step 3.....	136
19. DRAG & DROP	138
The Drag/Drop Process	138
The DragEvent Class.....	139
Constants	139
Methods.....	140
Listening for Drag Event	140
Starting a Drag Event	141

20. NOTIFICATIONS	148
Create and Send Notifications	149
Step 1 - Create Notification Builder	149
Step 2 - Setting Notification Properties	149
Step 3 - Attach Actions.....	149
Step 4 - Issue the notification	150
The NotificationCompat.Builder Class	150
Big View Notification	162
21. LOCATION-BASED SERVICES	165
The Location Object	165
Get the Current Location	167
Get the Updated Location	168
Location Quality of Service	168
Displaying a Location Address	169
Install the Google Play Services SDK	170
Create Android Application	170
22. SENDING EMAIL	182
Intent Object - Action to send Email	182
Intent Object - Data/Type to send Email	182
Intent Object - Extra to send Email	182
23. SENDING SMS	190
Using SmsManager to send SMS	190
Using Built-in Intent to send SMS	197
Intent Object - Action to send SMS	197
Intent Object - Data/Type to send SMS	198
Intent Object - Extra to send SMS	198
24. PHONE CALLS	205
Intent Object - Action to make Phone Call	205

Intent Object - Data/Type to make Phone Call	205
25. PUBLISHING ANDROID APPLICATION	212
Export Android Application	213
Google Play Registration	217
26. ALERT DIALOG TUTORIAL	219
27. ANIMATIONS	234
Tween Animation	234
Zoom in animation	235
28. AUDIO CAPTURE	250
29. AUDIO MANAGER	263
30. AUTOCOMPLETE	276
31. BEST PRACTICES	286
Best Practices - User input	286
AsyncTask Vs Services.	286
Best Practices - Performance	287
Best Practices - Security and privacy	287
32. BLUETOOTH	299
33. CAMERA	314
Using existing android camera application in our application	314
Directly using Camera API Provided by Android in our Application	323
34. CLIPBOARD	335
Copying data	335
Pasting data	336
35. CUSTOM FONTS	346

36. DATA BACKUP	352
Test your BackupAgent	356
Install your application on a suitable Android system image.	356
Ensure data backup is enabled	356
Performing backup.....	357
Uninstall and reinstall your application	357
37. DEVELOPER TOOLS	358
SDK tools	358
Android.....	359
DDMS.....	359
Running DDMS	359
How it works	359
Using DDMS	360
Making SMS.....	360
Making Call.....	361
Capturing ScreenShot	363
Sqlite3.....	364
Use Sqlite3 from a remote shell.	364
Using Sqlite3 directly.....	365
Platform tools	365
38. EMULATOR.....	366
Creating AVD	366
Creating Snapshots	366
Changing Orientation.....	366
Emulator Commands.	368
Emulator - Sending SMS.....	369
Sending SMS through Telnet.	369
Emulator - Making Call.....	370
Emulator - Transferring files.....	371
39. FACEBOOK INTEGRATION.....	372
Integrating Facebook SDK	372
Generating application signature.....	372
Registering your application	373

Downloading SDK and integrating it	373
Creating facebook login application.....	373
Intent share	374
40. GESTURES	384
Handling Pinch Gesture.....	384
41. GOOGLE MAPS	395
Adding Google Map	395
Google Map - Activity file.....	395
Google Map - Layout file.....	395
Google Map - AndroidManifest file	395
Customizing Google Map	396
Adding Marker	396
Changing Map Type.....	396
Enable/Disable zoom	397
Integrating Google Maps.....	398
Download and configure. Google Play Services SDK.....	399
Install Google services SDK	399
Import SDK to eclipse.....	399
Configure your project with SDK.....	399
Obtaining the API key	400
Getting Certificate from KeyTool	400
Getting key from Google Console	401
Specify Android Manifest Settings	401
Adding Google Maps to your application.	402
42. IMAGE EFFECTS	408
43. IMAGE SWITCHER	421
44. INTERNAL STORAGE	431
Writing file.....	431
Reading file	431
45. JETPLAYER	442
Using JetCreator.....	444

Installing python.....	444
Installing WxPython	444
Running JetCreator.....	444
Creating JetContent	445
Verifying Results	446
46. JSON PARSER	448
JSON - Elements	449
JSON - Parsing	449
47. LINKEDIN INTEGRATION	463
Integrating LinkedIn SDK	463
Registering your application	463
Downloading SDK and integrating it	464
Posting updates on LinkedIn application	464
Intent share	464
48. LOADING SPINNER	474
49. LOCALIZATION	481
Localizing Strings.....	481
Italy, res/values-it/strings.xml	481
Spanish, res/values-it/strings.xml	482
French, res/values-it/strings.xml	482
50. LOGIN SCREEN	489
51. MEDIA PLAYER	501
52. MULTITOUCH	517
53. NAVIGATION	530
Providing Up Navigation	530
Handling device back button.....	530
54. NETWORK CONNECTION	542
Checking Network Connection	542

Performing Network Operations	543
55. NFC GUIDE	556
How It Works:	556
Three Modes of Operation.....	556
How it works with Android:	556
Future Applications	558
56. PHP/MYSQL	559
PHP - MYSQL	559
Creating Database.....	559
Creating Tables.....	559
Inserting Values in tables	560
PHP - GET and POST methods	560
Android - Connecting MYSQL	561
Connecting Via Get Method.....	561
Connecting Via Post Method	561
PHP - MYSQL part.....	562
Android Part.....	563
57. PROGRESS CIRCLE	579
58. PROGRESS BAR USING PROGRESS DIALOG	588
59. PUSH NOTIFICATION	598
60. RENDERSRIPT	611
How RenderScript Works:	611
How to Begin:	611
A RenderScript Kernel	611
RenderScript APIs.....	612
How to use RenderScript Support Library.....	613
61. RSS READER	615
RSS Example	615
RSS Elements	615
Parsing RSS	616

62. SCREEN CAST	629
Screen Cast Steps	629
Step 1	629
Step	629
Step 3	629
Step	629
Step 5	630
63. SDK MANAGER	633
Running Android SDK Manager:	633
Recommended	634
Enabling Proxy in Android SDK Manager	634
Adding New Third Party Sites	635
64. SENSORS	637
Getting list of sensors supported.	638
65. SESSION MANAGEMENT	645
Shared Preferences	645
Session Management through Shared Preferences	646
66. SIP PROTOCOL	662
Applications	662
Requirements	662
SIP Classes	662
Functions of SIP	663
Components of SIP	663
UAC	663
UAS	663
SipManager	663
67. SPELLING CHECKER	665

68. SQLITE DATABASE	675
Database - Package	675
Database - Creation	675
Database - Insertion	676
Database - Fetching	676
Database - Helper class	677
69. SUPPORT LIBRARY	706
Support Library Features	706
Downloading the Support Libraries	707
Choosing Support Libraries	708
Changes in Android.Manifest	708
API Version	708
70. TESTING	709
Test Structure	709
Testing Tools in Android	709
JUnit	710
Monkey	711
Monkey features	711
Monkey Usage	711
71. TEXT TO SPEECH	720
72. TEXTURE VIEW	731
73. TWITTER INTEGRATION	741
Integrating Twitter SDK	741
Registering your application	741
Downloading SDK and integrating it	743
Posting tweets on twitter application	743
Intent share	743
74. UI DESIGN	753
UI screen components	753

Understanding Screen Components	754
View and ViewGroups	754
Types of layout	754
Linear Layout.....	754
AbsoluteLayout	755
TableLayout.....	755
RelativeLayout.....	756
FrameLayout	756
Units of Measurement	757
Screen Densities	758
Optimizing layouts	758
75. UI PATTERNS	759
UI Patterns components	759
Action Bar	759
Action Bar Components	759
Confirming and Acknowledging	760
Confirming.....	760
Acknowledging	761
Settings	761
Placement of Settings	761
Help	761
Placement of Help.....	762
Selection	762
Using Contextual Action Bar (CAB).....	762
76. UI TESTING	763
uiautomatorviewer	763
uiautomator	767
77. WEBVIEW	775
78. WI-FI	785
79. WIDGETS	793
Widget - XML file	793
Widget - Layout file	793

Widget - Java file.....	793
Widget - Manifest file	794
80. XML PARSER	803
XML - Elements	803
XML - Parsing	804

1. 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. ENVIORNMENT 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
```

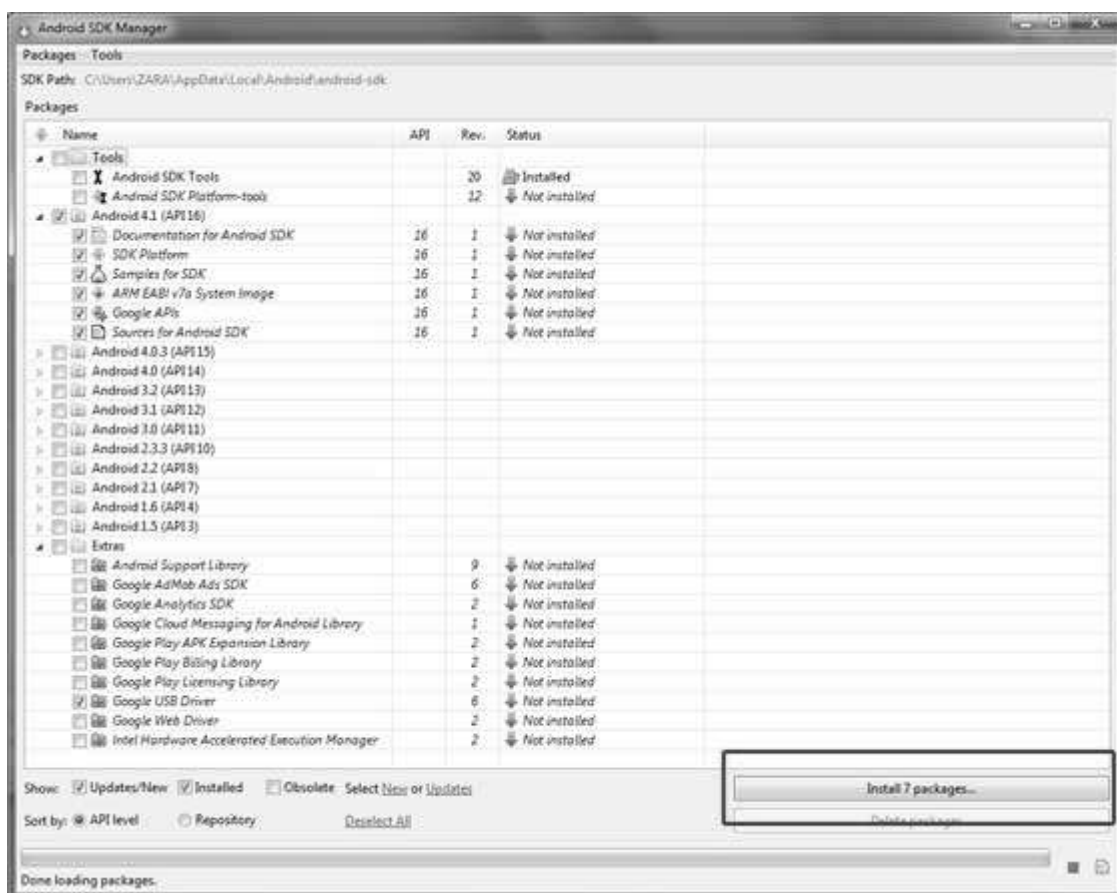
Alternatively, if you use an Integrated Development Environment (IDE) Eclipse, then it will know automatically where you have installed your Java.

Step 2 - Setup Android SDK

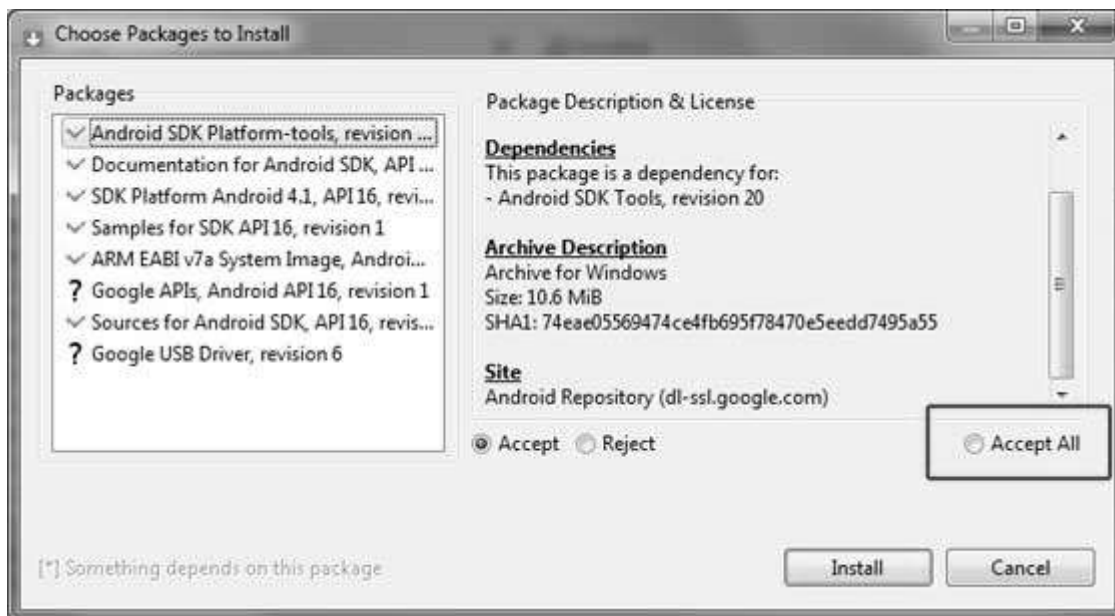
You can download the latest version of Android SDK from Android’s official website: <http://developer.android.com/sdk/index.html>. If you are installing SDK on Windows machine, then you will find *ainstaller_rXX-windows.exe*, so just download and run this exe which will launch *Android SDK Tool Setup* 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.

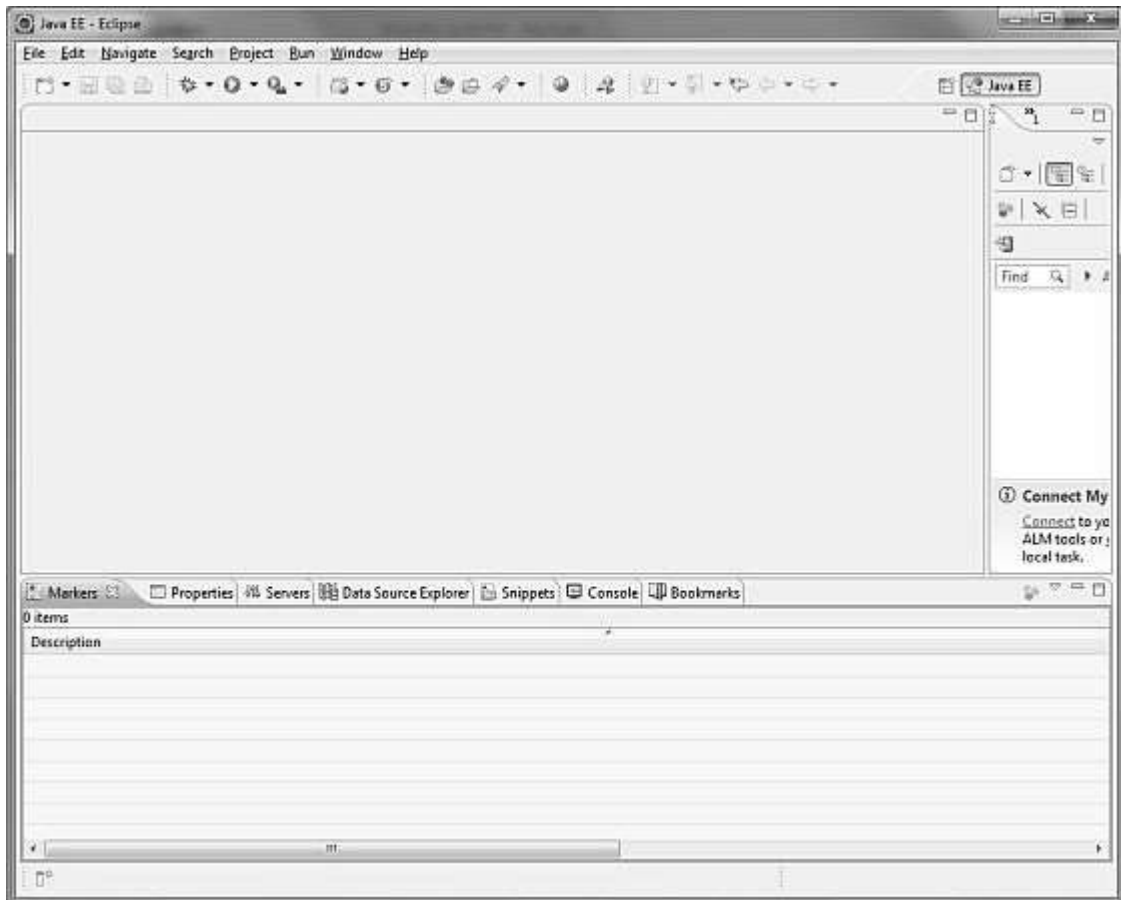
Eclipse can be started by executing the following commands on windows machine, or you can simply double click on eclipse.exe

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

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

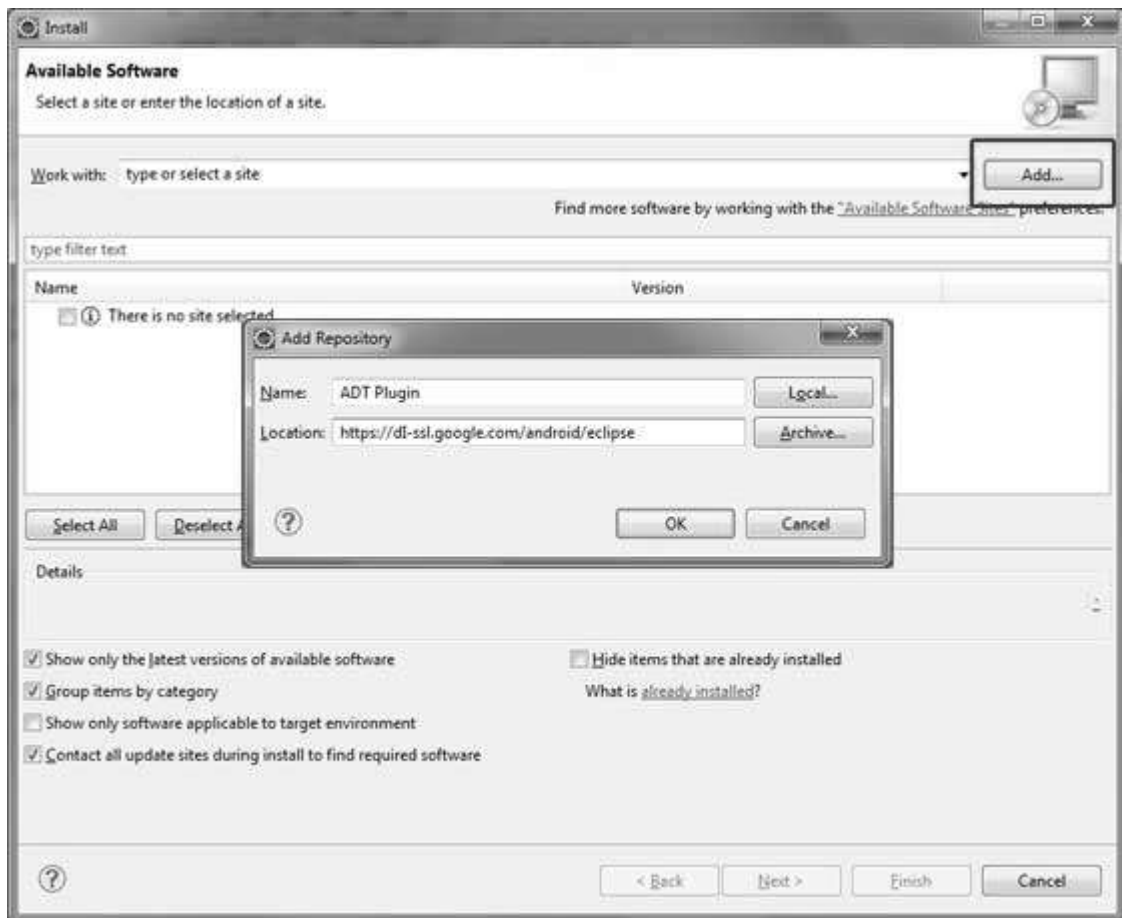
```
$/usr/local/eclipse/eclipse
```

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

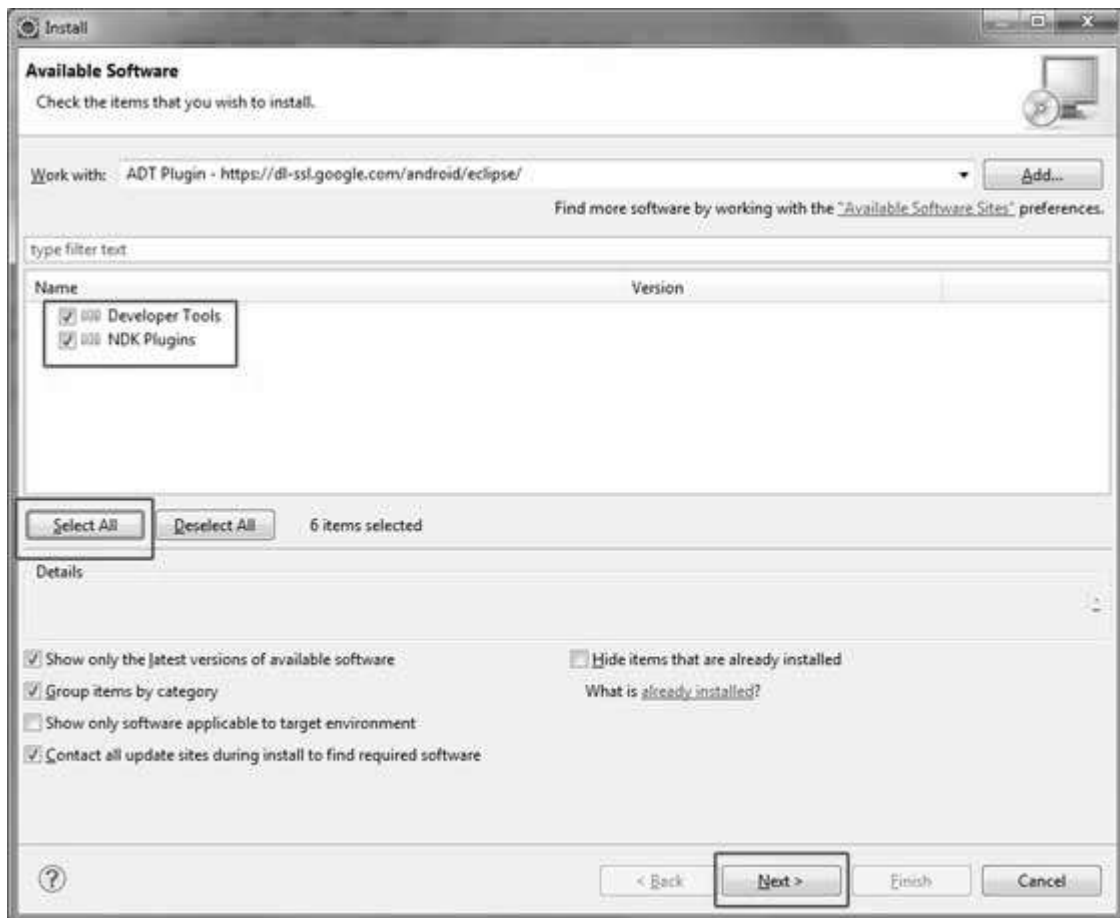


Step 4 - Setup Android Development Tools (ADT) Plugin

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.



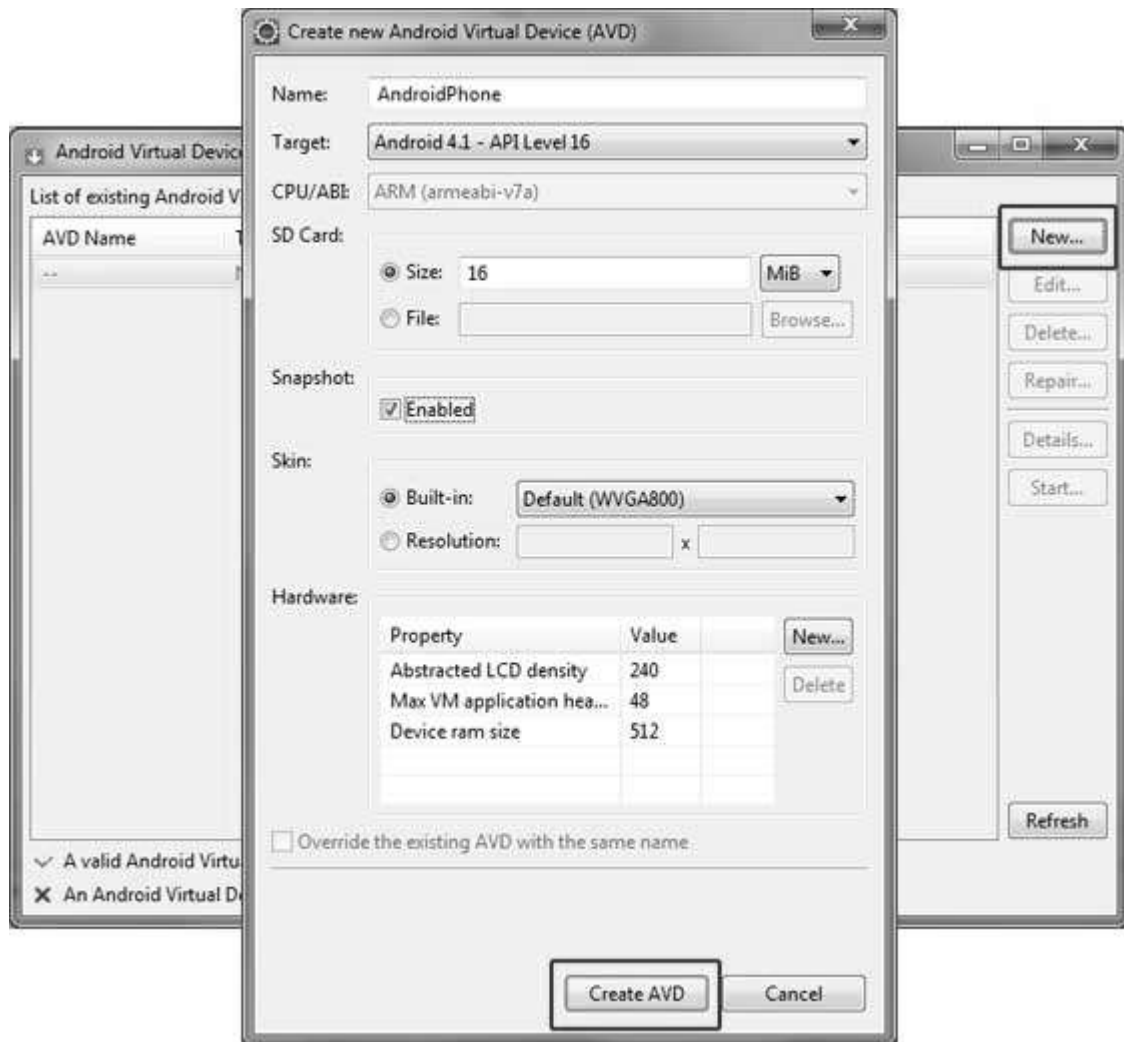
Now use **Add** button to add *ADT Plugin* as name and *https://dl-ssl.google.com/android/eclipse/* as the location. Then click OK to add this location. As soon as you will click OK button to add this location, Eclipse starts searching for the plug-in available in the given location and finally lists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

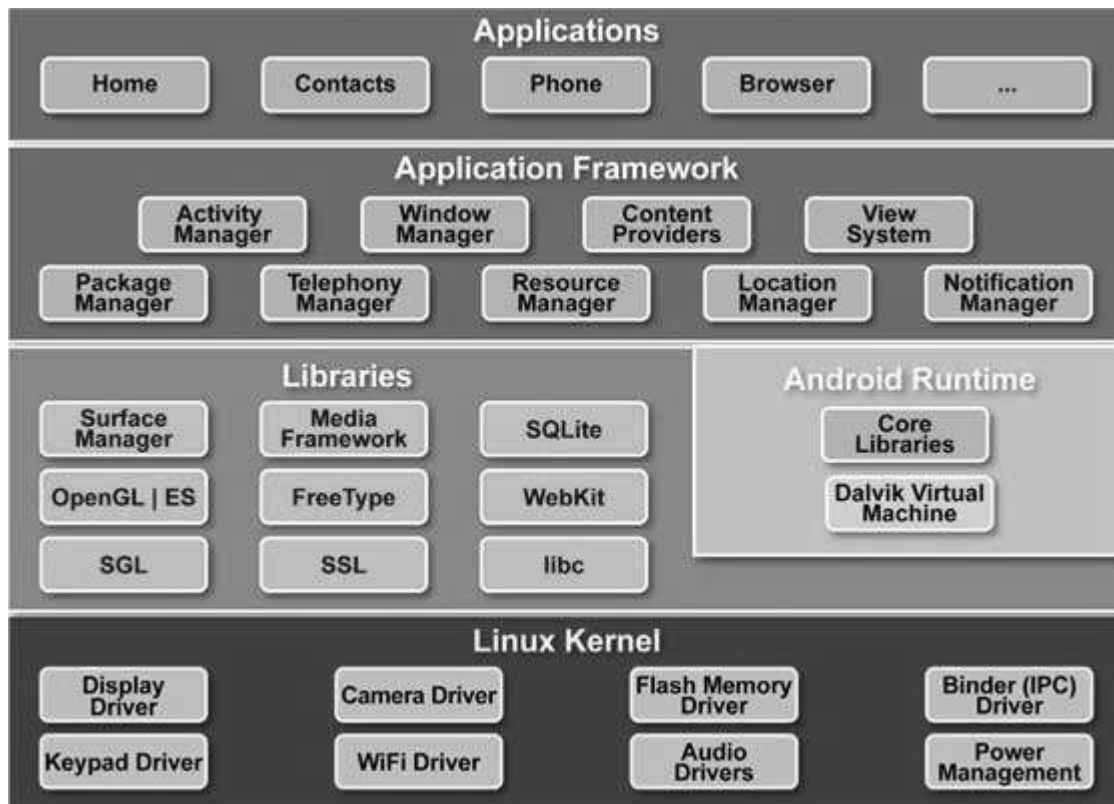
To test your Android applications you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager using Eclipse menu options **Window > AVD Manager** which will launch Android AVD Manager. Use **New** button to create a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

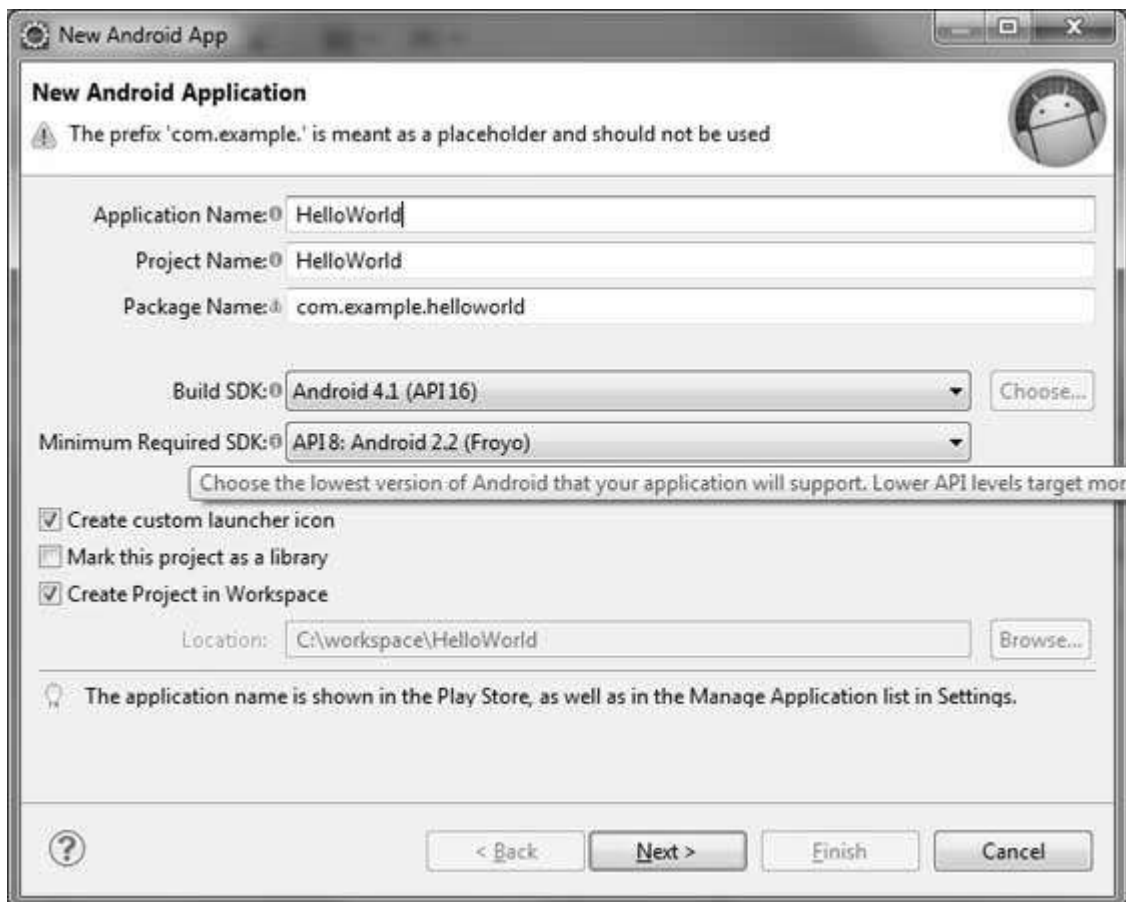
5. HELLO WORLD EXAMPLE

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.

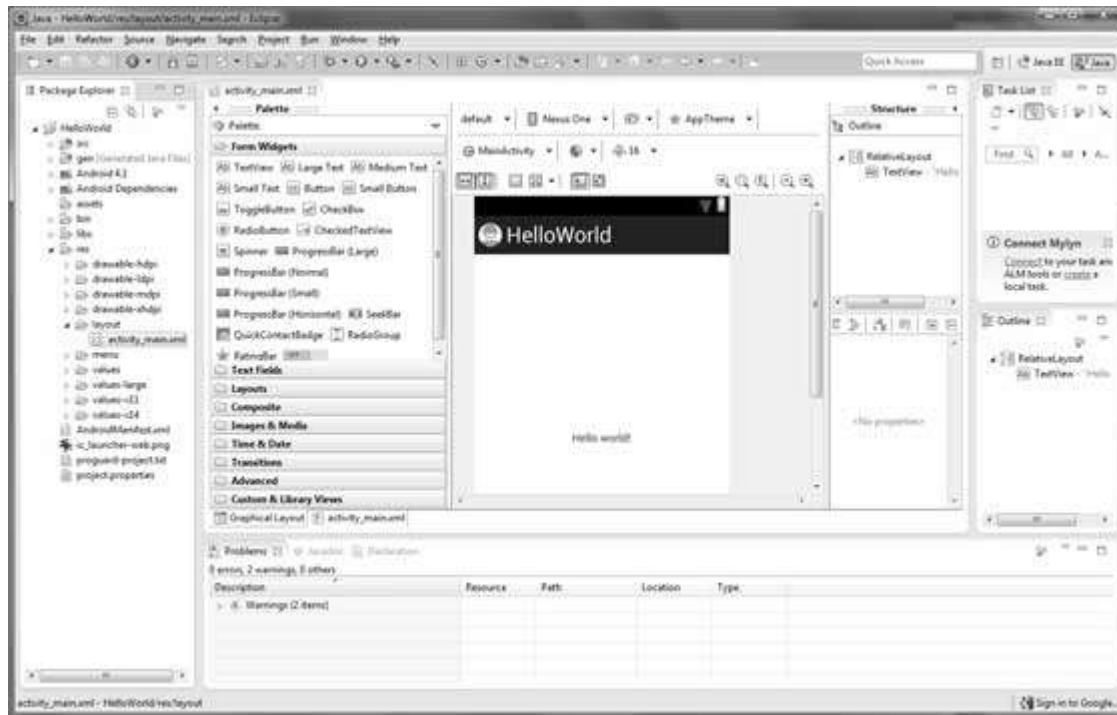
So let us proceed to write a simple Android Application which will print "Hello World!".

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:

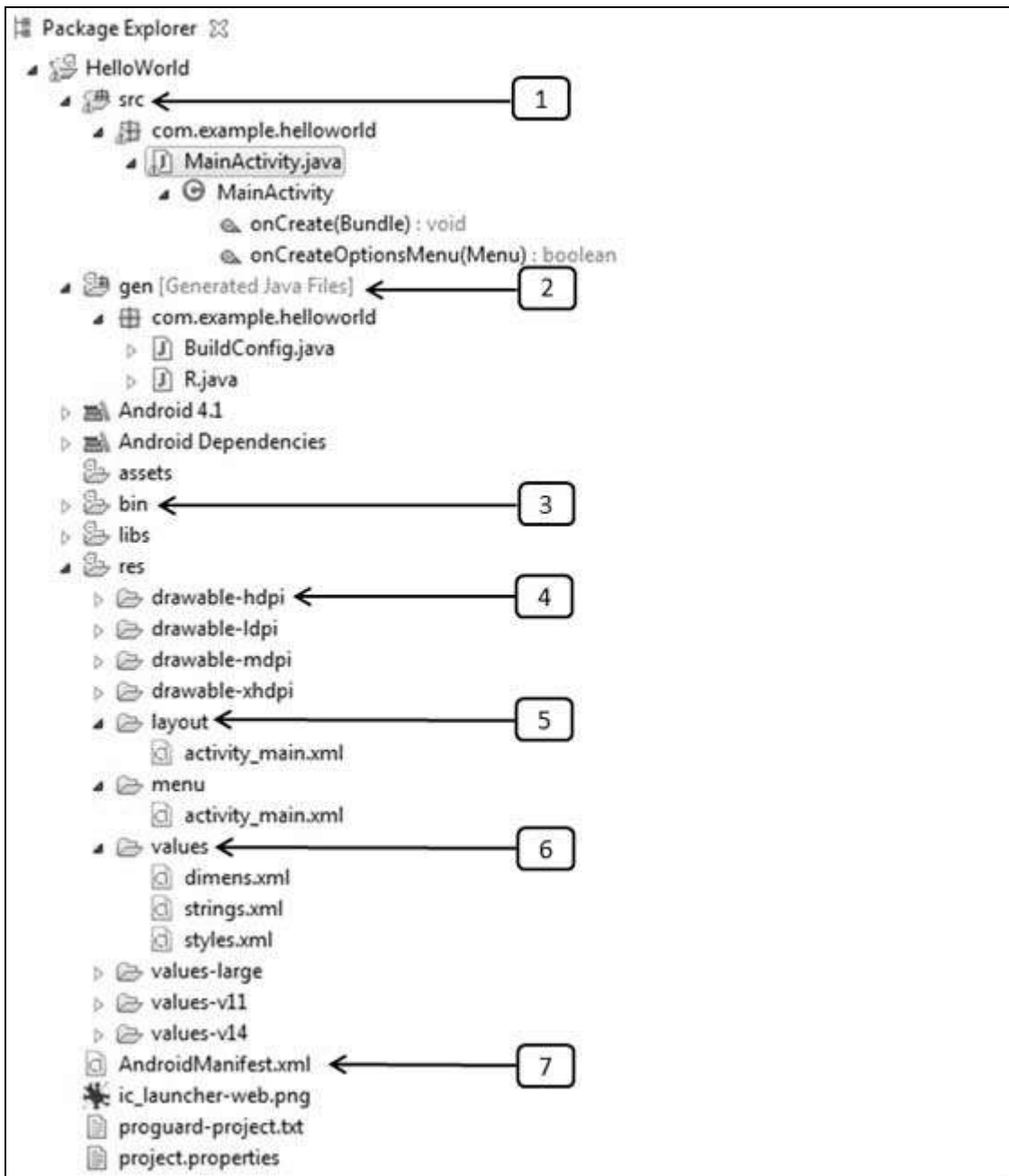


Next, follow the instructions provided and keep all other entries as default till the final step. Once your project is created successfully, you will have the following project screen:



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	<p>src</p> <p>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.</p>
2	<p>gen</p> <p>This contains the .R file, a compiler-generated file that references all the</p>

	resources found in your project. You should not modify this file.
3	<p>bin</p> <p>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.</p>
4	<p>res/drawable-hdpi</p> <p>This is a directory for drawable objects that are designed for high-density screens.</p>
5	<p>res/layout</p> <p>This is a directory for files that define your app's user interface.</p>
6	<p>res/values</p> <p>This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.</p>
7	<p>AndroidManifest.xml</p> <p>This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.</p>

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"


```

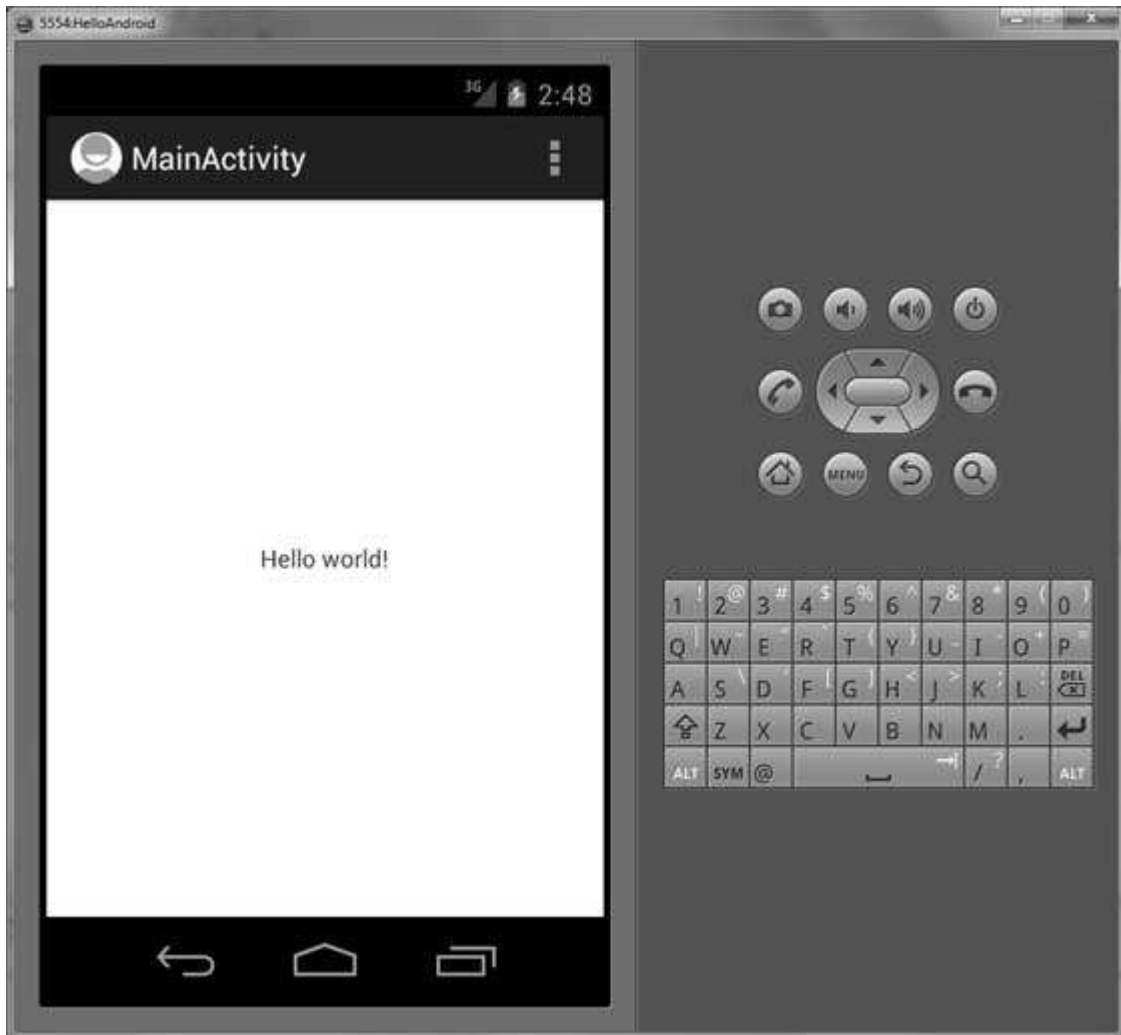
```
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:padding="@dimen/padding_medium"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />

</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. 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:

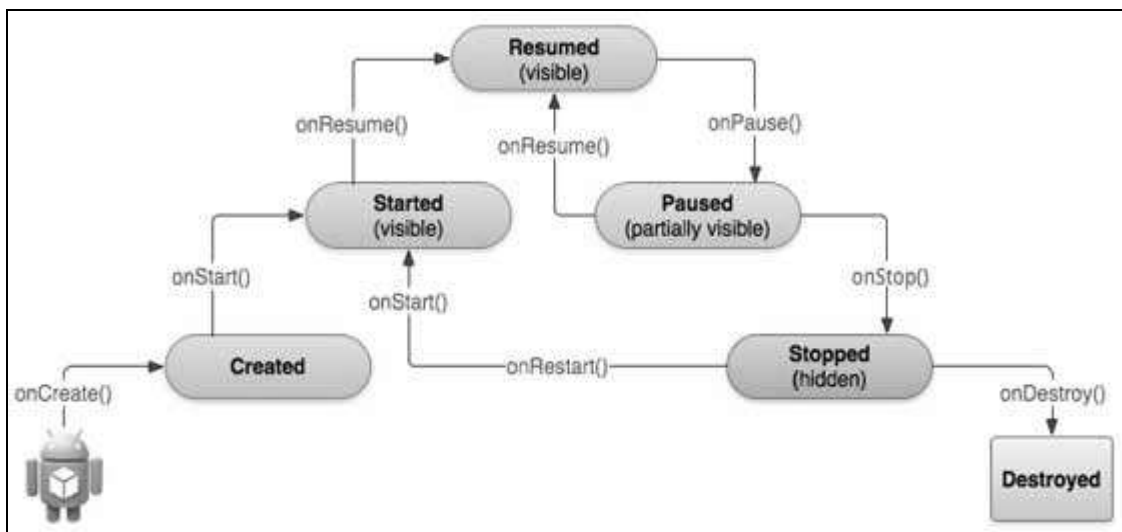
```
<?xml version="1.0" encoding="utf-8"?>  
<EditText xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:textColor="@color/opaque_red"  
    android:text="@string/hello" />
```

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. 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
onCreate()	This is the first callback and called when the activity is first created.
onStart()	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" />

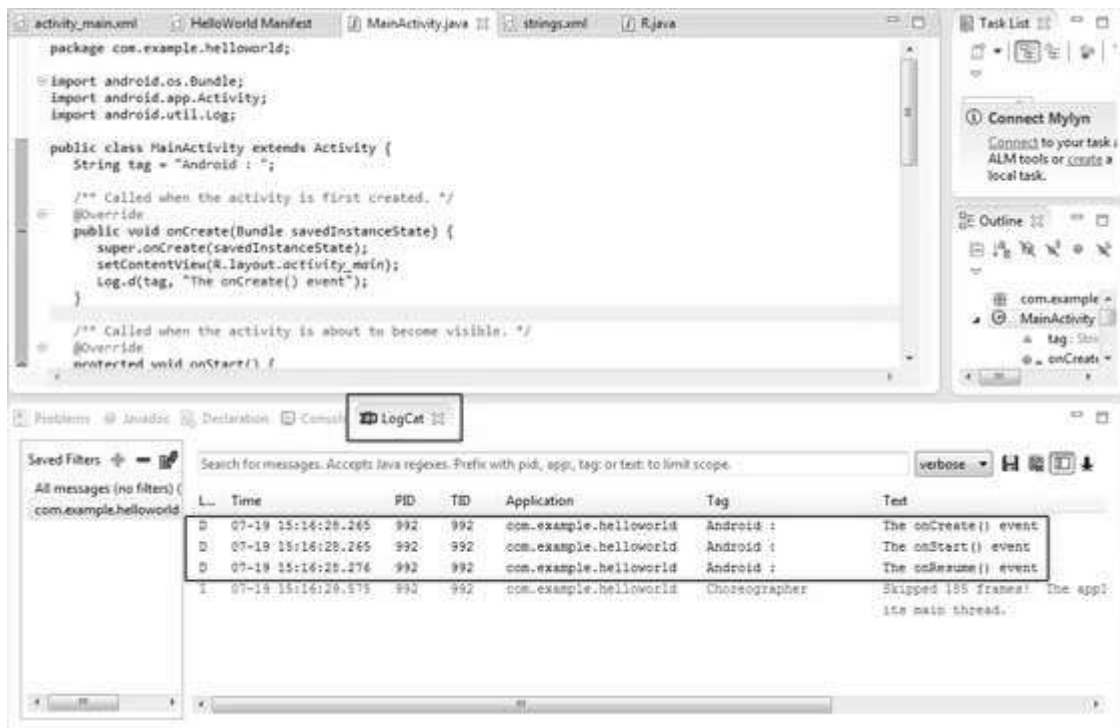
```


```
<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

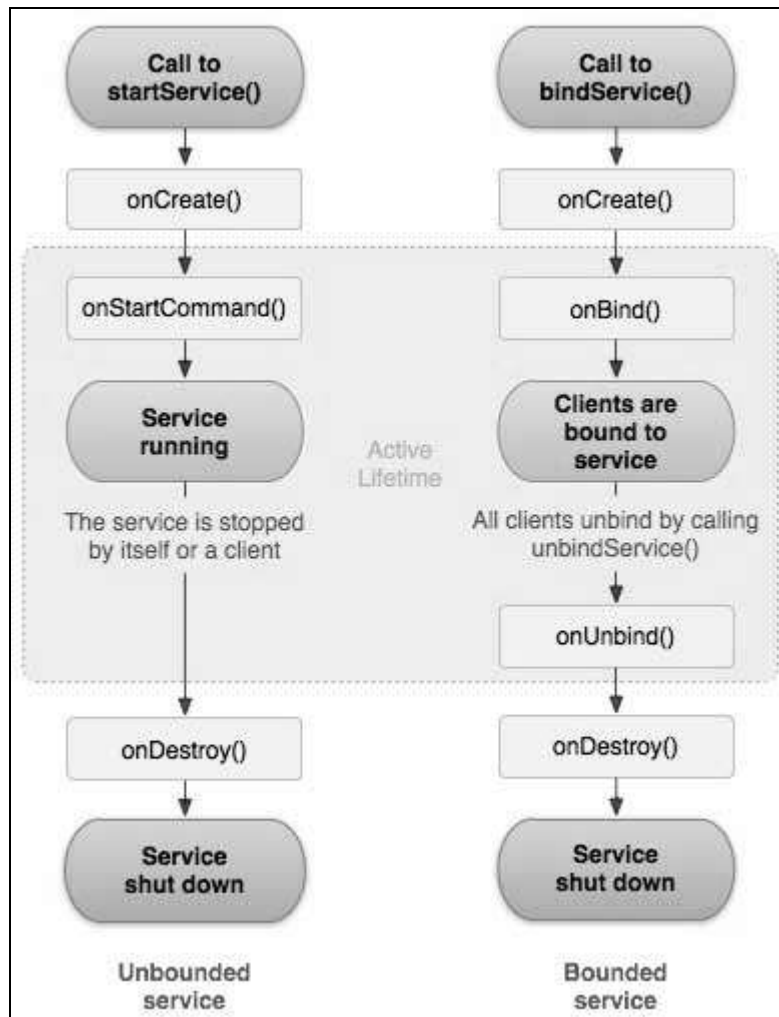
```

8. SERVICES

A service is a component that runs in the background to perform long-running operations without needing to interact with the user. 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 can essentially take two states:

State	Description
Started	A service is started when an application component, such as an activity, starts it by calling <code>startService()</code> . Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
Bound	A service is bound when an application component binds to it by calling <code>bindService()</code> . A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with inter-process communication (IPC).

A service has lifecycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the lifecycle when the service is created with `startService()` and the diagram on the right shows the lifecycle when the service is created with `bindService()`:*(image courtesy : android.com)*



To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callback methods. However, it is important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback	Description
onStartCommand()	The system calls this method when another component, such as an activity, requests that the service be started, by calling <i>startService()</i> . If you implement this method, it is your responsibility to stop the service when its work is done, by calling <i>stopSelf()</i> or <i>stopService()</i> methods.
onBind()	The system calls this method when another component wants to bind with the service by calling <i>bindService()</i> . If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an <i>IBinder</i> object. You must always

	implement this method, but if you don't want to allow binding, then you should return <i>null</i> .
<code>onUnbind()</code>	The system calls this method when all clients have disconnected from a particular interface published by the service.
<code>onRebind()</code>	The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its <code>onUnbind(Intent)</code> .
<code>onCreate()</code>	The system calls this method when the service is first created using <code>onStartCommand()</code> or <code>onBind()</code> . This call is required to perform one-time setup.
<code>onDestroy()</code>	The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

The following skeleton service demonstrates each of the lifecycle methods:

```
package com.tutorialspoint;

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

    /** indicates how to behave if the service is killed */
    int mStartMode;

    /** interface for clients that bind */
    IBinder mBinder;

    /** indicates whether onRebind should be used */
    boolean mAllowRebind;
}
```



```
/** Called when the service is being created. */
@Override
public void onCreate() {

}

/** The service is starting, due to a call to startService() */
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    return mStartMode;
}

/** A client is binding to the service with bindService() */
@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

/** Called when all clients have unbound with unbindService() */
@Override
public boolean onUnbind(Intent intent) {
    return mAllowRebind;
}

/** Called when a client is binding to the service with
bindService()*/
@Override
public void onRebind(Intent intent) {

}

/** Called when The service is no longer used and is being destroyed
*/
```

```

@Override
public void onDestroy() {

}
}

```

Example:

This example will take you through simple steps to show how to create your own Android Service. 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> to add <i>startService()</i> and <i>stopService()</i> methods.
3	Create a new java file <i>MyService.java</i> under the package <i>com.example.helloworld</i> . This file will have implementation of Android service related methods.
4	Define your service in <i>AndroidManifest.xml</i> file using <code><service.../></code> tag. An application can have one or more services without any restrictions.
5	Modify the default content of <i>res/layout/activity_main.xml</i> file to include two buttons in linear layout.
6	Define two constants <i>start_service</i> and <i>stop_service</i> in <i>res/values/strings.xml</i> file.
7	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 can include each of the fundamental lifecycle methods. We have added *startService()* and *stopService()* methods to start and stop the service.

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.content.Intent;
import android.view.View;

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;
    }

    // Method to start the service
    public void startService(View view) {
        startService(new Intent(getApplicationContext(), MyService.class));
    }

    // Method to stop the service
    public void stopService(View view) {
        stopService(new Intent(getApplicationContext(), MyService.class));
    }
}
```

Following is the content of **src/com.example.helloworld/MyService.java**. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods *onStartCommand()* and *onDestroy()*:

```

package com.example.helloworld;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Let it continue running until it is stopped.
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service Destroyed",
            Toast.LENGTH_LONG).show();
    }
}

```

Following is the modified content of *AndroidManifest.xml* file. Here we have added `<service.../>` tag to include our service:

```

<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>
    <service android:name=".MyService" />
</application>
</manifest>

```

Following will be the content of **res/layout/activity_main.xml** file to include two buttons:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/btnStartService"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start_service"
        android:onClick="startService"/>


    <Button android:id="@+id/btnStopService"
        android:layout_width="fill_parent"

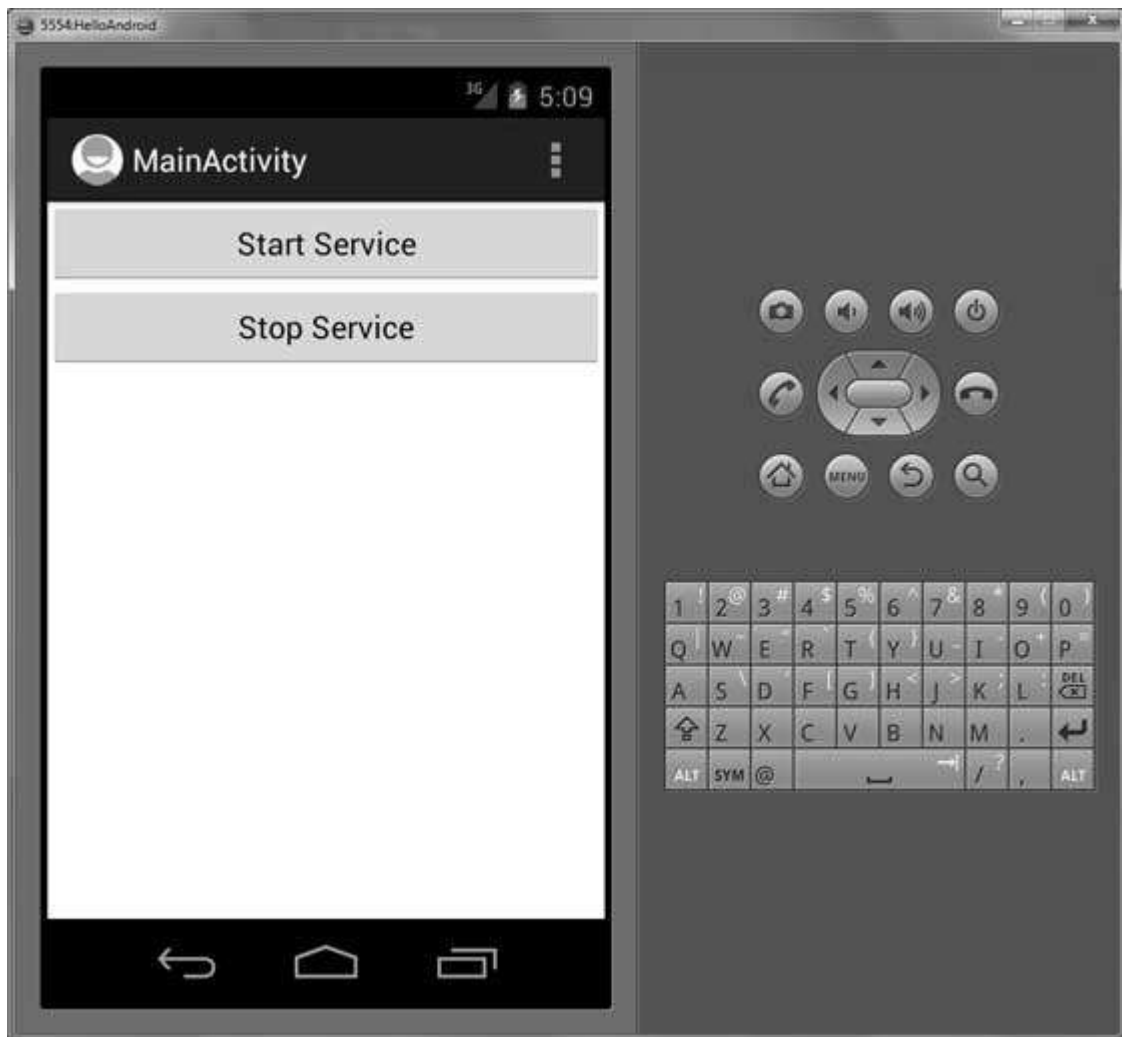
```

```
        android:layout_height="wrap_content"  
        android:text="@string/stop_service"  
        android:onClick="stopService" />  
  
</LinearLayout>
```

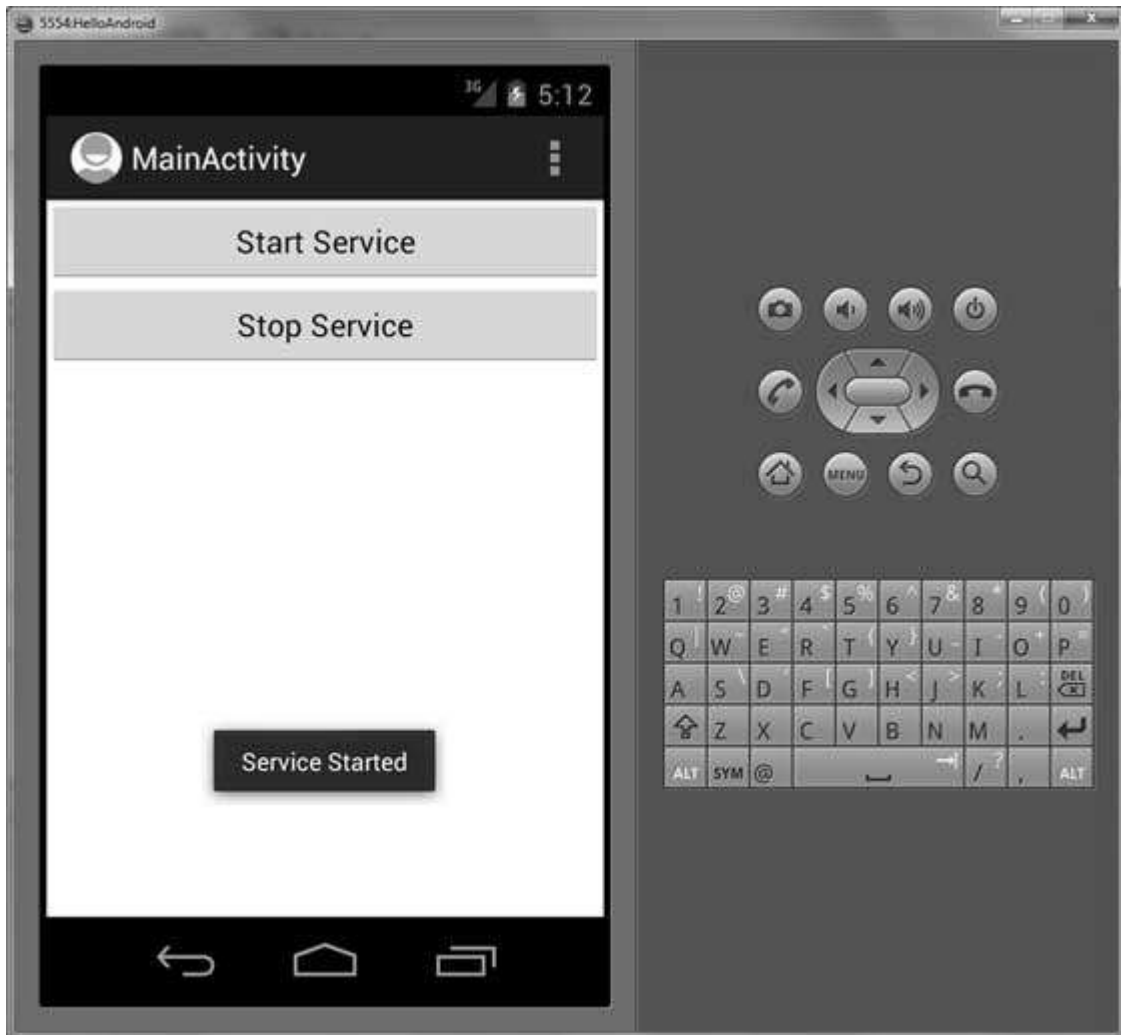
Following will be the content of **res/values/strings.xml** to define two new constants:

```
<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>  
    <string name="start_service">Start Service</string>  
    <string name="stop_service">Stop Service</string>  
  
</resources>
```

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 following Emulator window:



Now to start your service, let's click on **Start Service** button, this will start the service and as per our programming in `onStartCommand()` method, a message *Service Started* will appear on the bottom of the simulator as follows:



To stop the service, you can click the Stop Service button.

9. BROADCAST RECEIVERS

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. 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.

There are following two important steps to make BroadcastReceiver work for the system broadcasted intents:

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

There is one additional step in case you are going to implement your custom intents; then you will have to create and broadcast those intents.

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the `onReceive()` method where each message is received as an **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver {  
  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(context, "Intent Detected.",  
            Toast.LENGTH_LONG).show();  
    }  
  
}
```

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event `ACTION_BOOT_COMPLETED` which is fired by the system once the Android system has completed the boot process.

```

<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <receiver android:name="MyReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED">
        </action>
        </intent-filter>
    </receiver>

</application>

```

Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver* and implemented logic inside *onReceive()* will be executed.

There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

Event Constant	Description
android.intent.action.BATTERY_CHANGED	Sticky broadcast containing the charging state, level, and other information about the battery.
android.intent.action.BATTERY_LOW	Indicates low battery condition on the device.
android.intent.action.BATTERY_OKAY	Indicates the battery is now okay after being low.
android.intent.action.BOOT_COMPLETED	This is broadcast once, after the system has finished booting.
android.intent.action.BUG_REPORT	Show activity for reporting a bug.
android.intent.action.CALL	Perform a call to someone specified

	by the data.
android.intent.action.CALL_BUTTON	The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
android.intent.action.DATE_CHANGED	The date has changed.
android.intent.action.REBOOT	Have the device reboot.

Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the `sendBroadcast()` method inside your activity class. If you use the `sendStickyBroadcast(Intent)` method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

```
public void broadcastIntent(View view)
{
    Intent intent = new Intent();
    intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
    sendBroadcast(intent);
}
```

This intent `com.tutorialspoint.CUSTOM_INTENT` can also be registered in similar way as we have registered system generated intent.

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <receiver android:name="MyReceiver">
        <intent-filter>
            <action android:name="com.tutorialspoint.CUSTOM_INTENT">
        </action>
        </intent-filter>
```

```
</receiver>

</application>
```

Example:

This example will explain you how to create *BroadcastReceiver* to intercept custom intent. Once you are familiar with custom intent, then you can program your application to intercept system generated intents. So let's 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> to add <i>broadcastIntent()</i> method.
3	Create a new java file called <i>MyReceiver.java</i> under the package <i>com.example.helloworld</i> to define a <i>BroadcastReceiver</i> .
4	An application can handle one or more custom and system intents without any restrictions. Every indent you want to intercept must be registered in your <i>AndroidManifest.xml</i> file using <code><receiver.../></code> tag.
5	Modify the default content of <i>res/layout/activity_main.xml</i> file to include a button to broadcast intent.
6	Define a constant <i>broadcast_intent</i> in <i>res/values/strings.xml</i> file.
7	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 can include each of the fundamental lifecycle methods. We have added *broadcastIntent()* method to broadcast a custom intent.

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

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.content.Intent;
import android.view.View;

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;
    }

    // broadcast a custom intent.
    public void broadcastIntent(View view)
    {
        Intent intent = new Intent();
        intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
        sendBroadcast(intent);
    }
}
```

Following is the content of **src/com.example.helloworld/MyReceiver.java**:

```
package com.example.helloworld;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.widget.Toast;

public class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent Detected.",
            Toast.LENGTH_LONG).show();
    }
}
```

Following will be the modified content of *AndroidManifest.xml* file. Here we have added `<service.../>` tag to include our service:

```
<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>
```

```

    <receiver android:name="MyReceiver">
        <intent-filter>

            <action android:name="com.tutorialspoint.CUSTOM_INTENT">
            </action>
        </intent-filter>
    </receiver>
</application>
</manifest>

```

Following will be the content of **res/layout/activity_main.xml** file to include a button to broadcast our custom intent:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/btnStartService"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/broadcast_intent"
        android:onClick="broadcastIntent"/>

</LinearLayout>

```


Following will be the content of **res/values/strings.xml** to define two new constants:

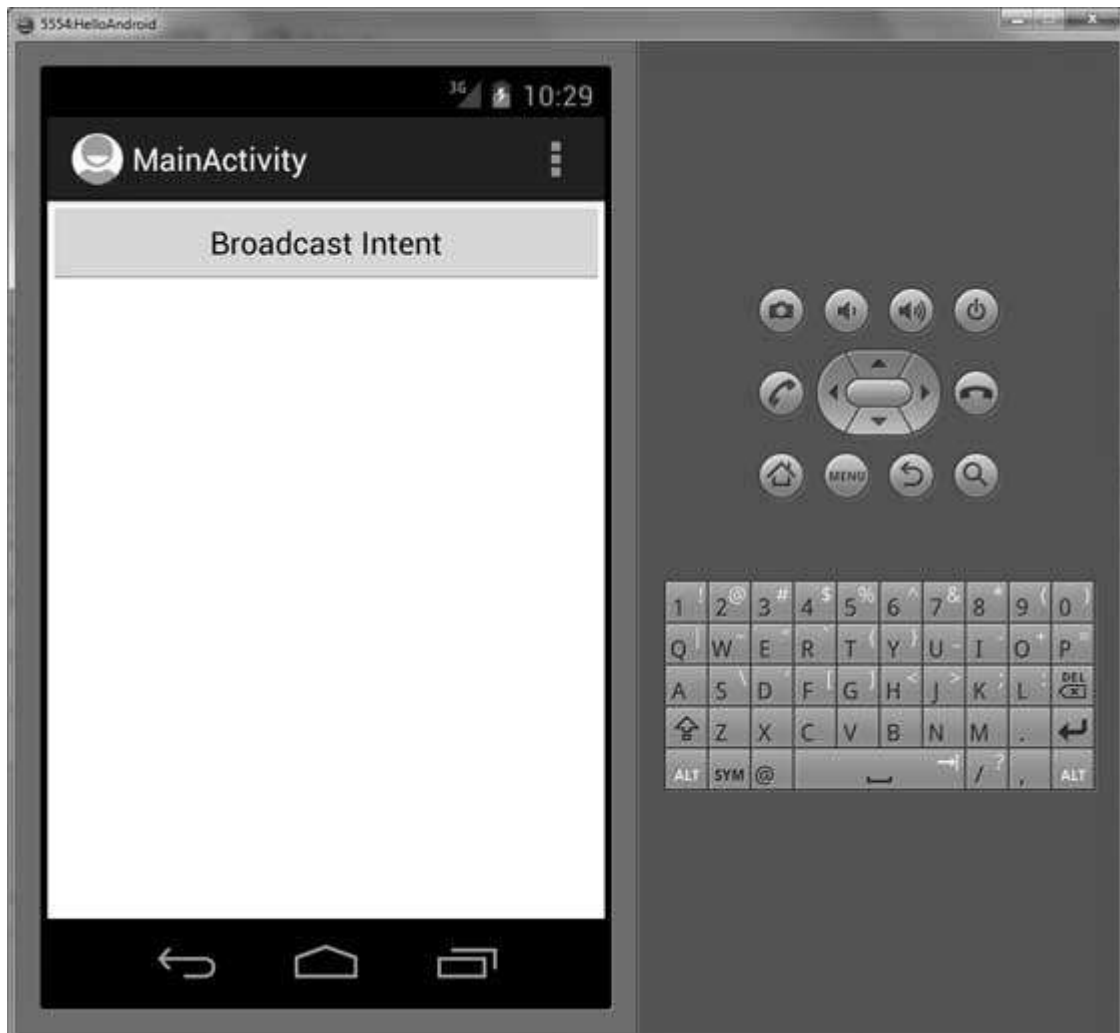
```

<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>
    <string name="broadcast_intent">Broadcast Intent</string>

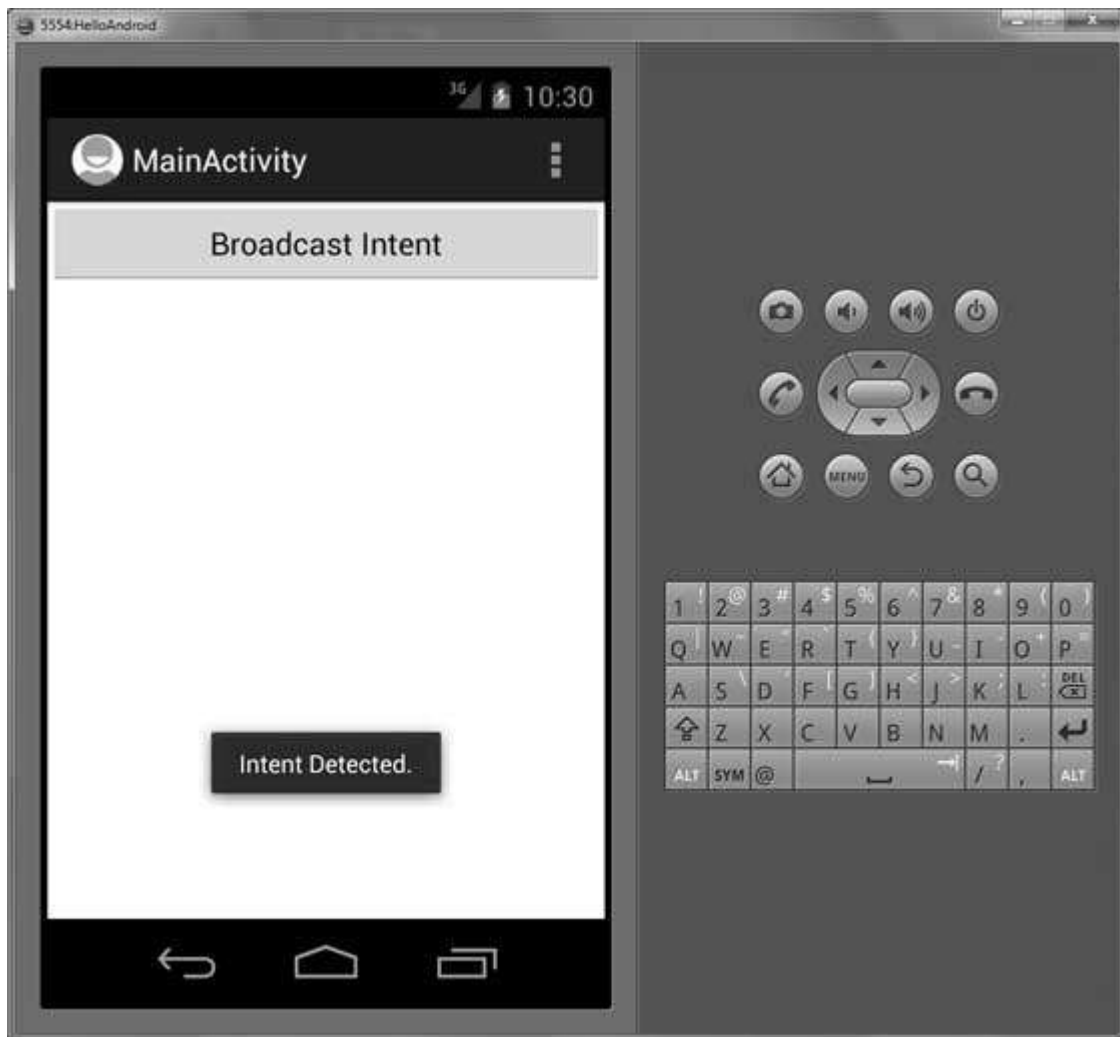
</resources>

```

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 following Emulator window:



Now to broadcast our custom intent, let's click on **Broadcast Intent** button, this will broadcast our custom intent "*com.tutorialspoint.CUSTOM_INTENT*" which will be intercepted by our registered BroadcastReceiver i.e. MyReceiver and as per our implemented logic a toast will appear on the bottom of the simulator as follows:



You can try implementing other BroadcastReceiver to intercept system generated intents like system boot-up, date changed, low battery etc.

10. CONTENT PROVIDERS

A content provider component supplies data from one application to other on request. Such requests are handled by the methods of the `ContentResolver` class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

Each Android application runs in its own process with its own permissions which keeps an application data hidden from another application. But sometimes it is required to share data across applications. This is where content providers become very useful.

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using `insert()`, `update()`, `delete()`, and `query()` methods. In most cases this data is stored in an **SQLite** database.

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
{
}

```

Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format:

```
<prefix>://<authority>/<data_type>/<id>
```

Here is the detail of various parts of the URI:

Part	Description
prefix	This is always set to content://

authority	This specifies the name of the content provider, for example <i>contacts</i> , <i>browser</i> etc. For third-party content providers, this could be the fully qualified name, such as <i>com.tutorialspoint.statusprovider</i>
data_type	This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the <i>Contacts</i> content provider, then the data path would be <i>people</i> and URI would look like this <i>content://contacts/people</i>
id	This specifies the specific record requested. For example, if you are looking for contact number 5 in the <i>Contacts</i> content provider then URI would look like this <i>content://contacts/people/5</i> .

Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the *ContentProviderbase* class.
- Secondly, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using `<provider>` tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working:

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()** This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.

- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Example:

This example will explain you how to create your own *ContentProvider*. So let's follow the following steps similar to what we followed while creating *Hello World Example*:

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>MyContentProvider</i> under a package <i>com.example.mycontentprovider</i> , with blank Activity.
2	Modify main activity file <i>MainActivity.java</i> to add two new methods <i>onClickAddName()</i> and <i>onClickRetrieveStudents()</i> .
3	Create a new java file called <i>StudentsProvider.java</i> under the package <i>com.example.mycontentprovider</i> to define your actual provider and associated methods.
4	Register your content provider in your <i>AndroidManifest.xml</i> file using <code><provider.../></code> tag.
5	Modify the default content of <i>res/layout/activity_main.xml</i> file to include a small GUI to add students records.
6	Define required constants in <i>res/values/strings.xml</i> file.
7	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.mycontentprovider/MainActivity.java**. This file can include each of the fundamental lifecycle methods. We have added two new methods *onClickAddName()* and *onClickRetrieveStudents()* to handle user interaction with the application.

```
package com.example.mycontentprovider;
```



```
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.ContentValues;
import android.content.CursorLoader;
import android.database.Cursor;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void onClickAddName(View view) {
        // Add a new student record
        ContentValues values = new ContentValues();

        values.put(StudentsProvider.NAME,
            ((EditText)findViewById(R.id.txtName)).getText().toString());

        values.put(StudentsProvider.GRADE,
```

```

        ((EditText)findViewById(R.id.txtGrade)).getText().toString());

        Uri uri = getContentResolver().insert(
        StudentsProvider.CONTENT_URI, values);

        Toast.makeText(getBaseContext(),
        uri.toString(), Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveStudents(View view) {
        // Retrieve student records
        String URL = "content://com.example.provider.College/students";
        Uri students = Uri.parse(URL);
        Cursor c = managedQuery(students, null, null, null, "name");
        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,
                c.getString(c.getColumnIndex(StudentsProvider._ID)) +
                ", " + c.getString(c.getColumnIndex( StudentsProvider.NAME))
                +
                ", " + c.getString(c.getColumnIndex(
                StudentsProvider.GRADE)),
                Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }
}

```

Create a new file `StudentsProvider.java` under `com.example.mycontentprovider` package and following is the content of **`src/com.example.mycontentprovider/StudentsProvider.java`**:

```

package com.example.mycontentprovider;

import java.util.HashMap;

```

```
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;

public class StudentsProvider extends ContentProvider {

    static final String PROVIDER_NAME = "com.example.provider.College";
    static final String URL = "content://" + PROVIDER_NAME + "/students";
    static final Uri CONTENT_URI = Uri.parse(URL);

    static final String _ID = "_id";
    static final String NAME = "name";
    static final String GRADE = "grade";

    private static HashMap<String, String> STUDENTS_PROJECTION_MAP;

    static final int STUDENTS = 1;
    static final int STUDENT_ID = 2;

    static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "students", STUDENTS);
        uriMatcher.addURI(PROVIDER_NAME, "students/#", STUDENT_ID);
    }
}
```

```
/**
 * Database specific constant declarations
 */
private SQLiteDatabase db;
static final String DATABASE_NAME = "College";
static final String STUDENTS_TABLE_NAME = "students";
static final int DATABASE_VERSION = 1;
static final String CREATE_DB_TABLE =
    " CREATE TABLE " + STUDENTS_TABLE_NAME +
    " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
    " name TEXT NOT NULL, " +
    " grade TEXT NOT NULL);";

/**
 * Helper class that actually creates and manages
 * the provider's underlying data repository.
 */
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context){
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(CREATE_DB_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
        int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + STUDENTS_TABLE_NAME);
        onCreate(db);
    }
}
```



```

    }
}

@Override
public boolean onCreate() {
    Context context = getContext();
    DatabaseHelper dbHelper = new DatabaseHelper(context);
    /**
     * Create a write able database which will trigger its
     * creation if it doesn't already exist.
     */
    db = dbHelper.getWritableDatabase();
    return (db == null)? false:true;
}

@Override
public Uri insert(Uri uri, ContentValues values) {
    /**
     * Add a new student record
     */
    long rowID = db.insert(STUDENTS_TABLE_NAME, "", values);
    /**
     * If record is added successfully
     */
    if (rowID > 0)
    {
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLException("Failed to add a record into " + uri);
}

@Override

```

```

public Cursor query(Uri uri, String[] projection, String selection,
                   String[] selectionArgs, String sortOrder) {

    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(STUDENTS_TABLE_NAME);

    switch (uriMatcher.match(uri)) {
    case STUDENTS:
        qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
        break;
    case STUDENT_ID:
        qb.appendWhere( "_ID = " + uri.getPathSegments().get(1));
        break;
    default:
        throw new IllegalArgumentException("Unknown URI " + uri);
    }
    if (sortOrder == null || sortOrder == ""){
        /**
         * By default sort on student names
         */
        sortOrder = NAME;
    }
    Cursor c = qb.query(db,      projection, selection, selectionArgs,
                       null, null, sortOrder);

    /**
     * register to watch a content URI for changes
     */
    c.setNotificationUri(getContext().getContentResolver(), uri);

    return c;
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {

```

```

int count = 0;

switch (uriMatcher.match(uri)){
case STUDENTS:
    count = db.delete(STUDENTS_TABLE_NAME, selection,
        selectionArgs);
    break;
case STUDENT_ID:
    String id = uri.getPathSegments().get(1);
    count = db.delete( STUDENTS_TABLE_NAME, _ID + " = " + id +
        (!TextUtils.isEmpty(selection) ? " AND (" +
            selection + ')' : "")), selectionArgs);
    break;
default:
    throw new IllegalArgumentException("Unknown URI " + uri);
}

getContext().getContentResolver().notifyChange(uri, null);
return count;
}

@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {
    int count = 0;

    switch (uriMatcher.match(uri)){
    case STUDENTS:
        count = db.update(STUDENTS_TABLE_NAME, values,
            selection, selectionArgs);
        break;
    case STUDENT_ID:
        count = db.update(STUDENTS_TABLE_NAME, values, _ID +
            " = " + uri.getPathSegments().get(1) +

```

```

        (!TextUtils.isEmpty(selection) ? " AND (" +
        selection + ')' : ""), selectionArgs);

        break;
    default:
        throw new IllegalArgumentException("Unknown URI " + uri );
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}

@Override
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)){
        /**
         * Get all student records
         */
        case STUDENTS:
            return "vnd.android.cursor.dir/vnd.example.students";
        /**
         * Get a particular student
         */
        case STUDENT_ID:
            return "vnd.android.cursor.item/vnd.example.students";
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}
}
}

```

Following will be the modified content of *AndroidManifest.xml* file. Here we have added <provider.../> tag to include our content provider:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mycontentprovider"

```

```

    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.mycontentprovider.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
            />
            </intent-filter>
        </activity>
        <provider android:name="StudentsProvider"
            android:authorities="com.example.provider.College">
        </provider>
    </application>

</manifest>

```

Following will be the content of **res/layout/activity_main.xml** file to include a button to broadcast your custom intent:

```

<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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Name" />
    <EditText
    android:id="@+id/txtName"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent" />
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Grade" />
    <EditText
    android:id="@+id/txtGrade"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent" />
    <Button
    android:text="Add Name"
    android:id="@+id/btnAdd"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickAddName" />
    <Button
    android:text="Retrieve Students"
    android:id="@+id/btnRetrieve"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickRetrieveStudents" />
</LinearLayout>

```

Make sure you have following content of **res/values/strings.xml** file:

```

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

    <string name="app_name">MyContentProvider</string>


```

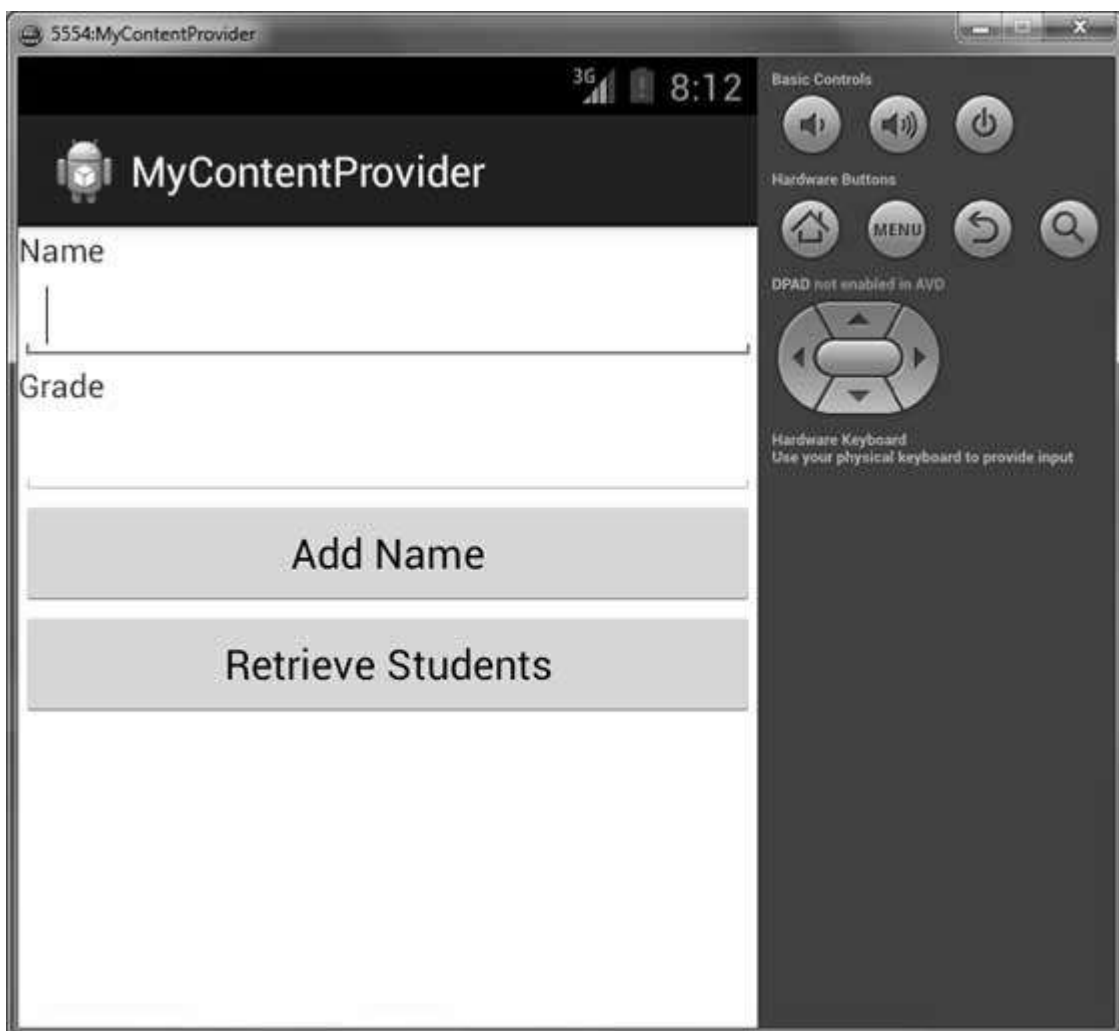
```

    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>

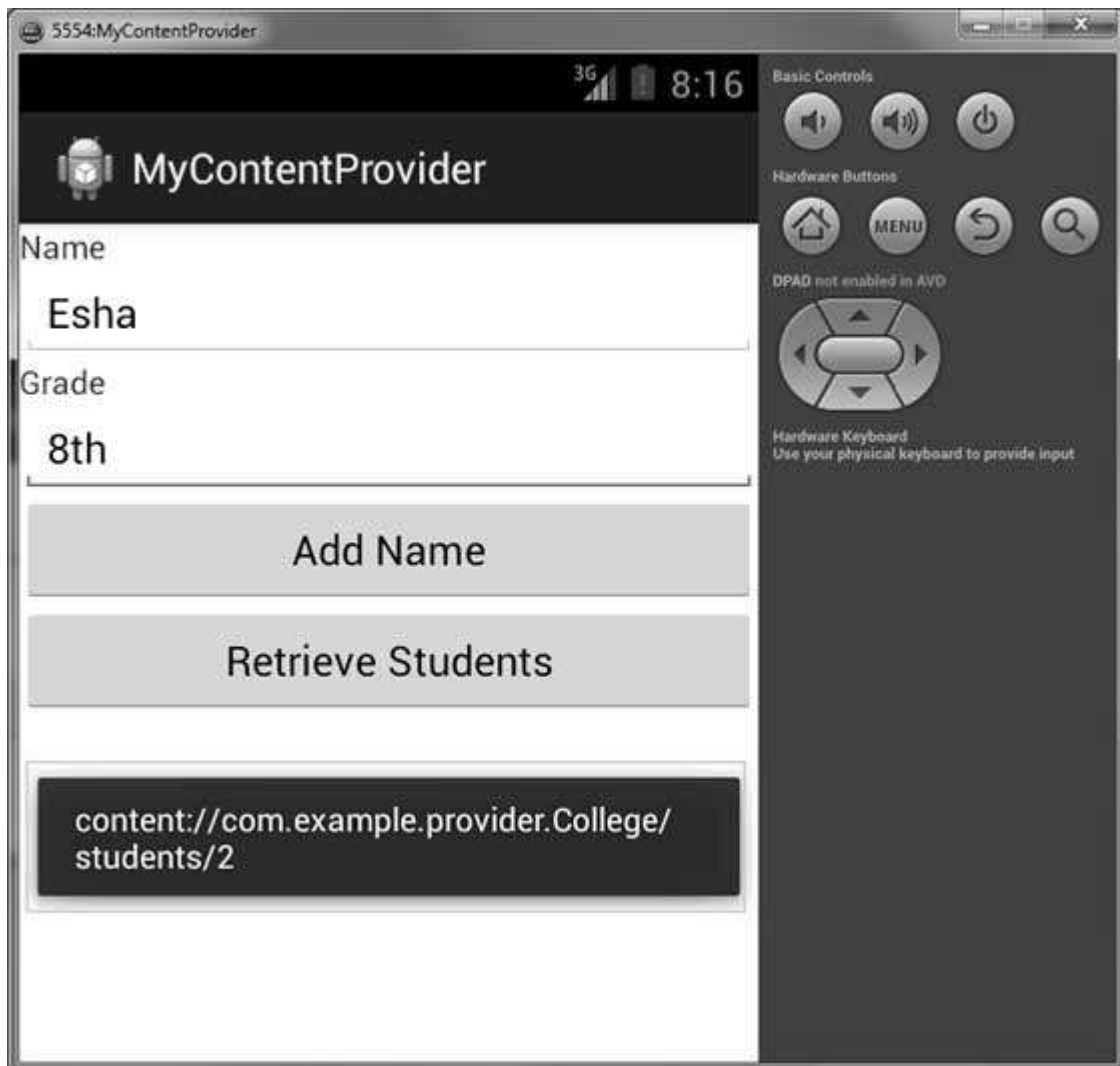
</resources>;

```

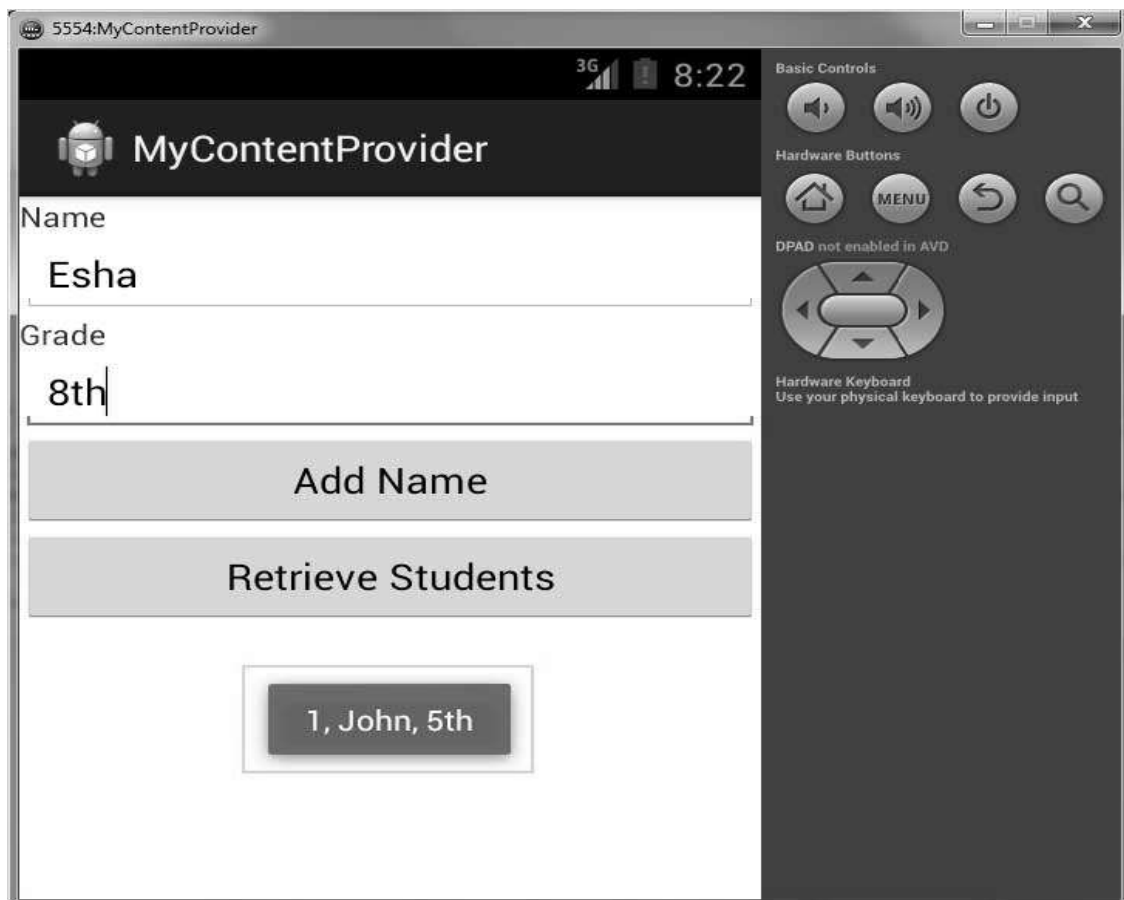
Let's try to run our modified **MyContentProvider** application we just created. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window, be patience because it may take some time based on your computer speed:



Now let's enter student **Name** and **Grade** and finally click on **Add Name** button, this will add student record in the database and will flash a message at the bottom showing ContentProvider URI along with record number added in the database. This operation makes use of our **insert()** method. Let's repeat this process to add few more students in the database of our content provider.



Once you are done with adding records in the database, now its time to ask ContentProvider to give us those records back, so let's click **Retrieve Students** button which will fetch and display all the records one by one which is as per our implementation of our **query()** method.



You can write activities against update and delete operations by providing callback functions in **MainActivity.java** file and then modify user interface to have buttons for update and deleted operations in the same way as we have done for add and read operations.

This way you can use existing Content Provider like Address Book or you can use Content Provider concept in developing nice database oriented applications where you can perform all sort of database operations like read, write, update and delete as explained above in the example.

11. FRAGMENTS

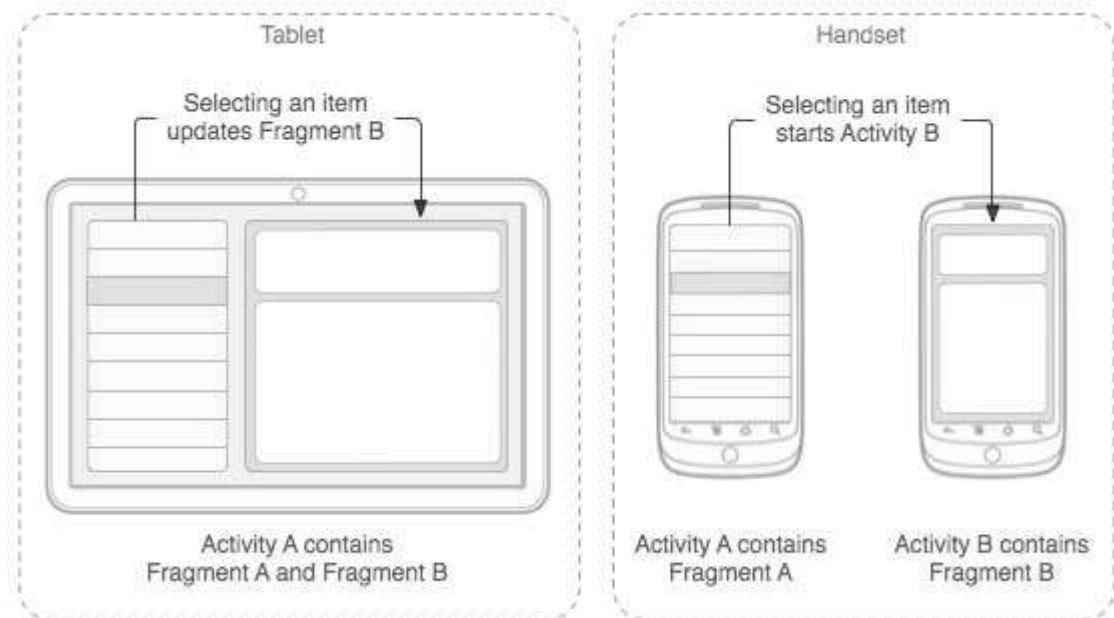
A Fragment is a piece of an application's user interface or behavior that can be placed in an Activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**. Following are the important points about fragment:

- A fragment has its own layout and its own behavior with its own lifecycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the lifecycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behavior that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which has API version 11.

You create fragments by extending **Fragment** class and you can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete lifecycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.

Phase I: When a fragment gets created, it goes through the following states:

onAttach()
 onCreate()
 onCreateView()
 onActivityCreated()

Phase II: When the fragment becomes visible, it goes through these states:

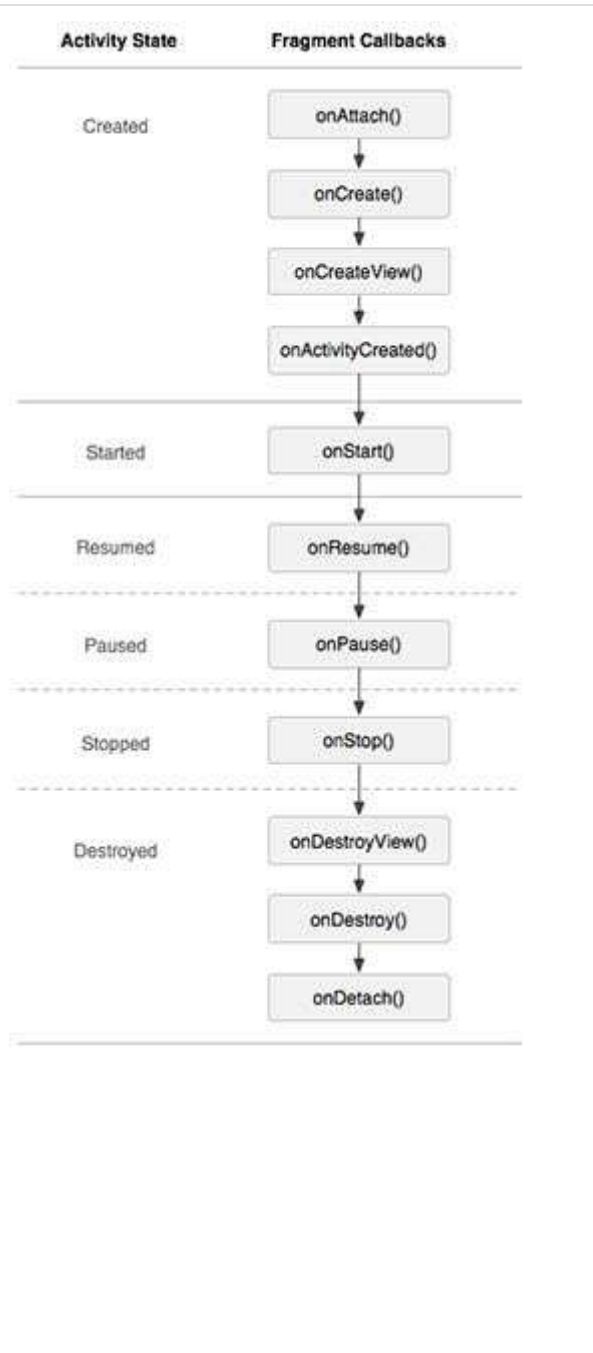
onStart()
 onResume()

Phase III: When the fragment goes into the background mode, it goes through these states:

onPaused()
 onStop()

Phase IV: When the fragment is destroyed, it goes through the following states:

onPaused()
 onStop()
 onDestroyView()
 onDestroy()
 onDetach()



How to use Fragments?

This involves number of simple steps to create Fragments.

- First of all decide how many fragments you want to use in an activity. For example, we want to use two fragments to handle landscape and portrait modes of the device.
- Next, based on number of fragments, create classes which will extend the *Fragment* class. The *Fragment* class has above mentioned callback functions. You can override any of the functions based on your requirements.

- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Here is the list of important methods which you can override in your fragment class:

- **onCreate()** The system calls this when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView()** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **onPause()** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.

Example:

This example will explain you how to create your own *Fragments*. Here we will create two fragments and one of them will be used when device is in landscape mode and another fragment will be used in case of portrait mode. So let's follow the below mentioned steps similar to what we followed while creating *Hello World Example*:

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>MyFragments</i> under a package <i>com.example.myfragments</i> , with blank Activity.
2	Modify main activity file <i>MainActivity.java</i> as shown below in the code. Here we will check orientation of the device and accordingly we will switch between different fragments.
3	Create a two java files <i>PM_Fragment.java</i> and <i>LM_Fragment.java</i> under the package <i>com.example.myfragments</i> to define your fragments and associated methods.
4	Create layout files <i>res/layout/lm_fragment.xml</i> and <i>res/layout/pm_fragment.xml</i> and

	define your layouts for both the fragments.
5	Modify the default content of <i>res/layout/activity_main.xml</i> file to include both the fragments.
6	Define required constants in <i>res/values/strings.xml</i> file.
7	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.mycontentprovider/MainActivity.java**:

```
package com.example.myfragments;

import android.os.Bundle;
import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.res.Configuration;
import android.view.WindowManager;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Configuration config = getResources().getConfiguration();

        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
            fragmentManager.beginTransaction();

        /**
```

```

    * Check the device orientation and act accordingly
    */
    if (config.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        /**
         * Landscape mode of the device
         */
        LM_Fragment ls_fragment = new LM_Fragment();
        fragmentTransaction.replace(android.R.id.content, ls_fragment);
    }else{
        /**
         * Portrait mode of the device
         */
        PM_Fragment pm_fragment = new PM_Fragment();
        fragmentTransaction.replace(android.R.id.content, pm_fragment);
    }
    fragmentTransaction.commit();
}
}
}

```

Create two fragment files **LM_Fragment.java** and **PM_Fragment.java** under *com.example.mycontentprovider* package.

Following is the content of **LM_Fragment.java** file:

```

package com.example.myfragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class LM_Fragment extends Fragment{
    @Override

```

```

public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
    /**
     * Inflate the layout for this fragment
     */
    return inflater.inflate(
        R.layout.lm_fragment, container, false);
}
}

```

Following is the content of **PM_Fragment.java** file:

```

package com.example.myfragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class PM_Fragment extends Fragment{
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        /**
         * Inflate the layout for this fragment
         */
        return inflater.inflate(
            R.layout.pm_fragment, container, false);
    }
}
}

```

Create two layout files: **lm_fragment.xml** and **pm_fragment.xml** under *res/layout* directory.

Following is the content of **lm_fragments.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#7bae16">

    <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/landscape_message"
      android:textColor="#000000"
      android:textSize="20px" />

    <!-- More GUI components go here -->

  </LinearLayout>
```

Following is the content of **pm_fragments.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#666666">

    <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/portrait_message"
      android:textColor="#000000"
```

```

        android:textSize="20px" />

<!-- More GUI components go here -->

</LinearLayout>

```

Following will be the content of **res/layout/activity_main.xml** file which includes your fragments:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

    <fragment
        android:name="com.example.fragments"
        android:id="@+id/lm_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <fragment
        android:name="com.example.fragments"
        android:id="@+id/pm_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>

```

Make sure you have following content of **res/values/strings.xml** file:

```

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

```

```

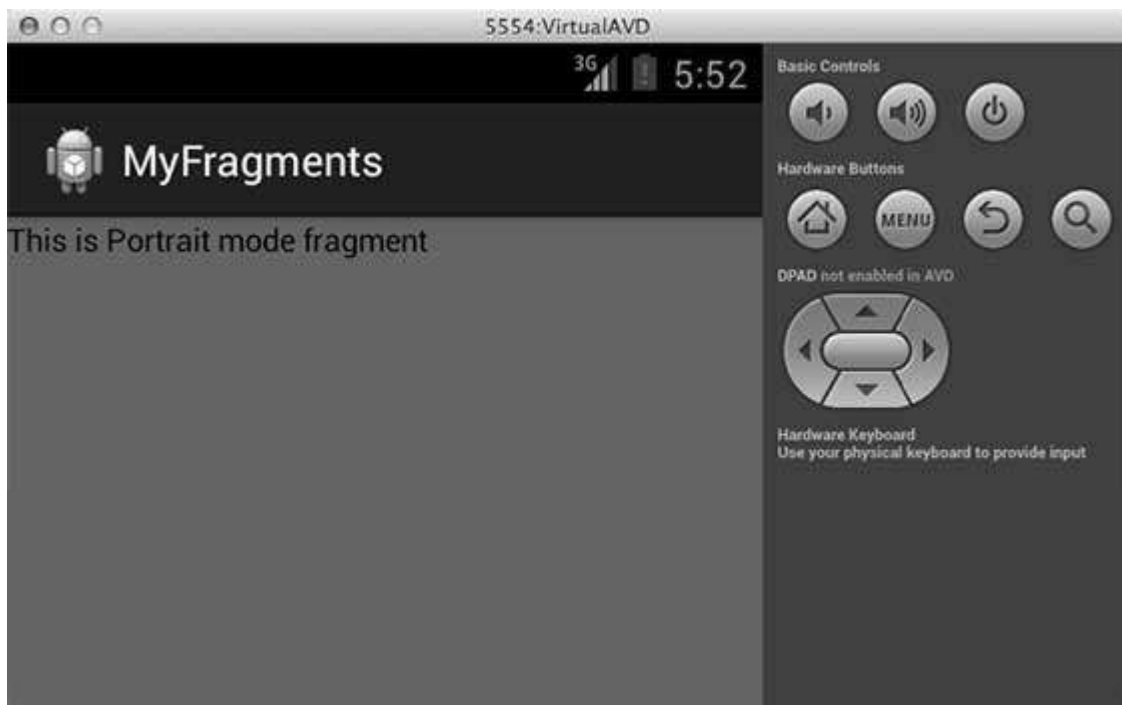
<string name="app_name">MyFragments</string>

<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="landscape_message">This is Landscape mode fragment
</string>
<string name="portrait_message">This is Portrait mode fragment
</string>

</resources>

```

Let's try to run our modified **MyFragments** 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 Emulator window where you will click on Menu button to see the following window. Be patience because it may take some time based on your computer speed:

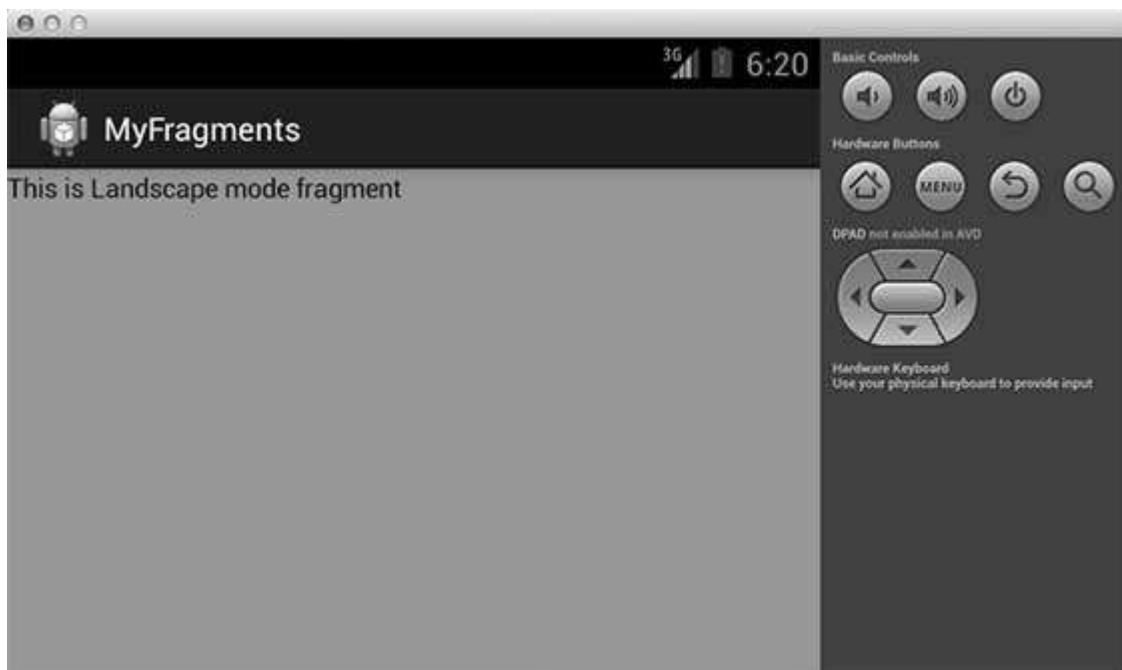


To change the mode of the emulator screen, let's do the following:

- **fn+control+F11** on Mac to change the landscape to portrait and vice versa.
- **ctrl+F11** on Windows.

- **ctrl+F11** on Linux.

Once you changed the mode, you will be able to see the GUI which you have implemented for landscape mode as below:



This way you can use same activity but different GUIs through different fragments. You can use different type of GUI components for different GUIs based on your requirements.

12. INTENTS & FILTERS

An Android **Intent** is an object carrying an *intent* i.e. message from one component to another component within the application or outside the application. The intents can communicate messages among any of the three core components of an application - activities, services, and broadcast receivers.

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed.

For example, let's assume, that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate **chooser**, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

For example, assume, that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION_WEB_SEARCH Intent to the Android Intent Resolver to open given URL in the web browser. The Intent Resolver parses through a list of Activities and chooses the one that would best match your Intent, in this case, the Web Browser Activity. The Intent Resolver then passes your web page to the web browser and starts the Web Browser Activity.

There are separate mechanisms for delivering intents to each type of component - activities, services, and broadcast receivers.

S.N.	Method & Description
1	Context.startActivity() The Intent object is passed to this method to launch a new activity or to get an existing activity to do something new.
2	Context.startService() The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.
3	Context.sendBroadcast() The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

Intent Objects

An Intent object is a bundle of information which is used by the component that receives the intent plus information used by the Android system.

An Intent object can contain the following components based on what it is communicating or going to perform:

Action

This is mandatory part of the Intent object and is a string naming the action to be performed or, in the case of broadcast intents, the action that took place and is being reported. The action largely determines how the rest of the intent object is structured. The Intent class defines a number of action constants corresponding to different intents. Here is a list of [Android Intent Standard Actions](#)

The action in an Intent object can be set by the `setAction()` method and read by `getAction()`.

Data

The URI of the data to be acted on and the MIME type of that data. For example, if the action field is `ACTION_EDIT`, the data field would contain the URI of the document to be displayed for editing.

The `setData()` method specifies data only as a URI, `setType()` specifies it only as a MIME type, and `setDataAndType()` specifies it as both a URI and a MIME type. The URI is read by `getData()` and the type by `getType()`.

Some examples of action/data pairs are:

S.N.	Action/Data Pair & Description
1	ACTION_VIEW content://contacts/people/1 Display information about the person whose identifier is "1".
2	ACTION_DIAL content://contacts/people/1 Display the phone dialer with the person filled in.
3	ACTION_VIEW tel:123 Display the phone dialer with the given number filled in.
4	ACTION_DIAL tel:123 Display the phone dialer with the given number filled in.

5	ACTION_EDIT content://contacts/people/1 Edit information about the person whose identifier is "1".
6	ACTION_VIEW content://contacts/people/ Display a list of people, which the user can browse through.

Category

The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent. The `addCategory()` method places a category in an Intent object, `removeCategory()` deletes a category previously added, and `getCategories()` gets the set of all categories currently in the object. Here is a list of [Android Intent Standard Categories](#).

You can check detail on Intent Filters in below section to understand how do we use categories to choose appropriate activity corresponding to an Intent.

Extras

This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the `putExtras()` and `getExtras()` methods respectively. Here is a list of [Android Intent Standard Extra Data](#)

Flags

These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it is launched etc.

Component Name

This optional field is an android **ComponentName** object representing either Activity, Service or BroadcastReceiver class. If it is set, the Intent object is delivered to an instance of the designated class, otherwise Android uses other information in the Intent object to locate a suitable target.

The component name is set by `setComponent()`, `setClass()`, or `setClassName()` and read by `getComponent()`.

Types of Intents

There are following two types of intents supported by Android till version 4.1

Explicit Intents

These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity. For example:

```
// Explicit Intent by specifying its class name
Intent i = new Intent(this, TargetActivity.class);
i.putExtra("Key1", "ABC");
i.putExtra("Key2", "123");

// Starts TargetActivity
startActivity(i);
```

Implicit Intents

These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example:

```
// Implicit Intent by specifying a URI
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.example.com"));

// Starts Implicit Activity
startActivity(i);
```

The target component which receives the intent can use the **getExtras()** method to get the extra data sent by the source component. For example:

```
// Get bundle object at appropriate place in your code
Bundle extras = getIntent().getExtras();

// Extract data using passed keys
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

Example:

Following example shows the functionality of an Android Intent to launch various Android built-in applications.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>IntentDemo</i> under a package <i>com.example.intentdemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add the code to define two listeners corresponding two buttons i.e. Start Browser and Start Phone.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> to add three buttons in linear layout.
4	Modify <i>res/values/strings.xml</i> to define required constant values.
5	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.intentdemo/MainActivity.java**.

```
package com.example.intentdemo;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

Button startBrowser = (Button) findViewById(R.id.start_browser);
startBrowser.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent i = new Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.example.com"));
        startActivity(i);
    }
});
Button startPhone = (Button) findViewById(R.id.start_phone);
startPhone.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent i = new Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("tel:9510300000"));
        startActivity(i);
    }
});
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action
    // bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"

```

```

    android:orientation="vertical" >

    <Button android:id="@+id/start_browser"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/start_browser"/>

    <Button android:id="@+id/start_phone"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/start_phone" />

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">IntentDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="start_browser">Start Browser</string>
    <string name="start_phone">Start Phone</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.intentdemo"
    android:versionCode="1"
    android:versionName="1.0" >


```

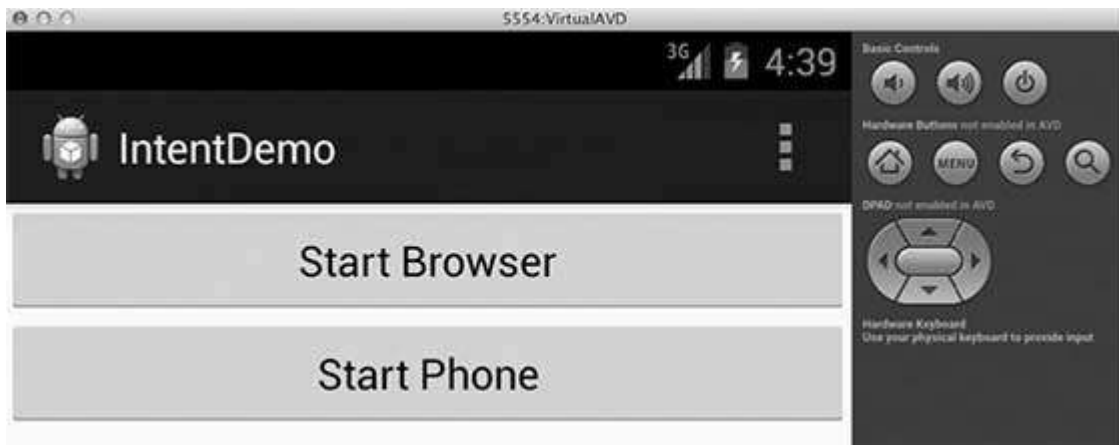
```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.intentdemo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
                />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Let's try to run your **IntentDemo** application. 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:



Now click on **Start Browser** button, which will start a browser configured and display <http://www.example.com> as shown below:



Similarly, you can launch phone interface using Start Phone button, which will allow you to dial already given phone number.

Intent Filters

You have seen how an Intent has been used to call another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.intentdemo.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data:

```
<activity android:name=".CustomActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="com.example.intentdemo.LAUNCH" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.intentdemo.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.

The **<data>** element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://".

There may be a situation that an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.

There are following test Android checks before invoking an activity:

- A filter **<intent-filter>** may list more than one action as shown above but this list cannot be empty; a filter must contain at least one **<action>** element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.
- A filter **<intent-filter>** may list zero, one or more than one categories. If there is no category mentioned then android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.
- Each **<data>** element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme**, **host**, **port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

Example:

Following example is a modification of the above example. Here we will see how Android resolves conflict if one intent is invoking two activities defined in, next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>IntentDemo</i> under a package <i>com.example.intentdemo</i> . While creating this project, make sure your <i>Target SDK</i> and <i>Compile With</i> are at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add the code to define three listeners corresponding to three buttons defined in layout file.
3	Add a new <i>src/CustomActivity.java</i> file to have one custom activity which will be invoked by different intents.
4	Modify layout XML file <i>res/layout/activity_main.xml</i> to add three buttons in linear layout.
5	Add one layout XML file <i>res/layout/custom_view.xml</i> to add a simple <code><TextView></code> to show the passed data through intent.
6	Modify <i>res/values/strings.xml</i> to define required constant values.
7	Modify <i>AndroidManifest.xml</i> to add <code><intent-filter></code> to define rules for your intent to invoke custom activity.
8	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.intentdemo/MainActivity.java**.

```
package com.example.intentdemo;

import android.net.Uri;
import android.os.Bundle;
```

```
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // First intent to use ACTION_VIEW action with correct data
        Button startBrowser_a = (Button)
        findViewById(R.id.start_browser_a);
        startBrowser_a.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Intent i = new Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("http://www.example.com"));
                startActivity(i);
            }
        });

        // Second intent to use LAUNCH action with correct data
        Button startBrowser_b = (Button)
        findViewById(R.id.start_browser_b);
        startBrowser_b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Intent i = new Intent("com.example.intentdemo.LAUNCH",
                Uri.parse("http://www.example.com"));
                startActivity(i);
            }
        });
    }
}
```



```

// Third intent to use LAUNCH action with incorrect data
Button startBrowser_c = (Button)
findViewById(R.id.start_browser_c);
startBrowser_c.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent i = new Intent("com.example.intentdemo.LAUNCH",
            Uri.parse("https://www.example.com"));
        startActivity(i);
    }
});

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the
    // action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

}

```

Following is the content of the modified main activity file **src/com.example.intentdemo/CustomActivity.java**.

```

package com.example.intentdemo;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class CustomActivity extends Activity {

```

```

@Override
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.custom_view);

    TextView label = (TextView) findViewById(R.id.show_data);

    Uri url = getIntent().getData();
    label.setText(url.toString());
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/start_browser_a"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start_browser_a"/>

    <Button android:id="@+id/start_browser_b"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start_browser_b"/>

    <Button android:id="@+id/start_browser_c"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start_browser_c"/>

```

```
</LinearLayout>
```

Following will be the content of **res/layout/custom_view.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TextView android:id="@+id/show_data"
        android:layout_width="fill_parent"
        android:layout_height="400dp"/>

</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants:

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

    <string name="app_name">IntentDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="start_browser_a">Start Browser with VIEW
action</string>
    <string name="start_browser_b">Start Browser with LAUNCH
action</string>
    <string name="start_browser_c">Exception Condition</string>

</resources>
```

Following is the default content of **AndroidManifest.xml**:


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.intentdemo"
```

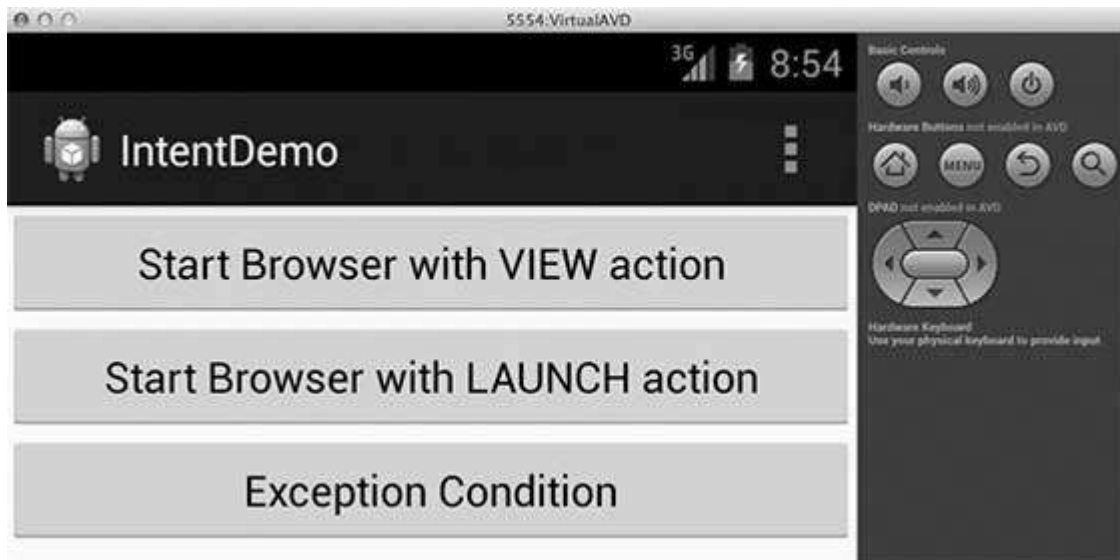
```
android:versionCode="1"
android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.intentdemo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

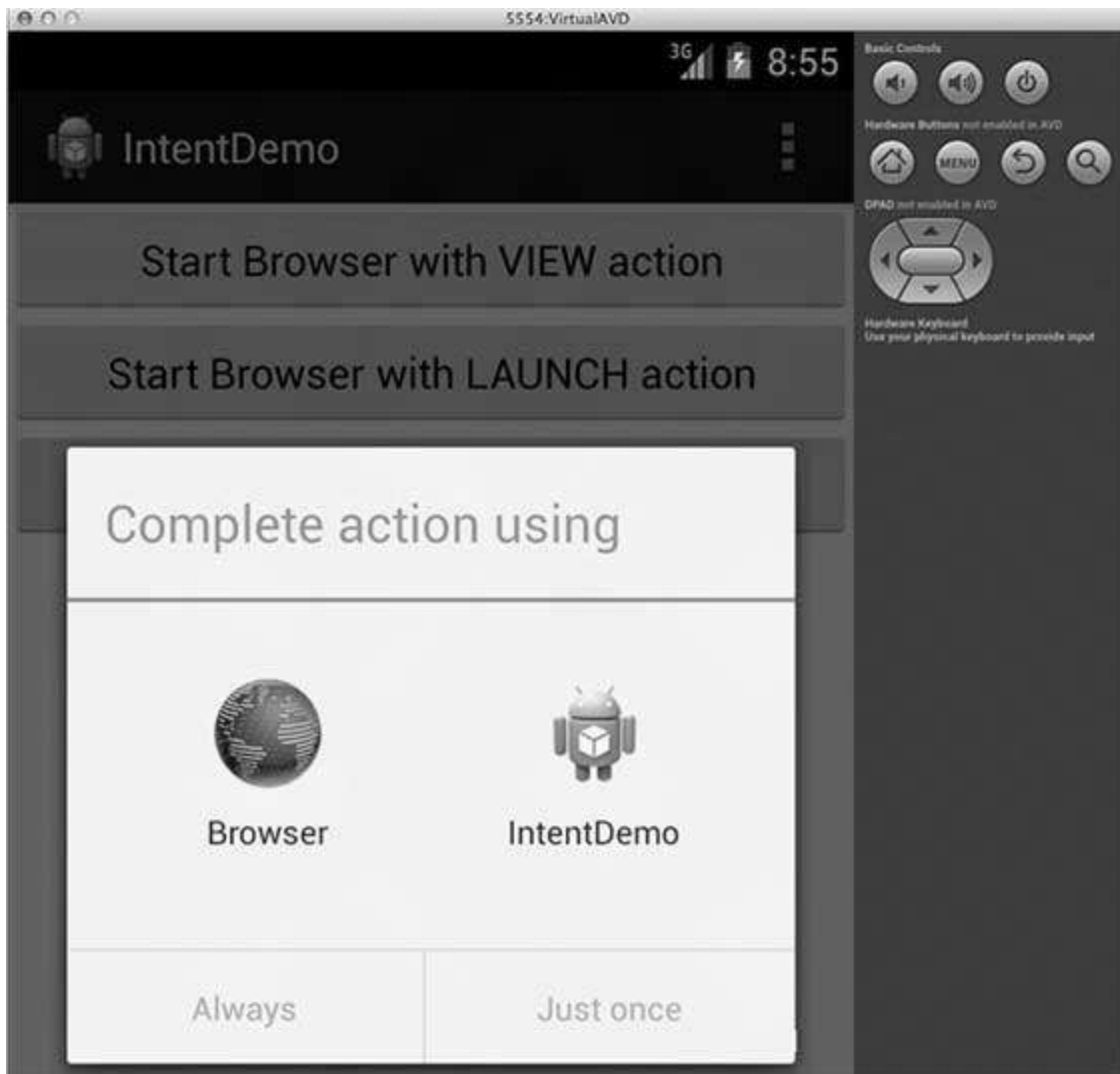
            <category android:name="android.intent.category.LAUNCHER"
                />
        </intent-filter>
    </activity>
    <activity android:name="com.example.intentdemo.CustomActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <action android:name="com.example.intentdemo.LAUNCH" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:scheme="http" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Let's try to run your **IntentDemo** application. 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:



Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser. So android displays following two options to select the activity you want to launch.



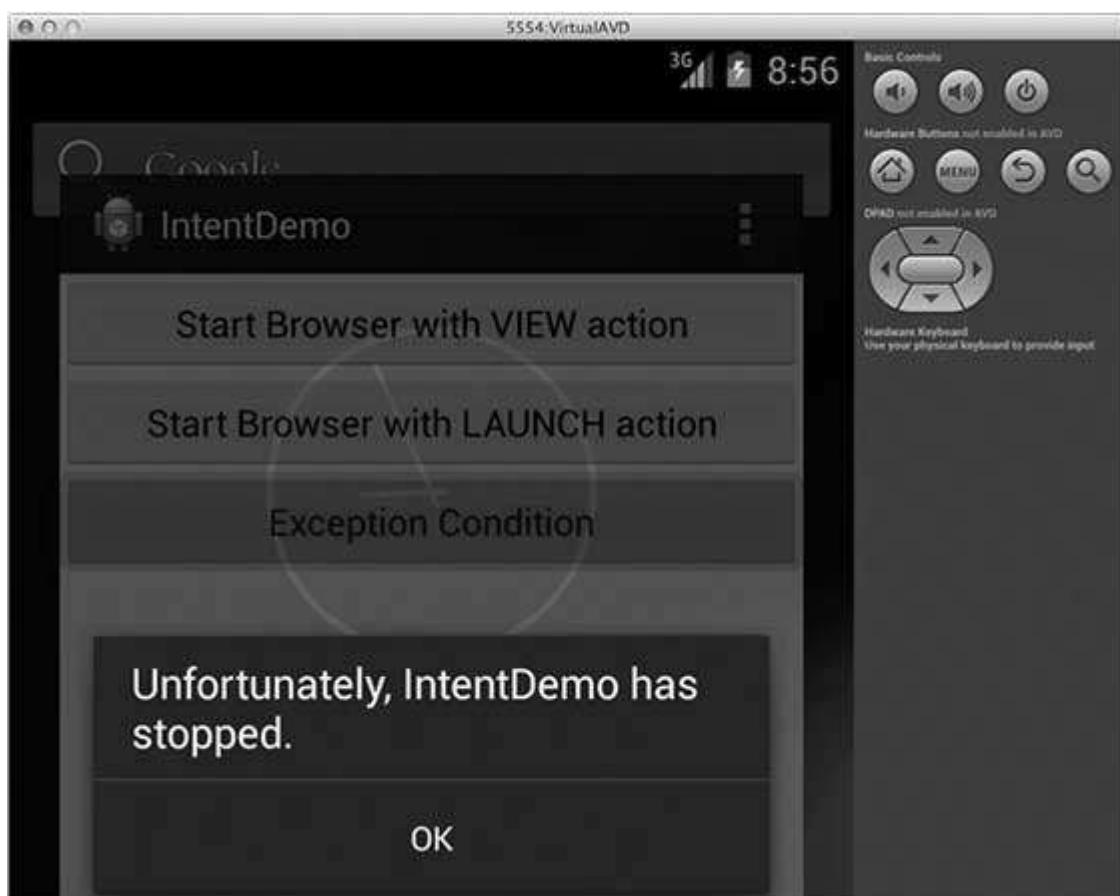
Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows:



Now go back using back button and click on "Start Browser with LAUNCH Action" button, here Android applies filter to choose define activity and it simply launch your custom activity and again it displays following screen:



Again, go back using back button and click on "Exception Condition" button, here Android tries to find out a valid filter for the given intent but it does not find a valid activity defined because this time we have used data as **https** instead of **http** though we are giving a correct action, so Android raises an exception and shows following screen:



13. UI LAYOUTS

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.

This tutorial is more about creating your GUI based on layouts defined in XML file. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout:

```
<?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="This is a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a Button" />

    <!-- More GUI components go here -->

</LinearLayout>
```


Once your layout is defined, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below:

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

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

S.N.	Layout & Description
1	<u>Linear Layout</u> LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	<u>Relative Layout</u> RelativeLayout is a view group that displays child views in relative positions.
3	<u>Table Layout</u> TableLayout is a view that groups views into rows and columns.
4	<u>Absolute Layout</u> AbsoluteLayout enables you to specify the exact location of its children.
5	<u>Frame Layout</u> The FrameLayout is a placeholder on screen that you can use to display a single view.
6	<u>List View</u> ListView is a view group that displays a list of scrollable items.
7	<u>Grid View</u> GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Attribute	Description
android:id	This is the ID which uniquely identifies the view.
android:layout_width	This is the width of the layout.
android:layout_height	This is the height of the layout
android:layout_marginTop	This is the extra space on the top side of the layout.
android:layout_marginBottom	This is the extra space on the bottom side of the layout.
android:layout_marginLeft	This is the extra space on the left side of the layout.
android:layout_marginRight	This is the extra space on the right side of the layout.
android:layout_gravity	This specifies how child Views are positioned.
android:layout_weight	This specifies how much of the extra space in the layout should be allocated to the View.
android:layout_x	This specifies the x-coordinate of the layout.
android:layout_y	This specifies the y-coordinate of the layout.
android:layout_width	This is the width of the layout.
android:layout_width	This is the width of the layout.

android:paddingLeft	This is the left padding filled for the layout.
android:paddingRight	This is the right padding filled for the layout.
android:paddingTop	This is the top padding filled for the layout.
android:paddingBottom	This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px (Pixels), mm (Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height:

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

Constant	Value	Description
top	0x30	Push object to the top of its container, not changing its size.
bottom	0x50	Push object to the bottom of its container, not changing its size.
left	0x03	Push object to the left of its container, not changing its size.
right	0x05	Push object to the right of its container, not changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.

center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.
clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.
clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
start	0x00800003	Push object to the beginning of its container, not changing its size.
end	0x00800005	Push object to the end of its container, not changing its size.

View Identification

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);
```

14. UI CONTROLS

An Android application user interface is everything that the user can see and interact with. You have learned about the various layouts that you can use to position your views in an activity. This chapter will give you detail on various views.

A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML. For example, a simple vertical layout with a text view and a button looks like this:

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

Android UI Controls

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

S.N.	UI Control & Description
------	--------------------------

1	<u>TextView</u> This control is used to display text to the user.
2	<u>EditText</u> EditText is a pre-defined subclass of TextView that includes rich editing capabilities.
3	<u>AutoCompleteTextView</u> 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	<u>Button</u> A push-button that can be pressed, or clicked, by the user to perform an action.
5	<u>ImageButton</u> AbsoluteLayout enables you to specify the exact location of its children.
6	<u>CheckBox</u> An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.
7	<u>ToggleButton</u> An on/off button with a light indicator.
8	<u>RadioButton</u> The RadioButton has two states: either checked or unchecked.
9	<u>RadioGroup</u> A RadioGroup is used to group together one or more RadioButtons.
10	<u>ProgressBar</u> The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.

11	<u>Spinner</u> A drop-down list that allows users to select one value from a set.
12	<u>TimePicker</u> The TimePicker view enable users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	<u>DatePicker</u> The DatePicker view enable users to select a date of the day.

Create UI Controls

As explained in previous chapter, 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/text_id"
```

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows:

```
<?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_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following:

```
TextView myText = (TextView) findViewById(R.id.text_id);
```


15. EVENT HANDLING

Events are a useful way to collect data about a user's interaction with interactive components of your app, like button presses or screen touch etc. The Android framework maintains an event queue into which events are placed as they occur and then each event is removed from the queue on a first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are following three concepts related to Android Event Management:

- **Event Listeners:** The **View** class is mainly involved in building up an Android GUI, same View class provides a number of Event Listeners. The Event Listener is the object that receives notification when an event occurs.
- **Event Listeners Registration:** Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers:** When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget loses its focus i.e. user goes away from the view item. You will use onFocusChange() event handler to handle such event.

onKey()	<p>OnFocusChangeListener()</p> <p>This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.</p>
onTouch()	<p>OnTouchListener()</p> <p>This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.</p>
onMenuItemClick()	<p>OnMenuItemClickListener()</p> <p>This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.</p>

There are many more event listeners available as a part of **View** class like OnHoverListener, OnDragListener etc., which may be needed for your application. So we recommend to refer official documentation for Android application development in case you are going to develop a sophisticated app.

Event Listeners Registration:

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event. Though there are several tricky ways to register your event listener for any event, let us list down only top 3 ways, out of which you can use any of them based on the situation.

- Using an Anonymous Inner Class
- Activity class implements the Listener interface.
- Using Layout file activity_main.xml to specify event handler directly.

Below section will provide you detailed examples on all the three scenarios:

Event Handling Examples

Event Listeners Registration Using an Anonymous Inner Class

Here you will create an anonymous implementation of the listener and will be useful if each class is applied to a single control only and you have advantage to pass arguments to event handler. In this approach event handler methods can access private data of Activity. No reference is needed to call to Activity.

But if you have applied the handler to more than one control, you would have to cut and paste the code for the handler and if the code for the handler is long, it makes the code harder to maintain.

Following are the simple steps to show how we will make use of separate Listener class to register and capture click event. Similarly, you can implement your listener for any other required event type.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>EventDemo</i> under a package <i>com.example.eventdemo</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify <i>src/MainActivity.java</i> file to add click event listeners and handlers for the two buttons defined.
3	Modify the default content of <i>res/layout/activity_main.xml</i> file to include Android UI controls.
4	Define required constants in <i>res/values/strings.xml</i> file.
5	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.eventdemo/MainActivity.java**. This file can include each of the fundamental life-cycle methods.

```
package com.example.eventdemo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //--- find both the buttons---
    Button sButton = (Button) findViewById(R.id.button_s);
    Button lButton = (Button) findViewById(R.id.button_l);

    // -- register click event with first button ---
    sButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            // --- find the text view --
            TextView txtView = (TextView) findViewById(R.id.text_id);
            // -- change text size --
            txtView.setTextSize(14);
        }
    });

    // -- register click event with second button ---
    lButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            // --- find the text view --
            TextView txtView = (TextView) findViewById(R.id.text_id);
            // -- change text size --
            txtView.setTextSize(24);
        }
    });
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

```

```

    }
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<?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" >

    <Button
        android:id="@+id/button_s"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/button_small"/>

    <Button
        android:id="@+id/button_l"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/button_large"/>

    <TextView
        android:id="@+id/text_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:capitalize="characters"
        android:text="@string/hello_world" />

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">EventDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
<string name="button_small">Small Font</string>
<string name="button_large">Large Font</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.guidemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.guidemo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"


```

```

        />
    </intent-filter>
</activity>
</application>

</manifest>

```

Let's try to run your **EventDemo** application. 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:



Now you try to click on two buttons one by one and you will see that font of the **Hello World** text will change, which happens because registered click event handler method is being called against each click event.

Registration Using the Activity Implements Listener Interface

Here your Activity class implements the Listener interface and you put the handler method in the main Activity and then you call `setOnClickListener(this)`.

This approach is fine if your application has only a single control of that Listener type otherwise you will have to do further programming to check which control has generated event. Secondly, you cannot pass arguments to the Listener so, again, works poorly for multiple controls.

Following are the simple steps to show how we will implement Listener class to register and capture click event. Similarly, you can implement your listener for any other required event type.

Step	Description
1	We do not need to create this application from scratch, so let's make use of above created Android application <i>EventDemo</i> .
2	Modify <i>src/MainActivity.java</i> file to add click event listeners and handlers for the two buttons defined.
3	We are not making any change in <i>res/layout/activity_main.xml</i> , it will remain as shown above.
4	We are also not making any change in <i>res/values/strings.xml</i> file, it will also remain as shown above.
5	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.eventdemo/MainActivity.java**. This file can include each of the fundamental life-cycle methods.

```
package com.example.eventdemo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```



```
//--- find both the buttons---
Button sButton = (Button) findViewById(R.id.button_s);
Button lButton = (Button) findViewById(R.id.button_l);

// -- register click event with first button ---
sButton.setOnClickListener(this);
// -- register click event with second button ---
lButton.setOnClickListener(this);
}


//--- Implement the OnClickListener callback
public void onClick(View v) {
    if(v.getId() == R.id.button_s)
    {
        // --- find the text view --
        TextView txtView = (TextView) findViewById(R.id.text_id);
        // -- change text size --
        txtView.setTextSize(14);
        return;
    }
    if(v.getId() == R.id.button_l)
    {
        // --- find the text view --
        TextView txtView = (TextView) findViewById(R.id.text_id);
        // -- change text size --
        txtView.setTextSize(24);
        return;
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
```

```

        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Now again let's try to run your **EventDemo** application. 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:



Now you try to click on two buttons one by one and you will see that font of the **Hello World** text will change, which happens because registered click event handler method is being called against each click event.

Registration Using Layout file activity_main.xml

Here you put your event handlers in Activity class without implementing a Listener interface or call to any listener method. Rather you will use the layout file (activity_main.xml) to specify the handler method via the **android:onClick** attribute for click event. You can control click events differently for different control by passing different event handler methods.

The event handler method must have a void return type and take a View as an argument. However, the method name is arbitrary, and the main class need not implement any particular interface.

This approach does not allow you to pass arguments to Listener and for the Android developers it will be difficult to know which method is the handler for which control, until they look into activity_main.xml file. Secondly, you cannot handle any other event except click event using this approach.

Following are the simple steps to show how we can make use of layout file Main.xml to register and capture click event.

Step	Description
1	We do not need to create this application from scratch, so let's make use of above created Android application <i>EventDemo</i> .
2	Modify <i>src/MainActivity.java</i> file to add click event listeners and handlers for the two buttons defined.
3	Modify layout file <i>res/layout/activity_main.xml</i> , to specify event handlers for the two buttons.
4	We are also not making any change in <i>res/values/strings.xml</i> file, it will also remain as shown above.
5	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.eventdemo/MainActivity.java**. This file can include each of the fundamental life-cycle methods.

```

package com.example.eventdemo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}

```

```

        setContentView(R.layout.activity_main);
    }

    //--- Implement the event handler for the first button.
    public void doSmall(View v) {
        // --- find the text view --
        TextView txtView = (TextView) findViewById(R.id.text_id);
        // -- change text size --
        txtView.setTextSize(14);
        return;
    }

    //--- Implement the event handler for the second button.
    public void doLarge(View v) {
        // --- find the text view --
        TextView txtView = (TextView) findViewById(R.id.text_id);
        // -- change text size --
        txtView.setTextSize(24);
        return;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
}

```

Following will be the content of **res/layout/activity_main.xml** file. Here we have to add **android:onClick="methodName"** for both the buttons, which will register given method names as click event handlers.

```

<?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" >

<Button
    android:id="@+id/button_s"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="@string/button_small"
    android:onClick="doSmall"/>

<Button
    android:id="@+id/button_l"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="@string/button_large"
    android:onClick="doLarge"/>

<TextView
    android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:capitalize="characters"
    android:text="@string/hello_world" />

</LinearLayout>
```

Again let's try to run your **EventDemo** application. 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:



Now you try to click on two buttons one by one and you will see that font of the **Hello World** text will change, which happens because registered click event handler method is being called against each click event.

Exercise:

We recommend you to try writing different event handlers for different event types and understand exact difference in different event types and their handling. Events related to menu, spinner, pickers widgets are little different but they are also based on the same concepts as explained above.

17. STYLES & THEMES

If you already know about Cascading Style Sheet (CSS) in web design then understanding Android Style will be easy as it also works in a similar way. There are number of attributes associated with each Android widget which you can set to change your application's look and feel. A style can specify properties such as height, padding, font color, font size, background color, and much more.

You can specify these attributes in Layout file as follows:

```
<?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_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:capitalize="characters"
        android:textColor="#00FF00"
        android:typeface="monospace"
        android:text="@string/hello_world" />

</LinearLayout>
```

But this way we need to define style attributes for every attribute separately which is not good for source code maintenance point of view. So we work with styles by defining them in separate file as explained below.

Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under **res/values/** directory of your project and will have **<resources>** as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.

You can define multiple styles per file using **<style>** tag but each style will have its name that uniquely identifies the style. Android style attributes are set using **<item>** tag as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:capitalize">characters</item>
    <item name="android:typeface">monospace</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textColor">#00FF00</item>/>
  </style>
</resources>
```

The value for the **<item>** can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property.

Using Styles

Once your style is defined, you can use it in your XML Layout file using **style** attribute as follows:

```
<?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_id"
  style="@style/CustomFontStyle"
  android:text="@string/hello_world" />

</LinearLayout>
```

To understand the concept related to Android Style, you can check [Style Demo Example](#).

Style Inheritance

Android supports Style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add.

It is simple to create a new style **LargeFont** that inherits the **CustomFontStyle** style defined above, but make the font size big, you can author the new style like this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle.LargeFont">
    <item name="android:textSize">20ps</item>
  </style>
</resources>
```

You can reference this new style as **@style/CustomFontStyle.LargeFont** in your XML Layout file. You can continue inheriting like this as many times as you'd like, by chaining names with periods. For example, you can extend **FontStyle.LargeFont** to be Red, with:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle.LargeFont.Red">
    <item name="android:textColor">#FF0000</item>/>
  </style>
</resources>
```

This technique for inheritance by chaining together names only works for styles defined by your own resources. You can't inherit Android built-in styles this way. To reference an Android built-in style, such as **TextAppearance**, you must use the **parent** attribute as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle" parent="@android:style/TextAppearance">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:capitalize">characters</item>
    <item name="android:typeface">monospace</item>
  </style>
</resources>
```

```

    <item name="android:textSize">12pt</item>
    <item name="android:textColor">#00FF00</item>/>
</style>
</resources>

```

Android Themes

Hope you understood the concept of Style, so now let's try to understand what is a **Theme**. A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.

Thus, when a style is applied as a theme, every **View** in the Activity or application will apply each style property that it supports. For example, you can apply the same **CustomFontStyle** style as a theme for an Activity and then all text inside that **Activity** will have green monospace font.

To set a theme for all the activities of your application, open the **AndroidManifest.xml** file and edit the **<application>** tag to include the **android:theme** attribute with the style name. For example:

```
<application android:theme="@style/CustomFontStyle">
```

But if you want a theme applied to just one Activity in your application, then add the **android:theme** attribute to the **<activity>** tag only. For example:

```
<activity android:theme="@style/CustomFontStyle">
```

There are number of default themes defined by Android which you can use directly or inherit them using **parent** attribute as follows:

```

<style name="CustomTheme" parent="android:Theme.Light">
    ...
</style>

```

To understand the concept related to Android Theme, you can check [Theme Demo Example](#).

Default Styles & Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the **R.style** class. To use the styles listed here, replace all underscores in the style name with a period. For example, you can apply `theme_NoTitleBar` theme with `"@android:style/Theme.NoTitleBar"`. You can see the following source code for Android styles and themes:

- Android Styles (styles.xml)
- Android Themes (themes.xml)

18. CUSTOM COMPONENTS

Android offers a great list of pre-built widgets like Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, AutoCompleteTextView etc. which you can use directly in your Android application development, but there may be a situation when you are not satisfied with existing functionality of any of the available widgets. Android provides you with means of creating your own custom components which you can customize to suit your needs.

If you only need to make small adjustments to an existing widget or layout, you can simply subclass the widget or layout and override its methods which will give you precise control over the appearance and function of a screen element.

This tutorial explains you how to create custom Views and use them in your application using simple and easy steps.

Creating a Simple Custom Component

The simplest way to create your custom component is to extend an existing widget class or subclass with your own class if you want to extend the functionality of existing widget like Button, TextView, EditText, ListView, CheckBox etc. otherwise you can do everything yourself by starting with the *android.view.View* class.

At its simplest form you will have to write your constructors corresponding to all the constructors of the base class. For example if you are going to extend **TextView** to create a **DateView** then following three constructors will be created for DateView class:

```
public class DateView extends TextView {
    public DateView(Context context) {
        super(context);
        //--- Additional custom code --
    }

    public DateView(Context context, AttributeSet attrs) {
        super(context, attrs);
        //--- Additional custom code --
    }

    public DateView(Context context, AttributeSet attrs, int defStyle) {
```

```

        super(context, attrs, defStyle);
        //--- Additional custom code ---
    }
}

```

Because you have created `DateView` as child of `TextView` so it will have access on all the attributes, methods and events related to `TextView` and you will be able to use them without any further implementation. You will implement additional custom functionality inside your own code as explained in the given examples below.

If you have requirement for implementing custom drawing/sizing for your custom widgets then you need to override **`onMeasure(int widthMeasureSpec, int heightMeasureSpec)`** and **`onDraw(Canvas canvas)`** methods. If you are not going to resize or change the shape of your built-in component then you do not need either of these methods in your custom component.

The `onMeasure()` method coordinate with the layout manager to report the widget's width and height, and you need to call `setMeasuredDimension(int width, int height)` from inside this method to report the dimensions.

You can then execute your custom drawing inside the `onDraw(Canvas canvas)` method, where `android.graphics.Canvas` is pretty similar to its counterpart in Swing, and has methods such as `drawRect()`, `drawLine()`, `drawString()`, `drawBitmap()` etc. which you can use to draw your component.

Once you are done with the implementation of a custom component by extending existing widget, you will be able to instantiate these custom components in two ways in your application development:

Instantiate using code inside activity class

It is very similar way of instantiating custom component the way you instantiate built-in widget in your activity class. For example you can use following code to instantiate above defined custom component:

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    DateView dateView = new DateView(this);
    setContentView(dateView);
}

```

```
}

```

Check this example to understand how to [Instantiate a Basic Android Custom Component](#) using code inside an activity.

Instantiate using Layout XML file

Traditionally you use Layout XML file to instantiate your built-in widgets, same concept will apply on your custom widgets as well so you will be able to instantiate your custom component using Layout XML file as explained below. Here **com.example.dateviewdemo** is the package where you have put all the code related to **DateView** class and **DateView** is Java class name where you have put complete logic of your custom component.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <com.example.dateviewdemo.DateView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#fff"
        android:textSize="40sp"
        android:background="#000"
    />

</RelativeLayout>
```

It is important to note here that we are using all TextView attributes along with custom component without any change. Similarly, you will be able to use all the events, and methods along with DateView component.

Check this example to understand how to [Instantiate a Basic Android Custom Component](#) using Layout XML file.

Custom Component with Custom Attributes

We have seen how we can extend functionality of built-in widgets but in both the examples given above we saw that extended component can make use of all the default attributes of its parent class. But consider a situation when you want to create your own attribute from scratch. Below is a simple procedure to create and use new attributes for Android Custom components. Consider we want to introduce three attributes and will use them as shown below:

```
<com.example.dateviewdemo.DateView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#fff"
    android:textSize="40sp"
    custom:delimiter="-"
    custom:fancyText="true"
/>
```

Step 1

The first step to enable us to use our custom attributes is to define them in a new xml file under *res/values/* and call it **attrs.xml**. Let's have a look on an example attrs.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="DateView">
        <attr name="delimiter" format="string"/>
        <attr name="fancyText" format="boolean"/>
    </declare-styleable>
</resources>
```

Here the **name=value** is what we want to use in our Layout XML file as attribute, and the **format=type** is the type of attribute.

Step 2

Your second step will be to read these attributes from Layout XML file and set them for the component. This logic will go in the constructors that get passed an *AttributeSet*, since that is what contains the XML attributes. To read the values in the XML, you need to first create a *TypedArray* from the *AttributeSet*, then use that to read and set the values as shown in the below example code:

```

TypedArray a = context.obtainStyledAttributes(attrs,
R.styleable.DateView);

final int N = a.getIndexCount();
for (int i = 0; i < N; ++i)
{
    int attr = a.getIndex(i);
    switch (attr)
    {
        case R.styleable.DateView_delimiter:
            String delimiter = a.getString(attr);
            //...do something with delimiter...
            break;
        case R.styleable.DateView_fancyText:
            boolean fancyText = a.getBoolean(attr, false);
            //...do something with fancyText...
            break;
    }
}
a.recycle();

```

Step 3

Finally you can use your defined attributes in your Layout XML file as follows:

```

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

xmlns:custom="http://schemas.android.com/apk/res/com.example.dateviewdemo"
"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"

```



```
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".MainActivity" >

        <com.example.dateviewdemo.DateView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textColor="#fff"
            android:textSize="40sp"
            custom:delimiter="-"
            custom:fancyText="true"
        />

    </RelativeLayout>
```

The important part is `xmlns:custom="http://schemas.android.com/apk/res/com.example.dateviewdemo"`. Note that `http://schemas.android.com/apk/res/` will remain as is, but last part will be set to your package name and also that you can use anything after `xmlns:`, in this example we have used **custom**, but you could use any name you like.

Check this example to understand how to [Create Custom Attributes for Android Custom Component](#) with simple steps.

19. DRAG & DROP

Android drag/drop framework allows your users to move data from one View to another View in the current layout using a graphical drag and drop gesture. The framework includes following three important components to support drag & drop functionality:

- **Drag event class:**
- **Drag listeners:**
- **Helper methods and classes:**

The Drag/Drop Process

There are basically four steps or states in the drag and drop process:

- **Started:** This event occurs when you start dragging an item in a layout, your application calls *startDrag()* method to tell the system to start a drag. The arguments inside *startDrag()* method provide the data to be dragged, metadata for this data, and a callback for drawing the drag shadow.

The system first responds by calling back to your application to get a drag shadow. It then displays the drag shadow on the device.

Next, the system sends a drag event with action *ACTION_DRAG_STARTED* to the registered drag event listeners for all the View objects in the current layout.

To continue to receive drag events, including a possible drop event, a drag event listener must return **true**, if the drag event listener returns false, then it will not receive drag events for the current operation until the system sends a drag event with action type *ACTION_DRAG_ENDED*.

- **Continuing:** The user continues the drag. System sends *ACTION_DRAG_ENTERED* action followed by *ACTION_DRAG_LOCATION* action to the registered drag event listener for the View where dragging point enters. The listener may choose to alter its View object's appearance in response to the event or can react by highlighting its View.

The drag event listener receives an *ACTION_DRAG_EXITED* action after the user has moved the drag shadow outside the bounding box of the View.

- **Dropped:** The user releases the dragged item within the bounding box of a View. The system sends the View object's listener, a drag event with action type *ACTION_DROP*.

- **Ended:** Just after the action type ACTION_DROP, the system sends out a drag event with action type ACTION_DRAG_ENDED to indicate that the drag operation is over.

The DragEvent Class

The **DragEvent** represents an event that is sent out by the system at various times during a drag and drop operation. This class provides few Constants and important methods which we use during Drag/Drop process.

Constants

Following are all constants integers available as a part of DragEvent class.

S.N.	Constants & Description
1	ACTION_DRAG_STARTED Signals the start of a drag and drop operation.
2	ACTION_DRAG_ENTERED Signals to a View that the drag point has entered the bounding box of the View.
3	ACTION_DRAG_LOCATION Sent to a View after ACTION_DRAG_ENTERED if the drag shadow is still within the View object's bounding box.
4	ACTION_DRAG_EXITED Signals that the user has moved the drag shadow outside the bounding box of the View.
5	ACTION_DROP Signals to a View that the user has released the drag shadow, and the drag point is within the bounding box of the View.
6	ACTION_DRAG_ENDED Signals to a View that the drag and drop operation has concluded.

Methods

Following are few important and most frequently used methods available as a part of DragEvent class.

S.N.	Constants & Description
1	int getAction() Inspect the action value of this event.
2	ClipData getClipData() Returns the ClipData object sent to the system as part of the call to startDrag().
3	ClipDescription getClipDescription() Returns the ClipDescription object contained in the ClipData.
4	boolean getResult() Returns an indication of the result of the drag and drop operation.
5	float getX() Gets the X coordinate of the drag point.
6	float getY() Gets the Y coordinate of the drag point.
7	String toString() Returns a string representation of this DragEvent object.

Listening for Drag Event

If you want any of your views within a Layout to respond to Drag event then your view either implements **View.OnDragListener** or setup **onDragEvent(DragEvent)** callback method. When the system calls the method or listener, it passes to them a DragEvent object explained above. You can have both a listener and a callback method for View object. If this occurs, the system first calls the listener and then defined callback as long as listener returns true.

The combination of the `onDragEvent(DragEvent)` method and `View.OnDragListener` is analogous to the combination of the `onTouchEvent()` and `View.OnTouchListener` used with touch events in old versions of Android.

Starting a Drag Event

You start with creating a **ClipData** and **ClipData.Item** for the data being moved. As part of the `ClipData` object, supply metadata that is stored in a **ClipDescription** object within the `ClipData`. For a drag and drop operation that does not represent data movement, you may want to use **null** instead of an actual object.

Next either you can extend **View.DragShadowBuilder** to create a drag shadow for dragging the view or simply you can use `View.DragShadowBuilder(View)` to create a default drag shadow that is the same size as the `View` argument passed to it, with the touch point centered in the drag shadow.

Example:

Following example shows the functionality of a simple Drag & Drop using a **View.setOnLongClickListener()** event listener along with **View.OnDragEventListener()**.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>DragNDropDemo</i> under a package <i>com.example.dragndropdemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add the code to define event listeners as well as a call back method for the logo image used in the example.
3	Copy image <i>logo.png</i> in <i>res/drawable-*</i> folder. You can use images with different resolution in case you want to provide them for different devices.
4	Modify layout XML file <i>res/layout/activity_main.xml</i> to define default view of the logo images.
5	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.dragndropdemo/MainActivity.java**. This file can include each of the fundamental life-cycle methods.

```
package com.example.dragndropdemo;

import android.os.Bundle;
import android.app.Activity;
import android.content.ClipData;
import android.content.ClipDescription;
import android.util.Log;
import android.view.DragEvent;
import android.view.View;
import android.view.View.DragShadowBuilder;
import android.view.View.OnDragListener;
import android.widget.*;

public class MainActivity extends Activity{
    ImageView ima;
    private static final String IMAGEVIEW_TAG = "Android Logo";
    String msg;

    private android.widget.RelativeLayout.LayoutParams layoutParams;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ima = (ImageView)findViewById(R.id.iv_logo);
        // Sets the tag
        ima.setTag(IMAGEVIEW_TAG);

        ima.setOnLongClickListener(new View.OnLongClickListener() {
```

```

@Override
public boolean onLongClick(View v) {
    ClipData.Item item = new
    ClipData.Item((CharSequence)v.getTag());

    String[] mimeTypes = {ClipDescription.MIMETYPE_TEXT_PLAIN};
    ClipData dragData = new ClipData(v.getTag().toString(),
    mimeTypes, item);

    // Instantiates the drag shadow builder.
    View.DragShadowBuilder myShadow = new DragShadowBuilder(ima);

    // Starts the drag
    v.startDrag(dragData, // the data to be dragged
    myShadow, // the drag shadow builder
    null, // no need to use local data
    0 // flags (not currently used, set to 0)
    );
    return true;
}
});

// Create and set the drag event listener for the View
ima.setOnDragListener( new OnDragListener(){
    @Override
    public boolean onDrag(View v, DragEvent event){
        switch(event.getAction())
        {
            case DragEvent.ACTION_DRAG_STARTED:
                layoutParams = (RelativeLayout.LayoutParams)
                v.getLayoutParams();
                Log.d(msg, "Action is DragEvent.ACTION_DRAG_STARTED");
                // Do nothing
                break;

```

```

        case DragEvent.ACTION_DRAG_ENTERED:
            Log.d(msg, "Action is DragEvent.ACTION_DRAG_ENTERED");
            int x_cord = (int) event.getX();
            int y_cord = (int) event.getY();
            break;
        case DragEvent.ACTION_DRAG_EXITED :
            Log.d(msg, "Action is DragEvent.ACTION_DRAG_EXITED");
            x_cord = (int) event.getX();
            y_cord = (int) event.getY();
            layoutParams.leftMargin = x_cord;
            layoutParams.topMargin = y_cord;
            v.setLayoutParams(layoutParams);
            break;
        case DragEvent.ACTION_DRAG_LOCATION :
            Log.d(msg, "Action is DragEvent.ACTION_DRAG_LOCATION");
            x_cord = (int) event.getX();
            y_cord = (int) event.getY();
            break;
        case DragEvent.ACTION_DRAG_ENDED :
            Log.d(msg, "Action is DragEvent.ACTION_DRAG_ENDED");
            // Do nothing
            break;
        case DragEvent.ACTION_DROP:
            Log.d(msg, "ACTION_DROP event");
            // Do nothing
            break;
        default: break;
    }
    return true;
}
});
}
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:


```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/iv_logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/logo"
        android:contentDescription="@string/drag_drop" />

</RelativeLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">DragNDropDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="drag_drop">Click on the image to drag and drop</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.guidemo"
    android:versionCode="1"
    android:versionName="1.0" >


```

```
<uses-sdk
    android:minSdkVersion="16"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.guidemo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
                />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Let's try to run your **DragNDropDemo** application. 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:



Now do long click on the displayed android logo and you will see that logo image moves a little after 1 seconds long click from its place, it is time when you should start dragging the image. You can drag it around the screen and drop it at a new location.



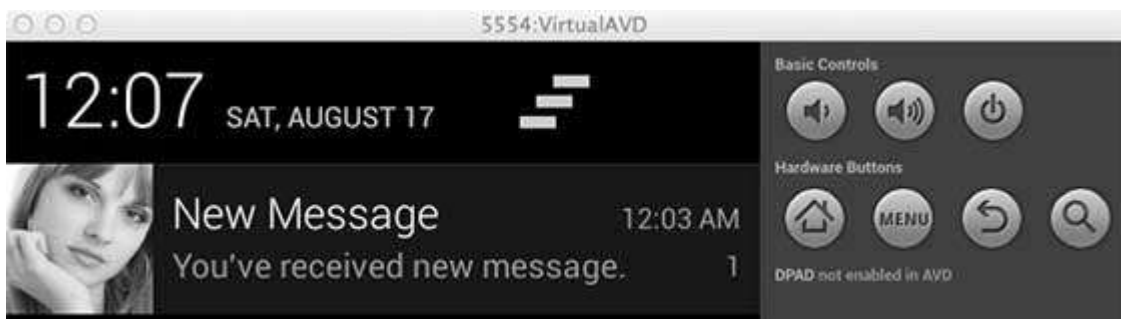
20. NOTIFICATIONS

Android **Toast** class provides a handy way to show alerts to the users, but these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

For important messages to be given to the user, it is required to have more persistent method. A **notification** is a message you can display as an icon at the top of the device which we call notification bar or status bar.



To see the details of the notification, you will have to select the icon which will display notification drawer having detail about the notification. While working with emulator with virtual device, you will have to click and drag down the status bar to expand it which will give you detail as follows. This will be just **64 dp** tall and called normal view.



Above expanded form can have a **Big View** which will have additional detail about the notification. You can add up to six additional lines in the notification. The following screenshot shows such notification.



Create and Send Notifications

There is a simple way to create a notification. Follow the below mentioned steps in your application to create a notification:

Step 1 - Create Notification Builder

The first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(this)
```

Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following:

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

You have plenty of optional properties which you can set for your notification. To learn more about them, see the reference documentation for *NotificationCompat.Builder*.

Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application. To associate the *PendingIntent* with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the *PendingIntent* by calling **setContentIntent()**.

A `PendingIntent` object helps you to perform an action on your application's behalf, often at a later time, without caring of whether or not your application is running.

We take help of stack builder object which will contain an artificial back stack for the started Activity. This ensures that navigating backward from the Activity leads out of your application to the Home screen.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);
```

Step 4 - Issue the notification

Finally, you pass the `Notification` object to the system by calling `NotificationManager.notify()` to send your notification. Make sure you call **`NotificationCompat.Builder.build()`** method on builder object before notifying it. This method combines all of the options that have been set and return a new **`Notification`** object.

```
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

The NotificationCompat.Builder Class

The `NotificationCompat.Builder` class allows easier control over all the flags, as well as help constructing the typical notification layouts. Following are few important and most frequently used methods available as a part of `NotificationCompat.Builder` class.

S.N.	Constants & Description
1	<p>Notification build()</p> <p>Combine all of the options that have been set and return a new Notification object.</p>
2	<p>NotificationCompat.Builder setAutoCancel (boolean autoCancel)</p> <p>Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel.</p>
3	<p>NotificationCompat.Builder setContent (RemoteViews views)</p> <p>Supply a custom RemoteViews to use instead of the standard one.</p>
4	<p>NotificationCompat.Builder setContentInfo (CharSequence info)</p> <p>Set the large text at the right-hand side of the notification.</p>
5	<p>NotificationCompat.Builder setIntent (PendingIntent intent)</p> <p>Supply a PendingIntent to send when the notification is clicked.</p>
6	<p>NotificationCompat.Builder setContentText (CharSequence text)</p> <p>Set the text (second row) of the notification, in a standard notification.</p>
7	<p>NotificationCompat.Builder setContentTitle (CharSequence title)</p> <p>Set the text (first row) of the notification, in a standard notification.</p>
8	<p>NotificationCompat.Builder setDefaults (int defaults)</p> <p>Set the default notification options that will be used.</p>
9	<p>NotificationCompat.Builder setLargeIcon (Bitmap icon)</p> <p>Set the large icon that is shown in the ticker and notification.</p>
10	<p>NotificationCompat.Builder setNumber (int number)</p> <p>Set the large number at the right-hand side of the notification.</p>

11	NotificationCompat.Builder setOngoing (boolean ongoing) Set whether this is an ongoing notification.
12	NotificationCompat.Builder setSmallIcon (int icon) Set the small icon to use in the notification layouts.
13	NotificationCompat.Builder setStyle (NotificationCompat.Style style) Add a rich notification style to be applied at build time.
14	NotificationCompat.Builder setTicker (CharSequence tickerText) Set the text that is displayed in the status bar when the notification first arrives.
15	NotificationCompat.Builder setVibrate (long[] pattern) Set the vibration pattern to use.
16	NotificationCompat.Builder setWhen (long when) Set the time that the event occurred. Notifications in the panel are sorted by this time.

Example:

Following example shows the functionality of an Android notification using a **NotificationCompat.Builder** Class which has been introduced in Android 4.1.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>NotificationDemo</i> under a package <i>com.example.notificationdemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add the code to define three methods <code>startNotification()</code> , <code>cancelNotification()</code> and <code>updateNotification()</code> to cover maximum functionality related to Android notifications.

3	Create a new Java file <i>src/NotificationView.java</i> , which will be used to display new layout as a part of new activity which will be started when user will click any of the notifications
4	Copy image <i>woman.png</i> in <i>res/drawable-*</i> folder and this image will be used as Notification icon. You can use images with different resolutions in case you want to provide them for different devices.
5	Modify layout XML file <i>res/layout/activity_main.xml</i> to add three buttons in linear layout.
6	Create a new layout XML file <i>res/layout/notification.xml</i> . This will be used as layout file for new activity which will start when user will click any of the notifications.
7	Modify <i>res/values/strings.xml</i> to define required constant values.
8	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.notificationdemo/MainActivity.java**. This file can include each of the fundamental life-cycle methods.

```
package com.example.notificationdemo;

import android.os.Bundle;
import android.app.Activity;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.TaskStackBuilder;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.util.Log;
import android.view.View;
import android.widget.Button;
```

```
public class MainActivity extends Activity {
    private NotificationManager mNotificationManager;
    private int notificationID = 100;
    private int numMessages = 0;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button startBtn = (Button) findViewById(R.id.start);
        startBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                displayNotification();
            }
        });

        Button cancelBtn = (Button) findViewById(R.id.cancel);
        cancelBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                cancelNotification();
            }
        });

        Button updateBtn = (Button) findViewById(R.id.update);
        updateBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                updateNotification();
            }
        });
    }

    protected void displayNotification() {
        Log.i("Start", "notification");

        /* Invoking the default notification service */
    }
}
```

```
NotificationCompat.Builder mBuilder =
new NotificationCompat.Builder(this);

mBuilder.setContentTitle("New Message");
mBuilder.setContentText("You've received new message.");
mBuilder.setTicker("New Message Alert!");
mBuilder.setSmallIcon(R.drawable.woman);

/* Increase notification number every time a new notification
arrives */
mBuilder.setNumber(++numMessages);

/* Creates an explicit intent for an Activity in your app */
Intent resultIntent = new Intent(this, NotificationView.class);

TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(NotificationView.class);

/* Adds the Intent that starts the Activity to the top of the stack
*/
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );

mBuilder.setContentIntent(resultPendingIntent);

mNotificationManager =
(NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);

/* notificationID allows you to update the notification later on.
```

```
    */
    mNotificationManager.notify(notificationID, mBuilder.build());
}

protected void cancelNotification() {
    Log.i("Cancel", "notification");
    mNotificationManager.cancel(notificationID);
}

protected void updateNotification() {
    Log.i("Update", "notification");

    /* Invoking the default notification service */
    NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this);

    mBuilder.setContentTitle("Updated Message");
    mBuilder.setContentText("You've got updated message.");
    mBuilder.setTicker("Updated Message Alert!");
    mBuilder.setSmallIcon(R.drawable.woman);

    /* Increase notification number every time a new notification
    arrives */
    mBuilder.setNumber(++numMessages);

    /* Creates an explicit intent for an Activity in your app */
    Intent resultIntent = new Intent(this, NotificationView.class);

    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(NotificationView.class);

    /* Adds the Intent that starts the Activity to the top of the stack
    */
    stackBuilder.addNextIntent(resultIntent);
```

```

PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );

mBuilder.setContentIntent(resultPendingIntent);

mNotificationManager =
    (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);

/* Update the existing notification using same notification ID */
mNotificationManager.notify(notificationID, mBuilder.build());
}
}

```

Following is the content of the modified main activity file **src/com.example.notificationdemo/NotificationView.java**.

```

package com.example.notificationdemo;

import android.os.Bundle;
import android.app.Activity;

public class NotificationView extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification);
    }
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/start"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start_note"/>

    <Button android:id="@+id/cancel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/cancel_note" />

    <Button android:id="@+id/update"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/update_note" />

</LinearLayout>
```

Following will be the content of **res/layout/notification.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="400dp"
        android:text="Hi, Your Detailed notification view goes here...." />
```

```
</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants:

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

    <string name="app_name">NotificationDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="start_note">Start Notification</string>
    <string name="cancel_note">Cancel Notification</string>
    <string name="update_note">Update Notification</string>

</resources>
```

Following is the default content of **AndroidManifest.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.notificationdemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.notificationdemo.MainActivity"
```

```


        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
/>

        </intent-filter>
    </activity>
    <activity android:name=".NotificationView"
        android:label="Details of notification"
        android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
    </activity>
</application>

</manifest>

```

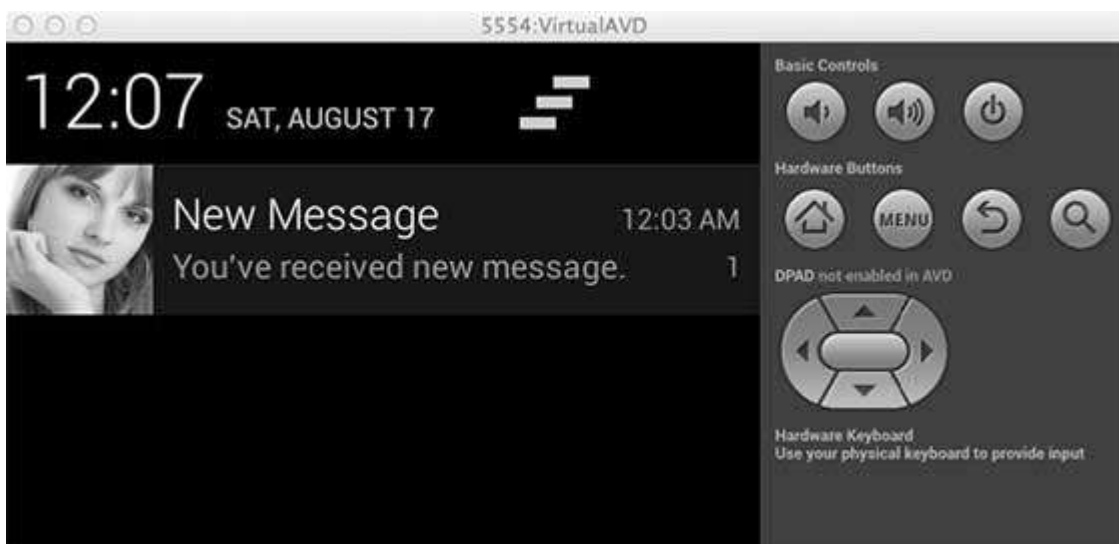
Let's try to run your **NotificationDemo** application. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything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



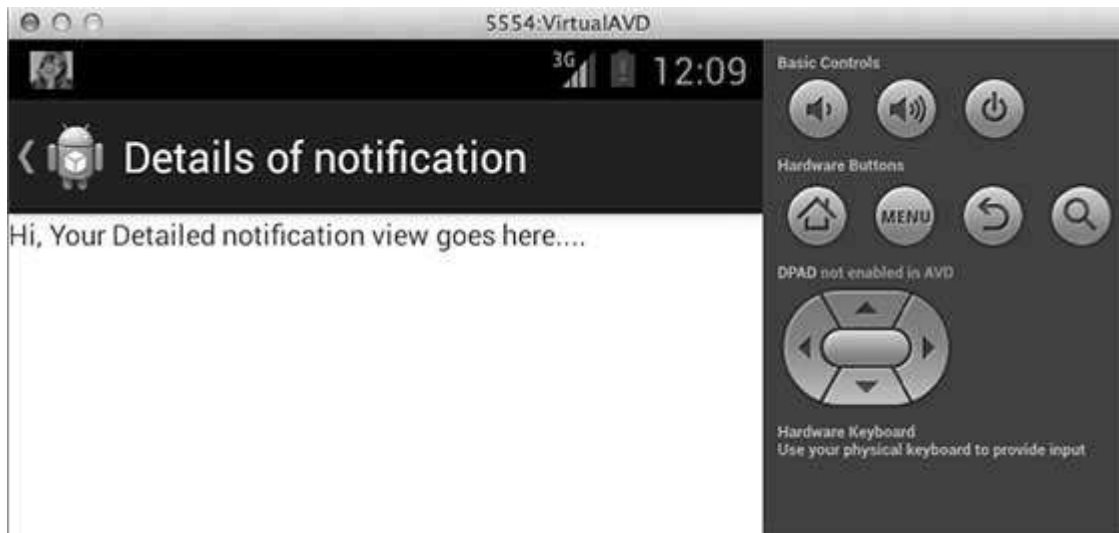
Now click **Start Notification** button, you will see a message "New Message Alert!" at the top that will display momentarily and after that you will have following screen having a small icon at the top left corner.



Now let's expand the view, long click on the small icon, after a second it will display date information and this is the time when you should drag status bar down without releasing mouse. You will see status bar will expand and you will get following screen:



Now let's try to click on the image icon, this will launch your new activity which you have set using intent and you will have following screen:



Next, you can click on "Detail of notification" and it will take you back to the main screen where you can try using **Update Notification** button which will update existing notification and number will increase by 1 but if you will send notification with new notification ID then it will keep adding in the stack and you see them separately listed on the screen.

Big View Notification

The following code snippet demonstrates how to alter the notification created in the previous snippet to use the Inbox big view style. I'm going to update `displayNotification()` modification method to show this functionality:

```
protected void displayNotification() {
    Log.i("Start", "notification");

    /* Invoking the default notification service */
    NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this);

    mBuilder.setContentTitle("New Message");
    mBuilder.setContentText("You've received new message.");
    mBuilder.setTicker("New Message Alert!");
    mBuilder.setSmallIcon(R.drawable.woman);
}
```

```
    /* Increase notification number every time a new notification
arrives */
    mBuilder.setNumber(++numMessages);

    /* Add Big View Specific Configuration */
    NotificationCompat.InboxStyle inboxStyle =
        new NotificationCompat.InboxStyle();

    String[] events = new String[6];
    events[0] = new String("This is first line....");
    events[1] = new String("This is second line...");
    events[2] = new String("This is third line...");
    events[3] = new String("This is 4th line...");
    events[4] = new String("This is 5th line...");
    events[5] = new String("This is 6th line...");

    // Sets a title for the Inbox style big view
    inboxStyle.setBigContentTitle("Big Title Details:");
    // Moves events into the big view
    for (int i=0; i < events.length; i++) {

        inboxStyle.addLine(events[i]);
    }
    mBuilder.setStyle(inboxStyle);

    /* Creates an explicit intent for an Activity in your app */
    Intent resultIntent = new Intent(this, NotificationView.class);

    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(NotificationView.class);

    /* Adds the Intent that starts the Activity to the top of the stack
    */
```

```

stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );

mBuilder.setContentIntent(resultPendingIntent);

mNotificationManager =
    (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);

/* notificationID allows you to update the notification later on.
*/
mNotificationManager.notify(notificationID, mBuilder.build());
}

```

Now if you will try to run your application then you will find following result in expanded form of the view:



21. LOCATION-BASED SERVICES

Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology. This becomes possible with the help of **Google Play services**, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your app to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The **Location** object represents a geographic location which can consist of a latitude, longitude, timestamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information:

S.N.	Method & Description
1	float distanceTo(Location dest) Returns the approximate distance in meters between this location and the given location.
2	float getAccuracy() Get the estimated accuracy of this location, in meters.
3	double getAltitude() Get the altitude if available, in meters above sea level.
4	float getBearing() Get the bearing, in degrees.
5	double getLatitude() Get the latitude, in degrees.
6	double getLongitude()

	Get the longitude, in degrees.
7	float getSpeed() Get the speed if it is available, in meters/second over ground.
8	boolean hasAccuracy() True if this location has an accuracy.
9	boolean hasAltitude() True if this location has an altitude.
10	boolean hasBearing() True if this location has a bearing.
11	boolean hasSpeed() True if this location has a speed.
12	void reset() Clears the contents of the location.
13	void setAccuracy(float accuracy) Set the estimated accuracy of this location, meters.
14	void setAltitude(double altitude) Set the altitude, in meters above sea level.
15	void setBearing(float bearing) Set the bearing, in degrees.
16	void setLatitude(double latitude) Set the latitude, in degrees.
17	void setLongitude(double longitude) Set the longitude, in degrees.

18	void setSpeed(float speed) Set the speed, in meters/second over ground.
19	String toString() Returns a string containing a concise, human-readable description of this object.

Get the Current Location

To get the current location, create a location client which is **LocationClient** object, connect it to Location Services using **connect()** method, and then call its **getLastLocation()** method. This method returns the most recent location in the form of **Location** object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces:

- **GooglePlayServicesClient.ConnectionCallbacks**
- **GooglePlayServicesClient.OnConnectionFailedListener**

These interfaces provide following important callback methods, which you need to implement in your activity class:

S.N.	Callback Methods & Description
1	abstract void onConnected(Bundle connectionHint) This callback method is called when location service is connected to the location client successfully. You will use connect() method to connect to the location client.
2	abstract void onDisconnected() This callback method is called when the client is disconnected. You will use disconnect() method to disconnect from the location client.
3	abstract void onConnectionFailed(ConnectionResult result) This callback method is called when there was an error connecting the client to the service.

You should create the location client in **onCreate()** method of your activity class, then connect it in **onStart()**, so that Location Services maintains the

current location while your activity is fully visible. You should disconnect the client in **onStop()** method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement **LocationListener** interface as well. This interface provide following callback method, which you need to implement in your activity class:

S.N.	Callback Method & Description
1	<p>abstract void onLocationChanged(Location location)</p> <p>This callback method is used for receiving notifications from the LocationClient when the location has changed.</p>

Location Quality of Service

The **LocationRequest** object is used to request a quality of service (QoS) for location updates from the **LocationClient**. There are following useful setter methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

S.N.	Method & Description
1	<p>setExpirationDuration(long millis)</p> <p>Set the duration of this request, in milliseconds.</p>
2	<p>setExpirationTime(long millis)</p> <p>Set the request expiration time, in millisecond since boot.</p>
3	<p>setFastestInterval(long millis)</p> <p>Explicitly set the fastest interval for location updates, in milliseconds.</p>
4	<p>setInterval(long millis)</p> <p>Set the desired interval for active location updates, in milliseconds.</p>

5	setNumUpdates(int numUpdates) Set the number of location updates.
6	setPriority(int priority) Set the priority of the request.

Now for example, if your application wants high accuracy location it should create a location request with **setPriority(int)** set to `PRIORITY_HIGH_ACCURACY` and **setInterval(long)** to 5 seconds. You can also use bigger interval and/or other priorities like `PRIORITY_LOW_POWER` for request "city" level accuracy or `PRIORITY_BALANCED_POWER_ACCURACY` for "block" level accuracy.

Activities should strongly consider removing all location request when entering the background (for example at `onPause()`), or at least swap the request to a larger interval and lower quality to save power consumption.

Displaying a Location Address

Once you have **Location** object, you can use **Geocoder.getFromLocation()** method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the **doInBackground()** method of an **AsyncTask** class.

The **AsyncTask** must be subclassed to be used and the subclass will override **doInBackground(Params...)** method to perform a task in the background and **onPostExecute(Result)** method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in `AsyncTask` which is **execute(Params... params)**, this method executes the task with the specified parameters.

Check following example to have better understanding on how we use `AsyncTask` in any Android application to get work done in the background without interfering main task.

Example:

Following example shows you in practical how to use Location Services in your app to get the current location and its equivalent addresses etc.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Install the Google Play Services SDK

Before you proceed to have location support in your Android Applications, you need to setup **Google Play Services SDK** using following simple steps:

google-play-services_lib/ to the location where you maintain your Android app projects. If you are using Eclipse, import the library

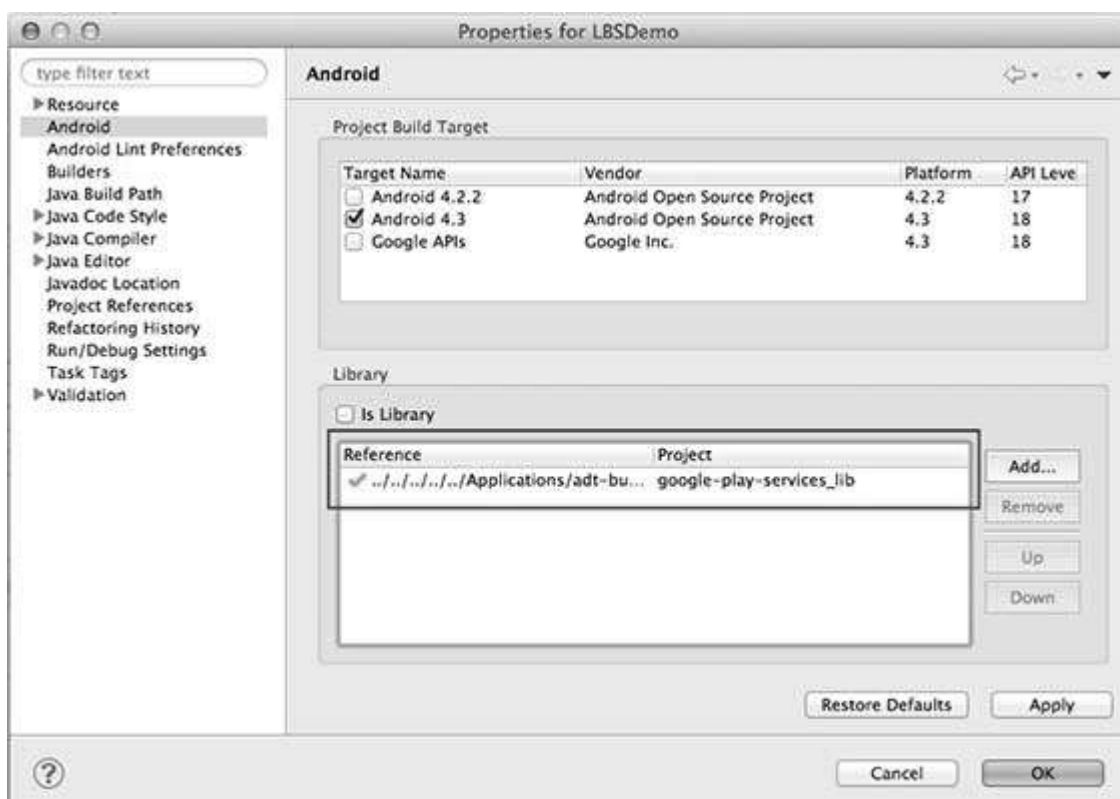
Steps	Description
1	<p>Launch the SDK Manager.</p> <p>From Eclipse (with ADT), select Window > Android SDK Manager.</p> <p>On Windows, double-click the SDK Manager.exe file at the root of the Android SDK directory.</p> <p>On Mac or Linux, open a terminal and navigate to the tools/ directory in the Android SDK directory, then execute android sdk.</p>
2	<p>Search for Google Play services option from the given package list under Extra and if it is not installed, then install it. The Google Play services SDK is saved in your Android SDK environment at <code><android-sdk>/extras/google/google_play_services/</code>.</p>
3	<p>Copy the library project at <code><android-sdk>/extras/google/google_play_services/libproject/</code> project into your workspace. Click File > Import, select Android > Existing Android Code into Workspace, and browse to <code><android-sdk>/extras/google/google_play_services/libproject/</code>, library project to import it.</p>

Create Android Application

Step	Description
1	<p>You will use Eclipse IDE to create an Android application and name it as <code>LBSDemo/i></code> under a package <code>com.example.lbsdemo</code>. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.</p>
2	<p>Add Google Play Service library in your project by following simple steps given below.</p>

3	Modify <i>src/MainActivity.java</i> file and add required code as shown below to take care of getting current location and its equivalent address.
4	Modify layout XML file <i>res/layout/activity_main.xml</i> to add all GUI components which include three buttons and two text views to show location/address.
5	Modify <i>res/values/strings.xml</i> to define required constant values.
6	Modify <i>AndroidManifest.xml</i> as shown below.
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Let's add **Google Play Service** reference in the project. Right click on the project and select **Build Path > Configure Build Path > Android >** and then click Add button which will show *google-play-service_lib* option to be added, just double click on it, which will add required library reference and you will have window as follows:



Following is the content of the modified main activity file **src/com.example.lbsdemo/MainActivity.java**.

```
package com.example.lbsdemo;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GooglePlayServicesClient;
import com.google.android.gms.location.LocationClient;

import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends FragmentActivity implements
GooglePlayServicesClient.ConnectionCallbacks,
GooglePlayServicesClient.OnConnectionFailedListener
{
    LocationClient mLocationClient;
    private TextView addressLabel;
    private TextView locationLabel;
    private Button getLocationBtn;
    private Button disconnectBtn;
    private Button connectBtn;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    locationLabel = (TextView) findViewById(R.id.locationLabel);
    addressLabel = (TextView) findViewById(R.id.addressLabel);
    getLocationBtn = (Button) findViewById(R.id.getLocation);

    getLocationBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            displayCurrentLocation();
        }
    });
    disconnectBtn = (Button) findViewById(R.id.disconnect);
    disconnectBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            mLocationClient.disconnect();
            locationLabel.setText("Got disconnected....");
        }
    });
    connectBtn = (Button) findViewById(R.id.connect);
    connectBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            mLocationClient.connect();
            locationLabel.setText("Got connected....");
        }
    });
    // Create the LocationRequest object
    mLocationClient = new LocationClient(this, this, this);
}
@Override
protected void onStart() {
```

```
        super.onStart();
        // Connect the client.
        mLocationClient.connect();
        locationLabel.setText("Got connected....");
    }
    @Override
    protected void onStop() {
        // Disconnect the client.
        mLocationClient.disconnect();
        super.onStop();
        locationLabel.setText("Got disconnected....");
    }
    @Override
    public void onConnected(Bundle dataBundle) {
        // Display the connection status
        Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onDisconnected() {
        // Display the connection status
        Toast.makeText(this, "Disconnected. Please re-connect.",
            Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        // Display the error code on failure
        Toast.makeText(this, "Connection Failure : " +
            connectionResult.getErrorCode(),
            Toast.LENGTH_SHORT).show();
    }
    public void displayCurrentLocation() {
        // Get the current location's latitude & longitude
        Location currentLocation = mLocationClient.getLastLocation();
        String msg = "Current Location: " +
```

```

        Double.toString(currentLocation.getLatitude()) + "," +
        Double.toString(currentLocation.getLongitude());

        // Display the current location in the UI
        locationLabel.setText(msg);

        // To display the current address in the UI
        (new GetAddressTask(this)).execute(currentLocation);
    }
    /*
     * Following is a subclass of AsyncTask which has been used to get
     * address corresponding to the given latitude & longitude.
     */
    private class GetAddressTask extends AsyncTask<Location, Void,
    String>{
        Context mContext;
        public GetAddressTask(Context context) {
            super();
            mContext = context;
        }

        /*
         * When the task finishes, onPostExecute() displays the address.
         */
        @Override
        protected void onPostExecute(String address) {
            // Display the current address in the UI
            addressLabel.setText(address);
        }
        @Override
        protected String doInBackground(Location... params) {
            Geocoder geocoder =
            new Geocoder(mContext, Locale.getDefault());
            // Get the current location from the input parameter list

```

```
Location loc = params[0];
// Create a list to contain the result address
List<Address> addresses = null;
try {
    addresses = geocoder.getFromLocation(loc.getLatitude(),
        loc.getLongitude(), 1);
} catch (IOException e1) {
    Log.e("LocationSampleActivity",
        "IO Exception in getFromLocation()");
    e1.printStackTrace();
    return ("IO Exception trying to get address");
} catch (IllegalArgumentException e2) {
    // Error message to post in the log
    String errorString = "Illegal arguments " +
        Double.toString(loc.getLatitude()) +
        ", " +
        Double.toString(loc.getLongitude()) +
        " passed to address service";
    Log.e("LocationSampleActivity", errorString);
    e2.printStackTrace();
    return errorString;
}
// If the reverse geocode returned an address
if (addresses != null && addresses.size() > 0) {
    // Get the first address
    Address address = addresses.get(0);
    /*
     * Format the first line of address (if available),
     * city, and country name.
     */
    String addressText = String.format(
        "%s, %s, %s",
        // If there's a street address, add it
        address.getMaxAddressLineIndex() > 0 ?
```



```

        address.getAddressLine(0) : "",
        // Locality is usually a city
        address.getLocality(),
        // The country of the address
        address.getCountryName());
        // Return the text
        return addressText;
    } else {
        return "No address found";
    }
}
} // AsyncTask class
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/getLocation"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/get_location"/>

    <Button android:id="@+id/disconnect"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/disconnect"/>

    <Button android:id="@+id/connect"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/connect"/>

```

```

    <TextView
    android:id="@+id/locationLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

    <TextView
    android:id="@+id/addressLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">LBSDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="get_location">Get Location</string>
    <string name="disconnect">Disconnect Service</string>
    <string name="connect">Connect Service</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.lbsdemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk

```

```

        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>


<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.lbsdemo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

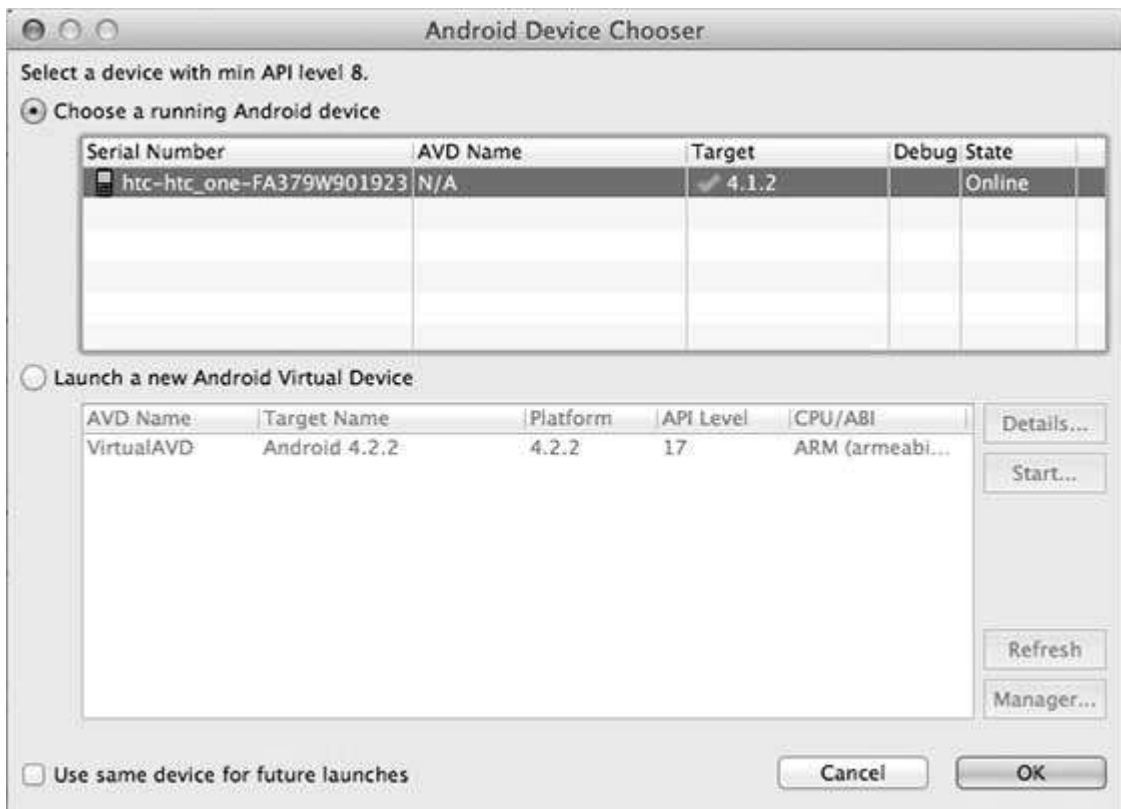
            <category android:name="android.intent.category.LAUNCHER"
/>

        </intent-filter>
    </activity>
</application>

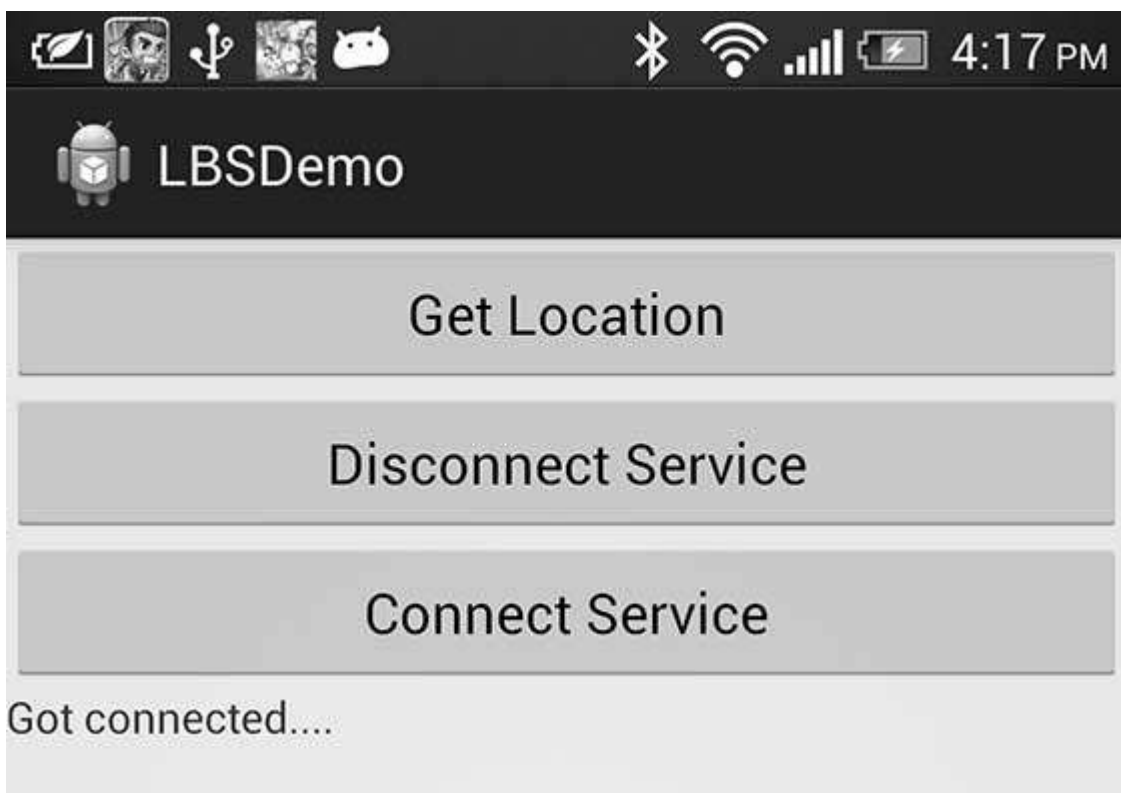
</manifest>

```

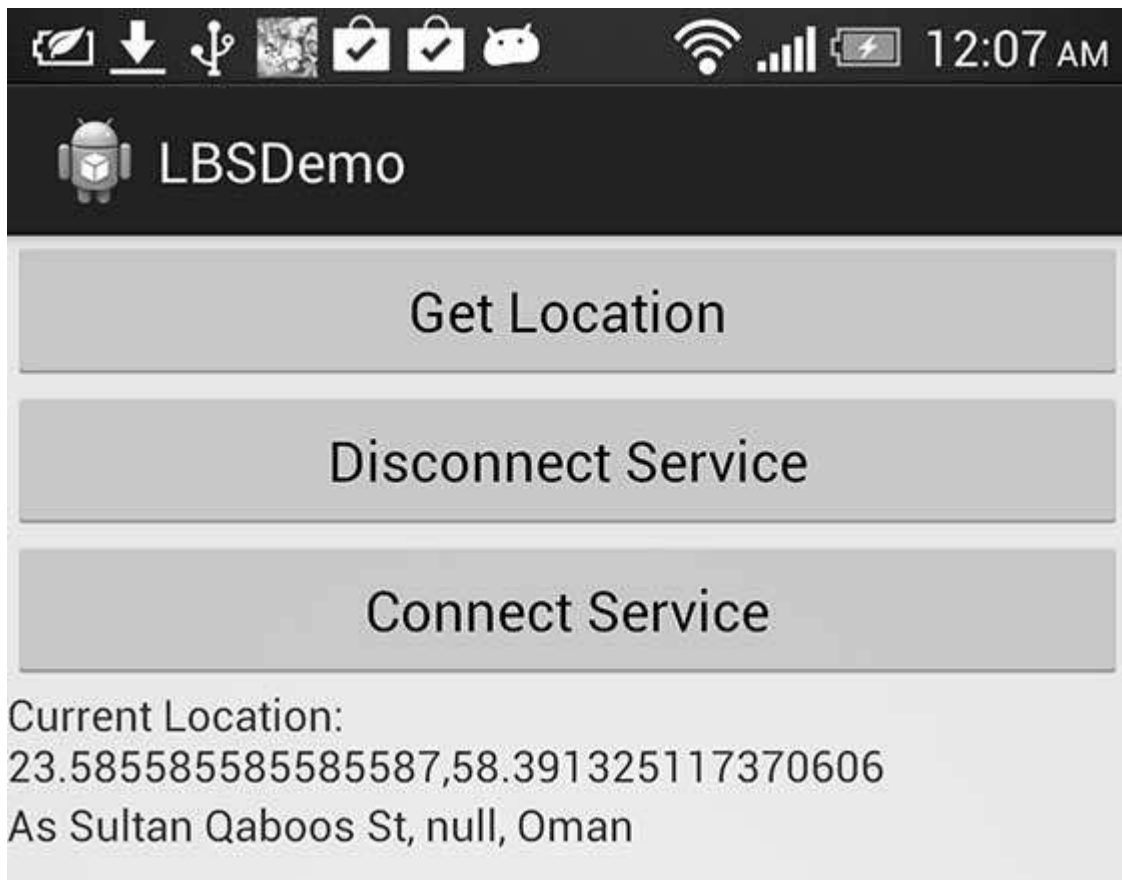
Let's try to run your **LBSDemo** application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select mobile device as an option and then check your mobile device which will display following screen:



Now to see location select Get Location Button which will display location information as follows:



You can try by disconnecting location client using **Disconnect Service** and then connecting it by using **Connect Service** button. You can also modify to get location update as explained above and in Android Official documentation.

22. SENDING EMAIL

You have learnt Android Intent, which is an object carrying an intent i.e. message from one component to another component within the application or outside the application.

As such you do not need to develop your email client from scratch because they are already available like Gmail and K9mail. But you will need to send email from your Android application, where you will have to write an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity will send an ACTION_SEND along with appropriate data load, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

Following section explains different parts of our Intent object required to send an email.

Intent Object - Action to send Email

You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

Intent Object - Data/Type to send Email

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows:

```
emailIntent.setData(Uri.parse("mailto:"));  
emailIntent.setType("text/plain");
```

Intent Object - Extra to send Email

Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client. You can use following extra fields in your email:

S.N.	Extra Data & Description
1	EXTRA_BCC

	A String[] holding e-mail addresses that should be blind carbon copied.
2	EXTRA_CC A String[] holding e-mail addresses that should be carbon copied.
3	EXTRA_EMAIL A String[] holding e-mail addresses that should be delivered to.
4	EXTRA_HTML_TEXT A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
5	EXTRA_SUBJECT A constant string holding the desired subject line of a message.
6	EXTRA_TEXT A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent.
7	EXTRA_TITLE A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.

Here is an example showing you how to assign extra data to your intent:

```
emailIntent.putExtra(Intent.EXTRA_EMAIL , new
String[]{"recipient@example.com"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject of email");
emailIntent.putExtra(Intent.EXTRA_TEXT , "body of email");
```

Example:

Following example shows you in practical how to use Intent object to launch Email client to send an Email to the given recipients.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work. Second you will need to have an Email client like GMail or K9mail installed on your device.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>SendEmailDemo</i> under a package <i>com.example.sendemailldemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending email.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to launch Email Client.
4	Modify <i>res/values/strings.xml</i> to define required constant values.
5	Modify <i>AndroidManifest.xml</i> as shown below.
6	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.sendemailldemo/MainActivity.java**.

```
package com.example.sendemailldemo;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
```



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button startBtn = (Button) findViewById(R.id.sendEmail);
    startBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            sendEmail();
        }
    });

}

protected void sendEmail() {
    Log.i("Send email", "");

    String[] TO = {"amrood.admin@gmail.com"};
    String[] CC = {"mcmohd@gmail.com"};
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:"));
    emailIntent.setType("text/plain");

    emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);
    emailIntent.putExtra(Intent.EXTRA_CC, CC);
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Your subject");
    emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message goes here");

    try {
        startActivity(Intent.createChooser(emailIntent, "Send
            mail..."));

        finish();

        Log.i("Finished sending email...", "");
    }
}
```

```

    } catch (android.content.ActivityNotFoundException ex) {
        Toast.makeText(MainActivity.this,
            "There is no email client installed.",
            Toast.LENGTH_SHORT).show();
    }
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/sendEmail"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/compose_email"/>

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">SendEmailDemo</string>
    <string name="hello_world">Hello world!</string>

```

```

    <string name="action_settings">Settings</string>
    <string name="compose_email">Compose Email</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sendemaildemo"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.sendemaildemo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
                    />
            </intent-filter>
        </activity>
    </application>
</manifest>

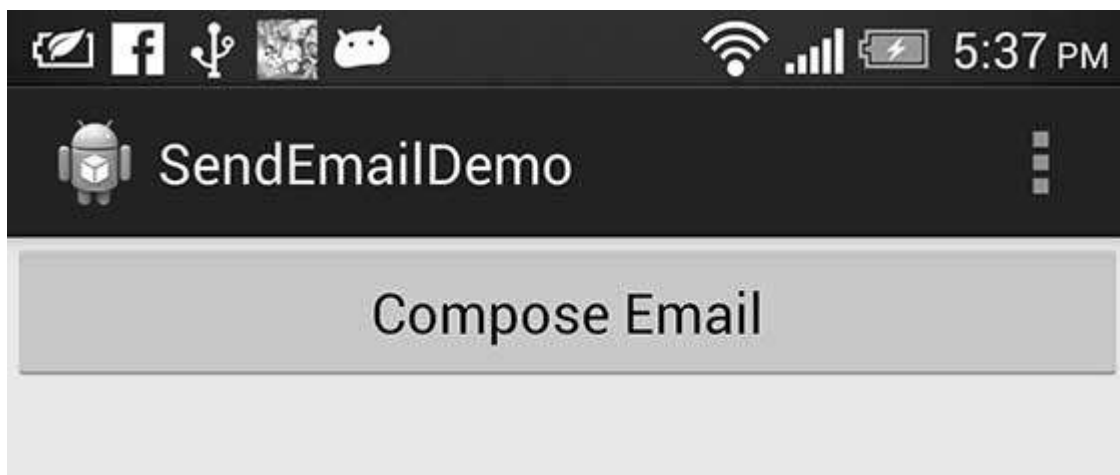
```

Let's try to run your **SendEmailDemo** application. We assume, you have connected your actual Android Mobile device with your computer. To run the app

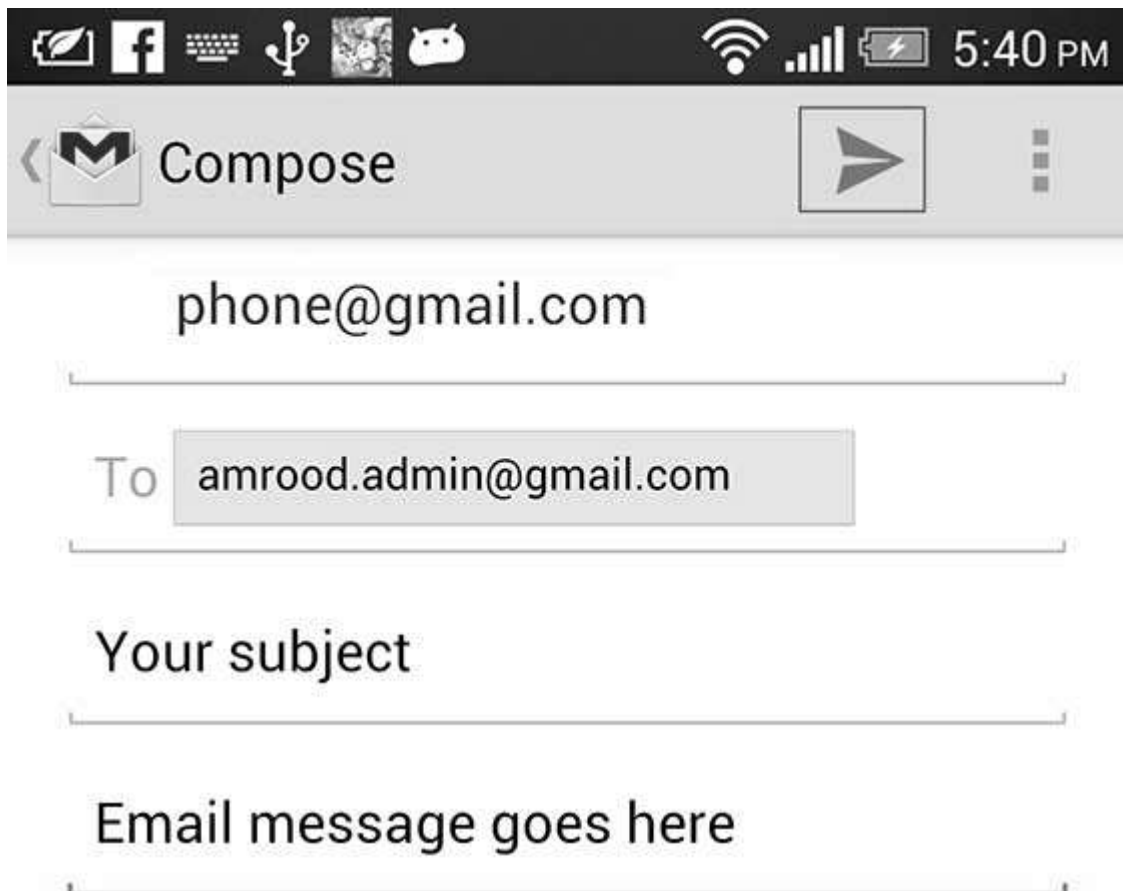
from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Now use **Compose Email** button to list down all the installed email clients. From the list, you can choose one of email clients to send your email. I'm going to use Gmail client to send my email which will have all the provided defaults fields available as shown below. Here **From:** will be default email ID you have registered for your Android device.



You can modify either of the given default fields and finally use send email button (marked with red rectangle) to send your email to the mentioned recipients.

23. SENDING SMS

There are following two ways to send SMS using Android device:

- Using SmsManager to send SMS
- Using Built-in Intent to send SMS

Using SmsManager to send SMS

The SmsManager manages SMS operations such as sending data to the given mobile device. You can create this object by calling the static method `SmsManager.getDefault()` as follows:

```
SmsManager smsManager = SmsManager.getDefault();
```

Once you have SmsManager object, you can use `sendDataMessage()` method to send SMS at the specified mobile number as below:

```
smsManager.sendTextMessage("phoneNo", null, "SMS text", null, null);
```

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below:

S.N.	Method & Description
1	ArrayList<String> divideMessage(String text) This method divides a message text into several fragments, none bigger than the maximum SMS message size.
2	static SmsManager getDefault() This method is used to get the default instance of the SmsManager
3	void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent) This method is used to send a data based SMS to a specific application port.
4	void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts,

	<p>ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)</p> <p>Send a multi-part text based SMS.</p>
5	<p>void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)</p> <p>Send a text based SMS.</p>

Example:

Following example shows you in practical how to use SmsManager object to send an SMS to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>SendSMSDemo</i> under a package <i>com.example.sendsmsdemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending email.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple GUI to take mobile number and SMS text to be sent and a simple button to send SMS.
4	Modify <i>res/values/strings.xml</i> to define required constant values.
5	Modify <i>AndroidManifest.xml</i> as shown below.
6	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.sendsmsdemo/MainActivity.java**.

```
package com.example.sendsmsdemo;

import android.os.Bundle;
import android.app.Activity;
import android.telephony.SmsManager;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    Button sendBtn;
    EditText txtphoneNo;
    EditText txtMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sendBtn = (Button) findViewById(R.id.btnSendSMS);
        txtphoneNo = (EditText) findViewById(R.id.editTextPhoneNo);
        txtMessage = (EditText) findViewById(R.id.editTextSMS);

        sendBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                sendSMSMessage();
            }
        });
    }
}
```



```

    }
    protected void sendSMSMessage() {
        Log.i("Send SMS", "");

        String phoneNo = txtphoneNo.getText().toString();
        String message = txtMessage.getText().toString();

        try {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage(phoneNo, null, message, null, null);
            Toast.makeText(getApplicationContext(), "SMS sent.",
                Toast.LENGTH_LONG).show();
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(),
                "SMS failed, please try again.",
                Toast.LENGTH_LONG).show();
            e.printStackTrace();
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
        present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<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/textViewPhoneNo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/phone_label" />

<EditText
    android:id="@+id/editTextPhoneNo"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="phone"/>

<TextView
    android:id="@+id/textViewMessage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sms_label" />

<EditText
    android:id="@+id/editTextSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="textMultiLine"/>

<Button android:id="@+id/btnSendSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/send_sms_label"/>

</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants:

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

    <string name="app_name">SendSMSDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="phone_label">Enter Phone Number:</string>
    <string name="sms_label">Enter SMS Message:</string>
    <string name="send_sms_label">Send SMS</string>

</resources>
```

Following is the default content of **AndroidManifest.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sendsmsdemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.SEND_SMS" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.sendsmsdemo.MainActivity"
            android:label="@string/app_name" >
```


```

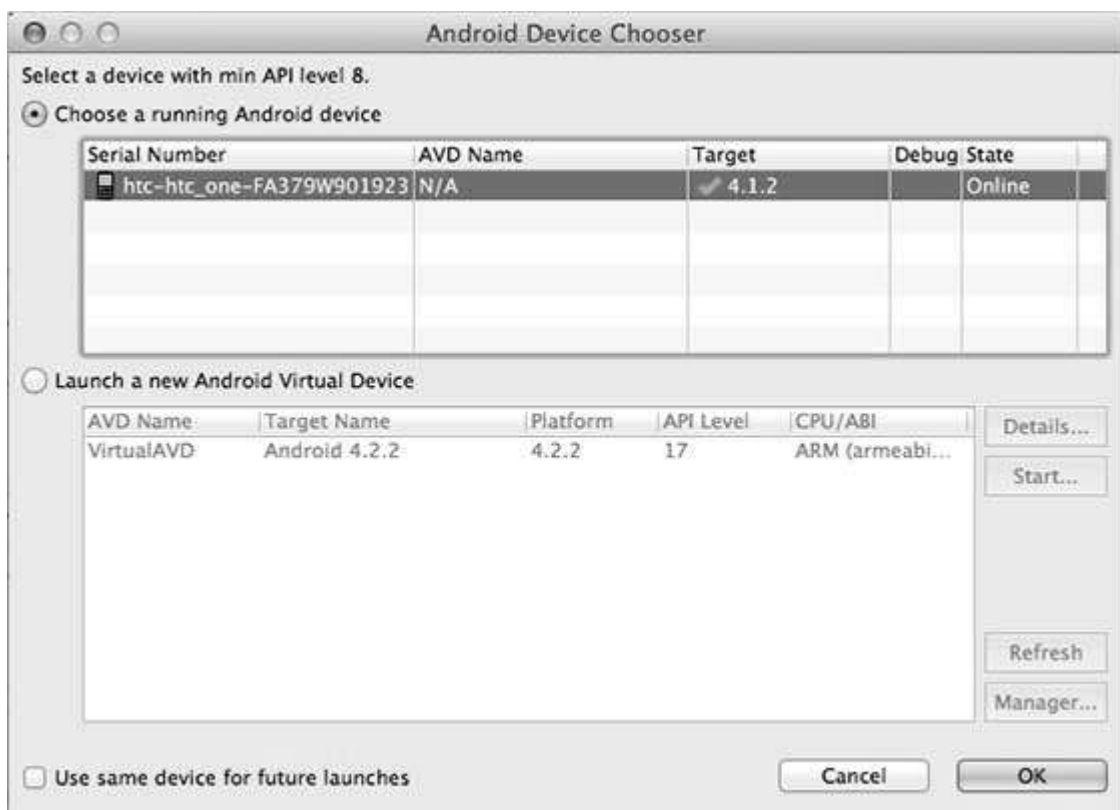
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
                />
        </intent-filter>
    </activity>
</application>

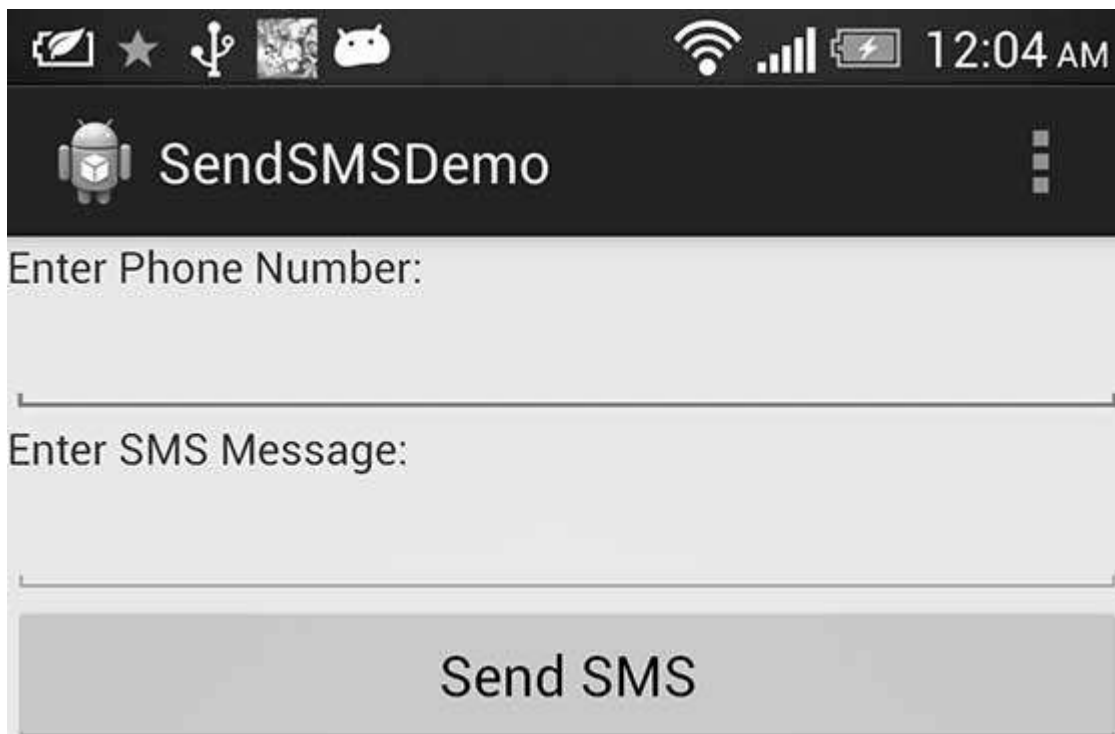
</manifest>

```

Let's try to run your **SendSMSDemo** application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Now you can enter a desired mobile number and a text message to be sent on that number. Finally click on **Send SMS** button to send your SMS. Make sure your GSM connection is working fine to deliver your SMS to its recipient.

You can take a number of SMS separated by comma and then inside your program you will have to parse them into an array string and finally you can use a loop to send message to all the given numbers. That's how you can write your own SMS client. Next section will show you how to use existing SMS client to send SMS.

Using Built-in Intent to send SMS

You can use Android Intent to send SMS by calling built-in SMS functionality of the Android. Following section explains different parts of our Intent object required to send an SMS.

Intent Object - Action to send SMS

You will use **ACTION_VIEW** action to launch an SMS client installed on your Android device. Following is a simple syntax to create an intent with ACTION_VIEW action.

```
Intent smsIntent = new Intent(Intent.ACTION_VIEW);
```

Intent Object - Data/Type to send SMS

To send an SMS you need to specify **smsto:** as URI using setData() method and data type will be to **vnd.android-dir/mms-sms** using setType() method as follows:

```
smsIntent.setData(Uri.parse("smsto:"));
smsIntent.setType("vnd.android-dir/mms-sms");
```

Intent Object - Extra to send SMS

Android has built-in support to add phone number and text message to send an SMS as follows:

```
smsIntent.putExtra("address" , new String("0123456789;3393993300"));
smsIntent.putExtra("sms_body" , "Test SMS to Angilla");
```

Here address and sms_body are case sensitive and should be specified in small characters only. You can specify more than one number in single string but separated by semi-colon (;).

Example:

Following example shows you in practical how to use Intent object to launch SMS client to send an SMS to the given recipients.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>SendSMSDemo</i> under a package <i>com.example.sendsmsdemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending SMS.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to launch SMS Client.
4	Modify <i>res/values/strings.xml</i> to define required constant values.

5	Modify <i>AndroidManifest.xml</i> as shown below.
6	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.sendsmsdemo/MainActivity.java**.

```
package com.example.sendsmsdemo;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button startBtn = (Button) findViewById(R.id.sendSMS);
        startBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                sendSMS();
            }
        });
    }
};
```

```

}
protected void sendSMS() {
    Log.i("Send SMS", "");

    Intent smsIntent = new Intent(Intent.ACTION_VIEW);
    smsIntent.setData(Uri.parse("smsto:"));
    smsIntent.setType("vnd.android-dir/mms-sms");

    smsIntent.putExtra("address" , new String ("0123456789"));
    smsIntent.putExtra("sms_body" , "Test SMS to Angilla");
    try {
        startActivity(smsIntent);
        finish();
        Log.i("Finished sending SMS...", "");
    } catch (android.content.ActivityNotFoundException ex) {
        Toast.makeText(MainActivity.this,
            "SMS failed, please try again later.",
            Toast.LENGTH_SHORT).show();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"

```



```

    android:orientation="vertical" >

    <Button android:id="@+id/sendSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/compose_sms"/>

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">SendSMSDemo</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="compose_sms">Compose SMS</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sendsmsdemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"

```

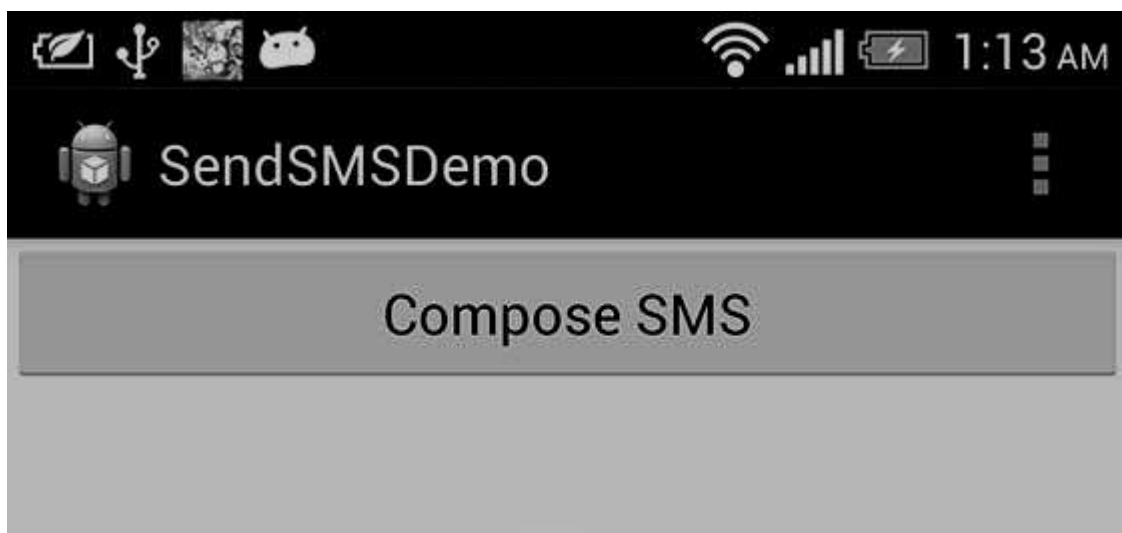
```
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.sendsmsdemo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
                />
        </intent-filter>
    </activity>
</application>
</manifest>
```

Let's try to run your **SendSMSDemo** application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Now use **Compose SMS** button to launch Android built-in SMS clients which is shown below:



You can modify either of the given default fields and finally use send SMS button (marked with red rectangle) to send your SMS to the mentioned recipient.

24. PHONE CALLS

As such every Android Device specially Mobile phone is meant to provide a functionality to make a phone call but still you may need to write an application where you want to give an option to your user to make a call using a hard coded phone number.

This chapter lists down all the simple steps to create an application which can be used to make a Phone Call. You can use Android Intent to make phone call by calling built-in Phone Call functionality of the Android. Following section explains different parts of our Intent object required to make a call.

Intent Object - Action to make Phone Call

You will use **ACTION_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION_CALL action.

```
Intent phoneIntent = new Intent(Intent.ACTION_CALL);
```

You can use **ACTION_DIAL** action instead of ACTION_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

Intent Object - Data/Type to make Phone Call

To make a phone call at a given number 91-800-001-0101, you need to specify **tel:** as URI using setData() method as follows:

```
phoneIntent.setData(Uri.parse("tel:91-800-001-0101"));
```

The interesting point is that, to make a phone call, you do not need to specify any extra data or data type.

Example:

Following example shows you in practical how to use Android Intent to make phone call to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>PhoneCallDemo</i> under a package <i>com.example.phonecalldemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of making a call.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to Call 91-800-001-0101 number.
4	Modify <i>res/values/strings.xml</i> to define required constant values.
5	Modify <i>AndroidManifest.xml</i> as shown below.
6	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.phonecalldemo/MainActivity.java**.

```
package com.example.phonecalldemo;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button startBtn = (Button) findViewById(R.id.makeCall);
    startBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            makeCall();
        }
    });

}

protected void makeCall() {
    Log.i("Make call", "");

    Intent phoneIntent = new Intent(Intent.ACTION_CALL);
    phoneIntent.setData(Uri.parse("tel:91-800-001-0101"));

    try {
        startActivity(phoneIntent);
        finish();
        Log.i("Finished making a call...", "");
    } catch (android.content.ActivityNotFoundException ex) {
        Toast.makeText(MainActivity.this,
            "Call failed, please try again later.",
            Toast.LENGTH_SHORT).show();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
    present.
```

```

        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/makeCall"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/make_call"/>

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants:

```

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

    <string name="app_name">PhoneCallDemo</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="make_call">Call 91-800-001-0101</string>

</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.phonecalldemo"

```



```

android:versionCode="1"
android:versionName="1.0" >

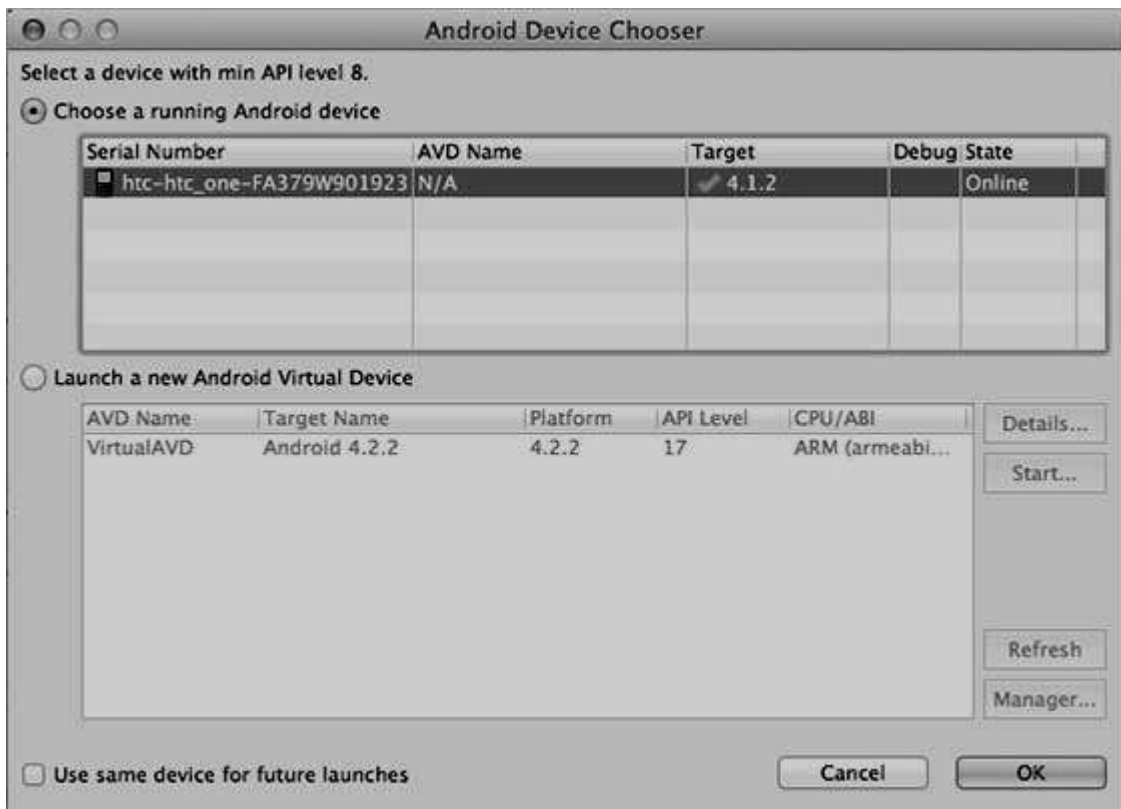
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"
/>

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.phonecalldemo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

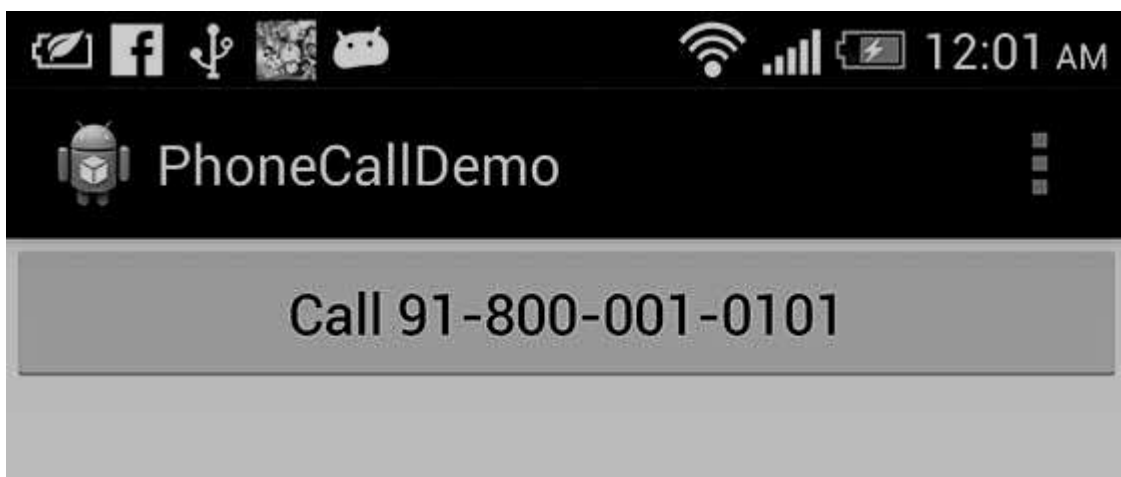
            <category
                android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>

```

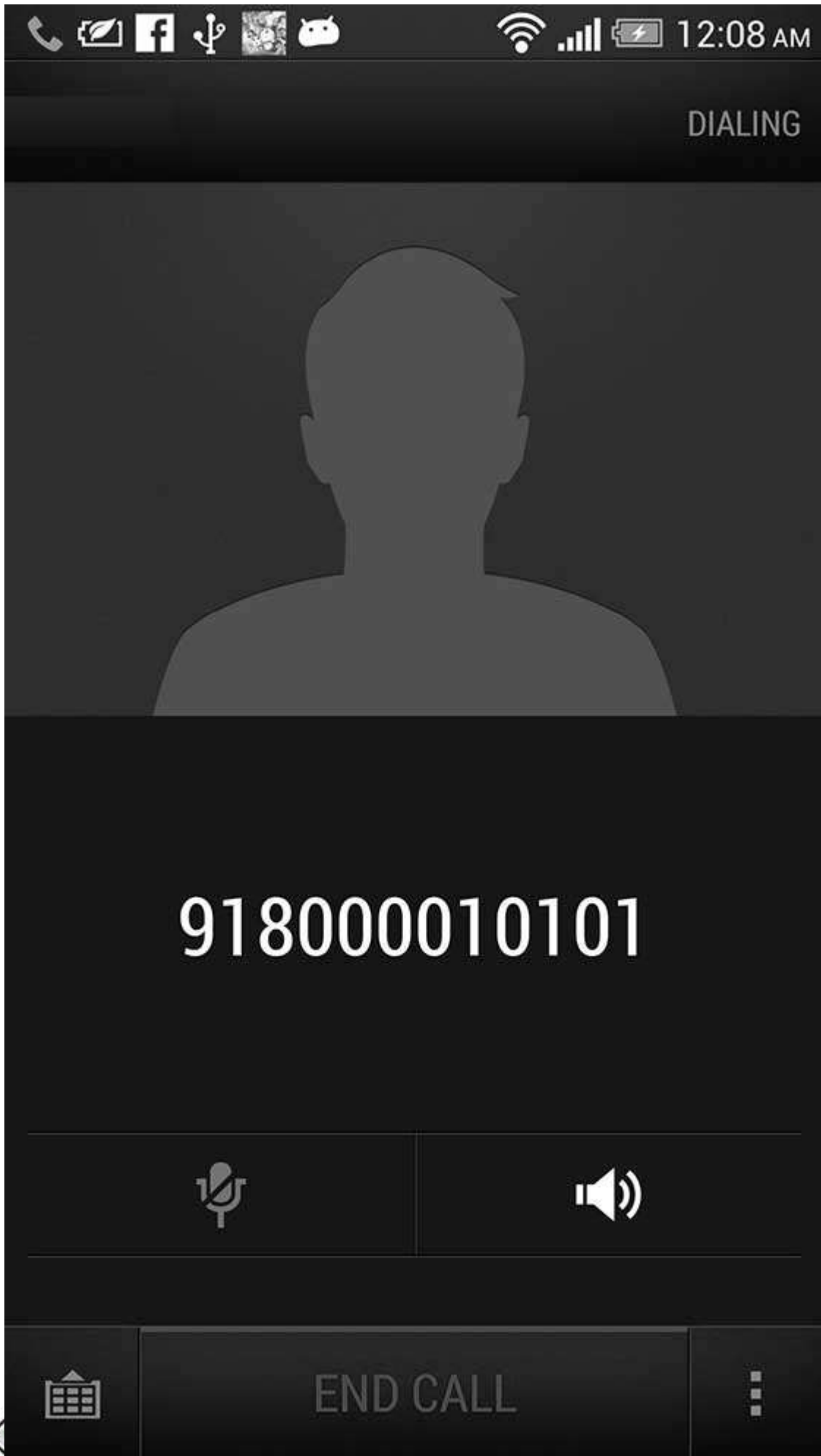
Let's try to run your **PhoneCallDemo** application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Now use **Call 91-800-001-0101** button to make phone call as shown below:



25. PUBLISHING ANDROID APPLICATION

Android application publishing is a process that makes your Android applications available to users. Infact, publishing is the last phase of the Android application development process.



Once you have developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application:

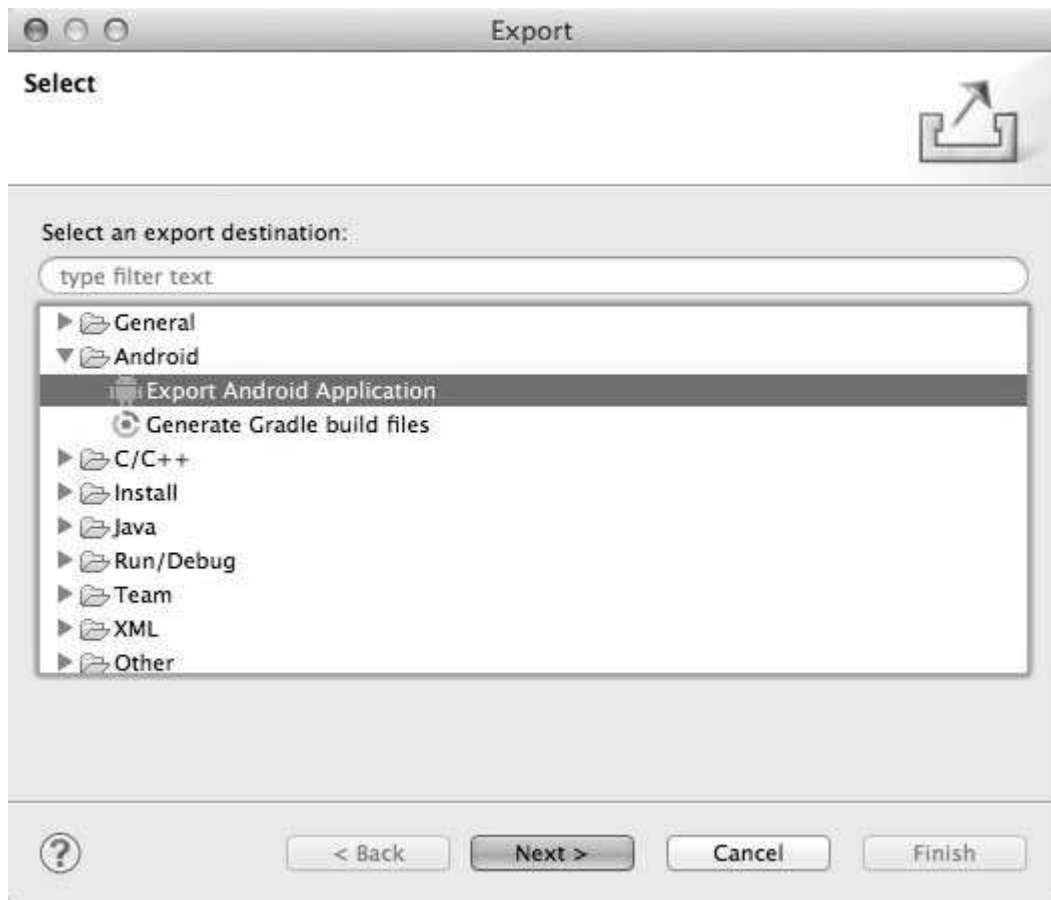
Step	Activity
1	Regression Testing Before you publish your application, you need to make sure that it is meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	Application Rating When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	Targeted Regions Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone, localization or any other specific requirement as per the targeted region.
4	Application Size Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and

	automatically handle the download to devices.
5	SDK and Screen Compatibility It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.
6	Application Pricing Deciding whether your app will be free or paid is important because, on Google Play, free apps must remain free. If you want to sell your application then you will have to specify its price in different currencies.
7	Promotional Content It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	Build and Upload release-ready APK The release-ready APK is what you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: Preparing for Release .
9	Finalize Application Detail Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colorful graphics, screenshots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

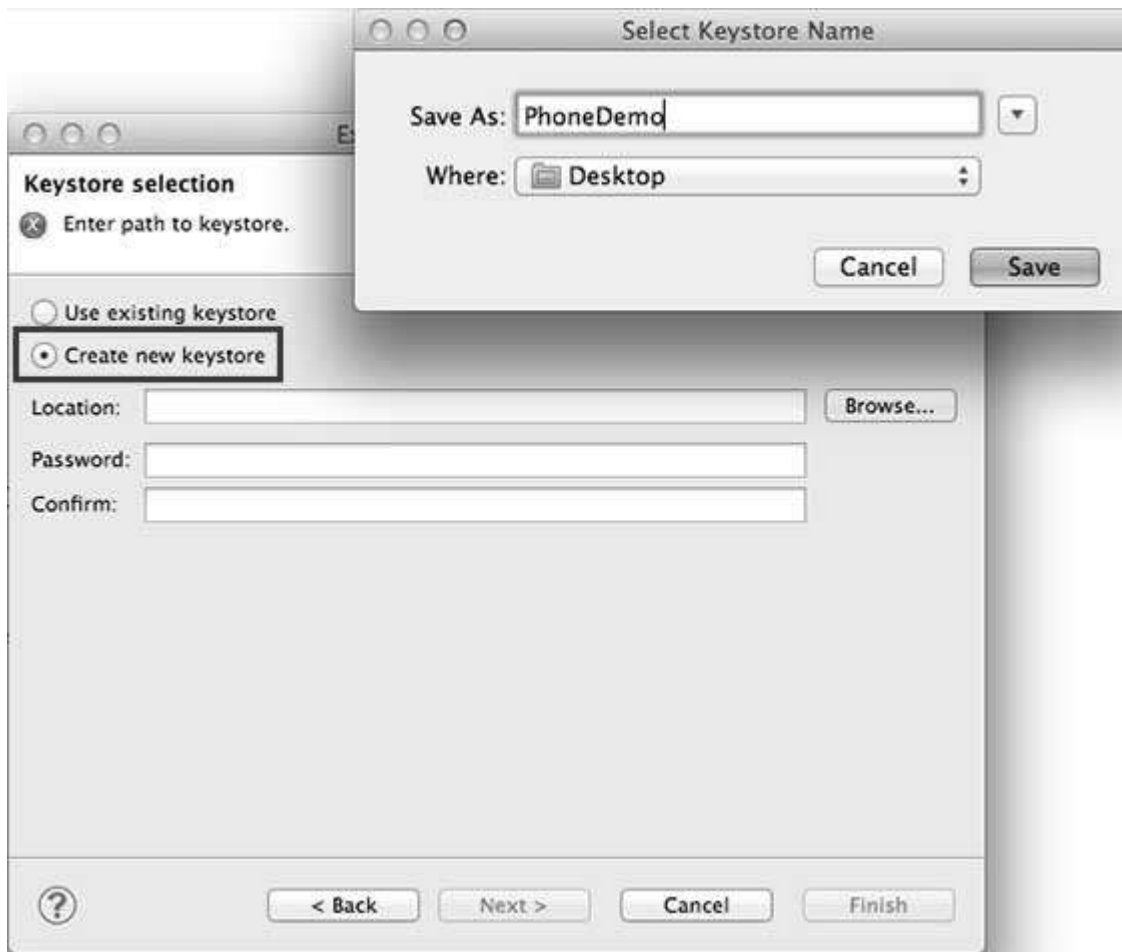
Export Android Application

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

To export an application, just open that application project in Eclipse and select **File->Export** from your Eclipse and follow the simple steps to export your application:



Next, select **Export Android Application** option as shown in the above screen shot and then click **Next** and again **Next** so that you get following screen where you will choose **Create new keystore** to store your application.



Enter your password to protect your application and click on **Next** button once again. It will display following screen to let you create a key for your application:

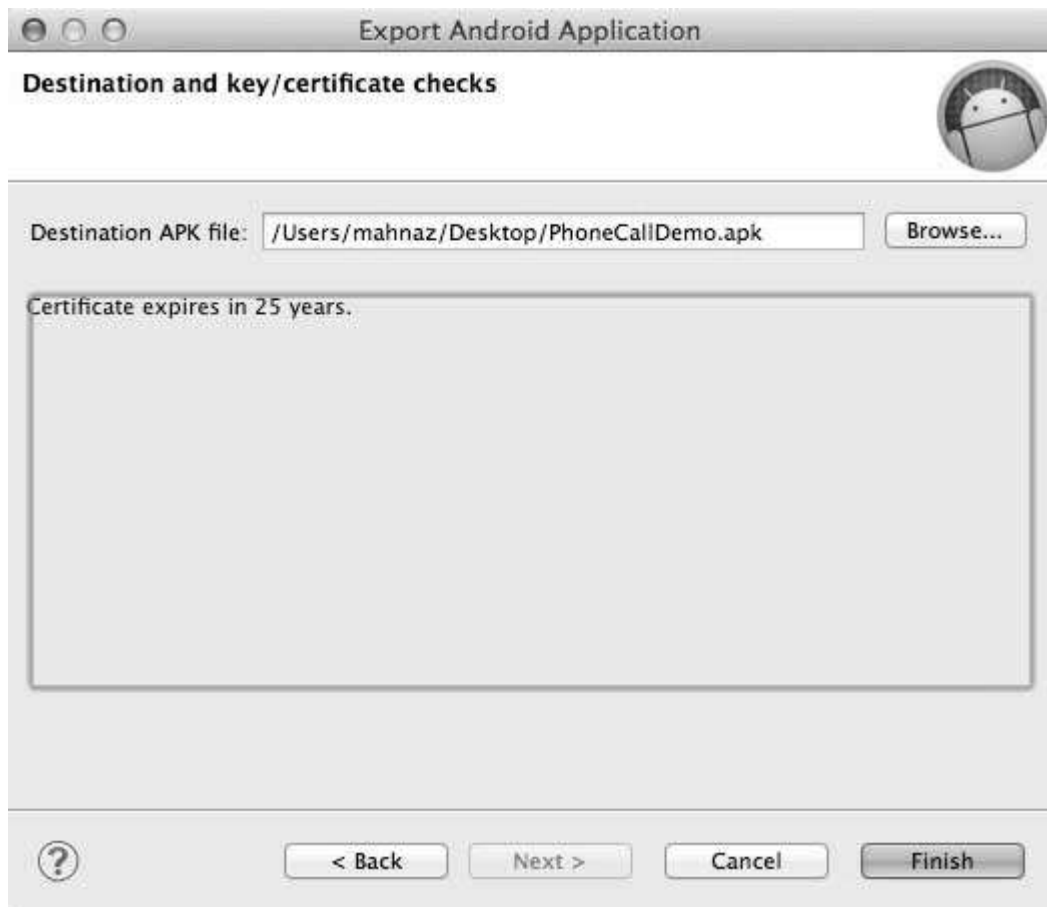


The screenshot shows a window titled "Export Android Application" with a sub-header "Key Creation". A warning icon and text state: "A 25 year certificate validity is recommended." The fields are filled with the following information:

Alias:	PhoneDemo
Password:	*****
Confirm:	*****
Validity (years):	2
First and Last Name:	John Imrahim
Organizational Unit:	Mobile Development
Organization:	tutorialspoint
City or Locality:	Hyderabad
State or Province:	AP
Country Code (XX):	INDIA

At the bottom, there are four buttons: "?", "< Back", "Next >", and "Cancel". The "Next >" button is highlighted in grey, indicating it is the active step.

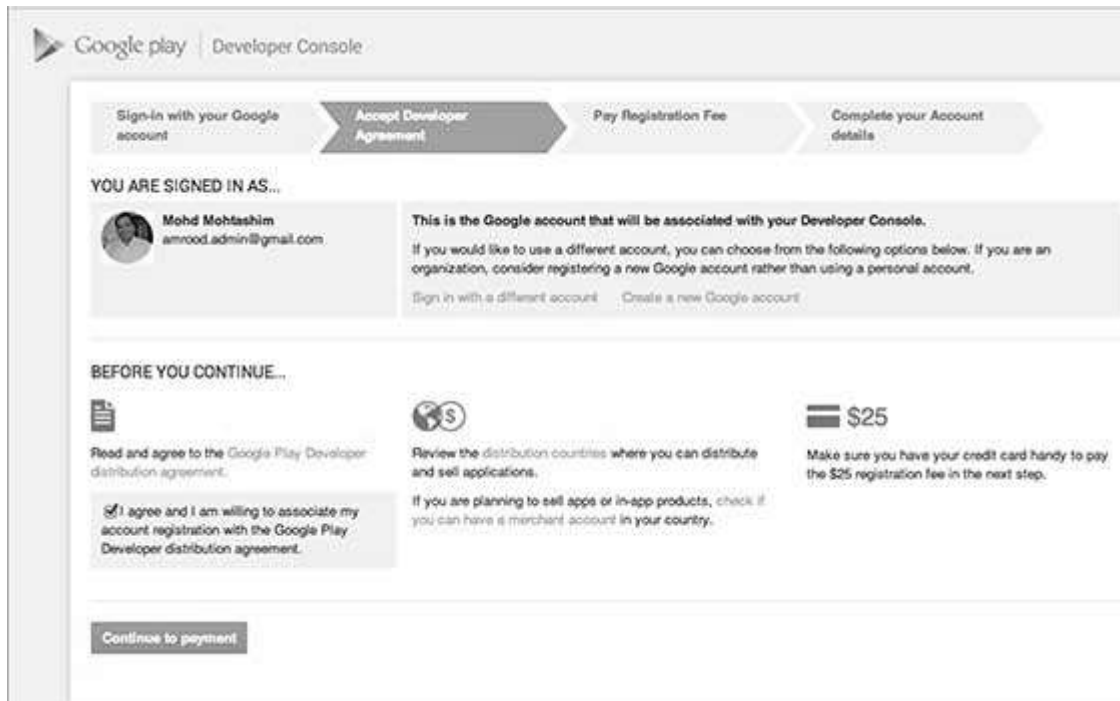
Once you filled up all the information, click **Next** button and finally it will ask you a location where Application will be exported:



Finally, you click on **Finish** button to generate your Android Application Package File which will be uploaded at Google Play marketplace.

Google Play Registration

The most important step is to register with Google Play using [Google Play Marketplace](#). You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace. You will have following screen to accept terms and condition.



You can use **Continue to payment** button to proceed to make a payment of \$25 as a registration fee and finally to complete your account detail.

Once you are a registered user at Google Play, you can upload **release-ready APK** for your application and finally you will complete application detail using application detail page as mentioned in step 9 of the above mentioned checklist.

26. ALERT DIALOG TUTORIAL

Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog.

In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below:

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
```

Now you have to set the positive (yes) or negative (no) button using the object of the AlertDialogBuilder class. Its syntax is-

```
alertDialogBuilder.setPositiveButton(CharSequence text,  
DialogInterface.OnClickListener listener)  
alertDialogBuilder.setNegativeButton(CharSequence text,  
DialogInterface.OnClickListener listener)
```

Apart from this, you can use other functions provided by the builder class to customize the alert dialog. These are listed below:

Sr.No	Method type & description
1	setIcon(Drawable icon) This method set the icon of the alert dialog box.
2	setCancelable(boolean cancelable) This method sets the property that the dialog can be cancelled or not.
3	setMessage(CharSequence message) This method sets the message to be displayed in the alert dialog.
4	setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems, DialogInterface.OnMultiChoiceClickListener listener) This method sets list of items to be displayed in the dialog as the content. The selected option will be notified by the listener.

5	<p>setOnCancelListener(DialogInterface.OnCancelListener onCancelListener)</p> <p>This method Sets the callback that will be called if the dialog is canceled.</p>
6	<p>setTitle(CharSequence title)</p> <p>This method sets the title that will appear in the dialog.</p>

After creating and setting the dialog builder, you will create an alert dialog by calling the create() method of the builder class. Its syntax is:

```
AlertDialog alertDialog = alertDialogBuilder.create();
alertDialog.show();
```

This will create the alert dialog and will show it on the screen.

Example:

The following example demonstrates the use of AlertDialog in android. It uses three different activities to demonstrate it. The dialog asks you to jump to positive activity or negative activity or cancel it.

To experiment with this example, you need to run this on an emulator or an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as AlertDialog under a package com.example.alertdialog. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add alert dialog code to launch the dialog.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Create a new activity called PositiveActivity and confirm it by visiting src/PositiveActivity.java.
5	Modify layout XML file of the newly created activity

	res/layout/activity_positive.xml and add any GUI component if required.
6	Create a new activity called NegativeActivity and confirm it by visiting src/NegativeActivity.java.
7	Modify layout XML file of the newly created activity res/layout/activity_negative.xml and add any GUI component if required.
8	Modify res/values/strings.xml to define required constant values.
9	Run the application and choose a running android device and install the application on it and verify the results.

Here is the modified code of src/com.example.alertdialog/MainActivity.java

```
package com.example.alertdialog;

import com.example.alertdialog.*;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

```

public void open(View view){
    AlertDialog.Builder alertDialogBuilder = new
    AlertDialog.Builder(this);
    alertDialogBuilder.setMessage(R.string.decision);
    alertDialogBuilder.setPositiveButton(R.string.positive_button,
    new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface arg0, int arg1) {
            Intent                positiveActivity                =                new
            Intent(getApplicationContext(),com.example.alertdialog.Positi
            veActivity.c                lass);
            startActivity(positiveActivity);

        }
    });
    alertDialogBuilder.setNegativeButton(R.string.negative_button,
    new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface dialog, int which) {
            Intent                negativeActivity                =                new
            Intent(getApplicationContext(),com.example.alertdialog.Negati
            veActivity.c                lass);
            startActivity(negativeActivity);

        }
    });

    AlertDialog alertDialog = alertDialogBuilder.create();
    alertDialog.show();

}

@Override

```

```
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar  
    // if it is present.  
    getMenuInflater().inflate(R.menu.main, menu);  
    return true;  
}  
}
```

Here is the default code of `src/com.example.alertdialog/PositiveActivity.java`

```
package com.example.alertdialog;  
  
import android.os.Bundle;  
import android.app.Activity;  
import android.view.Menu;  
  
public class PositiveActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_positive);  
    }  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        // Inflate the menu; this adds items to the action bar  
        // if it is present.  
        getMenuInflater().inflate(R.menu.positive, menu);  
        return true;  
    }  
}
```

Here is the default code of `src/com.example.alertdialog/NegativeActivity.java`

```
package com.example.alertdialog;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class NegativeActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_negative);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.negative, menu);
        return true;
    }
}
```

Here is the modified code of `res/layout/activity_main.xml`

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
```



```

        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".MainActivity" >

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="170dp"
            android:onClick="open"
            android:text="@string/hello_world" />

    </RelativeLayout>

```

Here is the modified code of **res/layout/activity_positive.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".PositiveActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="14dp"

```

```

    android:layout_marginTop="20dp"
    android:text="@string/positive"
    android:textAppearance="?android:attr/textAppearanceLarge" />

```

```
</RelativeLayout>
```

Here is the modified code of **res/layout/activity_negative.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".NegativeActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="14dp"
        android:layout_marginTop="17dp"
        android:text="@string/negative"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Here is the modified code of **Strings.xml**

```

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

```

```

<string name="app_name">AlertDialog</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="title_activity_positive">PositiveActivity</string>
<string name="title_activity_negative">NegativeActivity</string>
<string name="positive">Positive Activity</string>
<string name="negative">Negative Activity</string>
<string name="decision">Are you sure, you wanted to make this
decision</string>
<string name="positive_button">+ive</string>
<string name="negative_button">-ive</string>

</resources>

```

Here is the default code of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.alertdialog"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.alertdialog.MainActivity"
            android:label="@string/app_name" >

```

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity
    android:name="com.example.alertdialog.PositiveActivity"
    android:label="@string/title_activity_positive" >
</activity>
<activity
    android:name="com.example.alertdialog.NegativeActivity"
    android:label="@string/title_activity_negative" >
</activity>
</application>


</manifest>
```

Let's try to run your Camera application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.

Android Device Chooser

Select a device with min API level 8.

Choose a running Android device

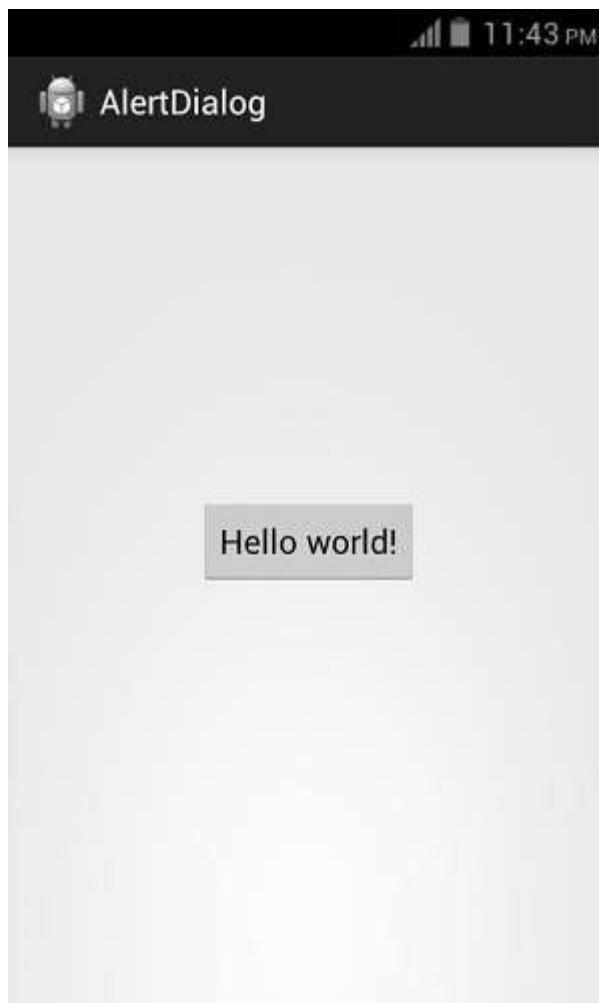
Serial Number	AVD Name	Target	Debug	State
 qmobile-a6-DDF787312B26590E	N/A	✓ 4.0.4	Yes	Online

Launch a new Android Virtual Device

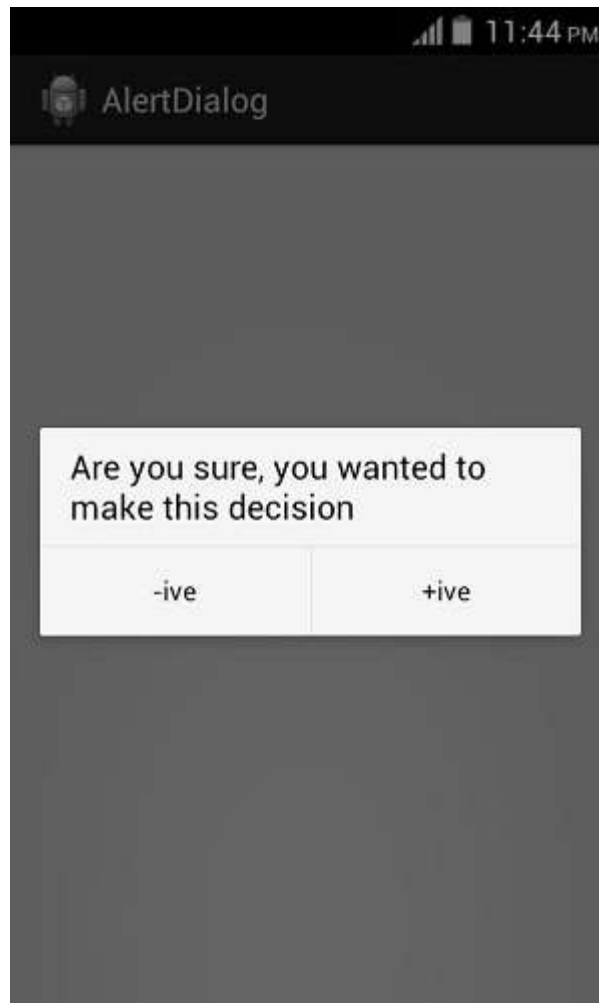
AVD Name	Target Name	Platform	API Level	CPU/ABI
--	No AVD available	--	--	

Use same device for future launches

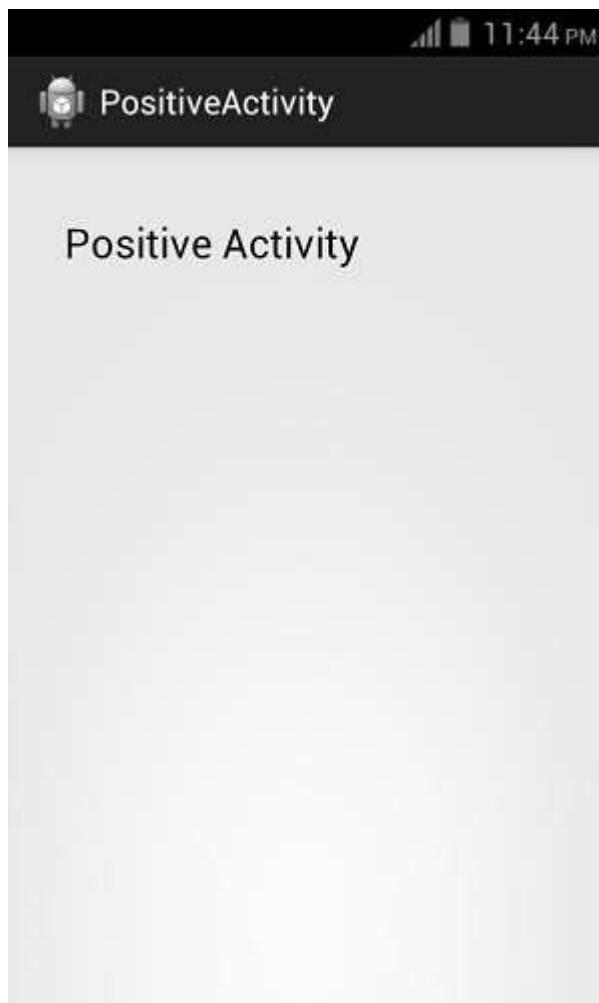
Select your mobile device as an option and then check your mobile device which will display following screen:



Now just tap the button hello world to see the alert box, which would be something like this:



Now select any of the two buttons and see the respective activity loading up. In case you select positive button, this screen would appear:



Now press back button on your device, and this time select negative from your alert dialog. The following screen would appear this time:



27. ANIMATIONS

Animation in android is possible in many ways. We will discuss one easy and widely used way of making animation called tweened animation.

Tween Animation

Tween Animation takes some parameters such as start value, end value, size, time duration, rotation angle etc., and perform the required animation on that object. It can be applied to any type of object. So in order to use this, android has provided us a class called Animation.

In order to perform animation in android, we are going to call a static function `loadAnimation()` of the class `AnimationUtils`. We are going to receive the result in an instance of Animation Object. Its syntax is as follows:

```
Animation animation =  
AnimationUtils.loadAnimation(getApplicationContext(),  
R.anim.myanimation);
```

Note the second parameter. It is the name of the our animation xml file. You have to create a new folder called **anim** under res directory and make an xml file under anim folder.

This animation class has many useful functions which are listed below:

Sr.No	Method & Description
1	start() This method starts the animation.
2	setDuration(long duration) This method sets the duration of an animation.
3	getDuration() This method gets the duration which is set by above method.
4	end() This method ends the animation.

5	<p>cancel()</p> <p>This method cancels the animation.</p>
---	--

In order to apply this animation to an object, we will just call the `startAnimation()` method of the object. Its syntax is:

```
ImageView image1 = (ImageView)findViewById(R.id.imageView1);
image.startAnimation(animation);
```

Zoom in animation

In order to perform a zoom in animation, create an XML file under `anim` folder under `res` directory and put this code in the file.

```
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <scale xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromXScale="0.5"
        android:toXScale="3.0"
        android:fromYScale="0.5"
        android:toYScale="3.0"
        android:duration="5000"
        android:pivotX="50%"
        android:pivotY="50%" >

    </scale>

</set>
```

The parameter **fromXScale** and **fromYScale** defines the start point and the parameters **toXScale** and **toYScale** defines the end point. The **duration** defines the time of animation and the **pivotX,pivotY** defines the center from where the animation would start.

Example:

The following example demonstrates the use of Animation in android. You would be able to choose different type of animation from the menu and the selected animation will be applied on an `imageView` on the screen.

To experiment with this example, you need to run this on an emulator or an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Animation under a package com.example.animation. While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add animation code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Create a new folder under res directory and call it anim. Confirm it by visiting res/anim.
5	Right click on anim and click on new and select Android XML file You have to create three different files that are listed below.
6	Create files myanimation.xml, clockwise.xml, fade.xml and add the XML code.
7	Modify res/values/string.xml file and add necessary string components.
8	Modify res/menu/main.xml file and add necessary menu components.
9	Run the application and choose a running android device and install the application on it and verify the results.

Here is the modified code of src/com.example.animation/MainActivity.java.

```
package com.example.animation;

import com.example.animation.R;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item)
    {
        super.onOptionsItemSelected(item);
        switch(item.getItemId())
        {
            case R.id.zoomInOut:
                ImageView image = (ImageView)findViewById(R.id.imageView1);
                Animation animation =
                AnimationUtils.loadAnimation(getApplicationContext(),
                R.anim.myanimation);
                image.startAnimation(animation);
                return true;
            case R.id.rotate360:
```

```

        ImageView image1 = (ImageView)findViewById(R.id.imageView1);
        Animation animation1 =
        AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.clockwise);
        image1.startAnimation(animation1);
        return true;
    case R.id.fadeInOut:
        ImageView image2 = (ImageView)findViewById(R.id.imageView1);
        Animation animation2 =
        AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.fade);
        image2.startAnimation(animation2);
        return true;
    }
    return false;
}
}
}

```

Here is the modified code of **res/layout/activity_main.xml**.

```

<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"
    android:gravity="top"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageView

```

```

        android:id="@+id/imageView1"
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="179dp"
        android:src="@drawable/ic_launcher" />

</RelativeLayout>

```

Here is the code of **res/anim/myanimation.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <scale xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromXScale="0.5"
        android:toXScale="3.0"
        android:fromYScale="0.5"
        android:toYScale="3.0"
        android:duration="5000"
        android:pivotX="50%"
        android:pivotY="50%" >

</scale>

    <scale xmlns:android="http://schemas.android.com/apk/res/android"
        android:startOffset="5000"
        android:fromXScale="3.0"
        android:toXScale="0.5"
        android:fromYScale="3.0"
        android:toYScale="0.5"
        android:duration="5000"
        android:pivotX="50%"

```

```

        android:pivotY="50%" >

</scale>

</set>

```

Here is the code of **res/anim/clockwise.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <rotate xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" >

</rotate>

    <rotate xmlns:android="http://schemas.android.com/apk/res/android"
        android:startOffset="5000"
        android:fromDegrees="360"
        android:toDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" >

</rotate>

</set>

```

Here is the code of **res/anim/fade.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"

```



```

android:interpolator="@android:anim/accelerate_interpolator" >

<alpha
    android:fromAlpha="0"
    android:toAlpha="1"
    android:duration="2000" >

</alpha>

<alpha
    android:startOffset="2000"
    android:fromAlpha="1"
    android:toAlpha="0"
    android:duration="2000" >

</alpha>

</set>

```

Here is the modified code of **res/menu/main.xml**.

```

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

    <item
        android:id="@+id/rotate360"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/rotate_String"/>

    <item
        android:id="@+id/zoomInOut"
        android:orderInCategory="100"
        android:title="@string/zoom_In_Out"/>

    <item

```

```

    android:id="@+id/fadeInOut"
    android:orderInCategory="100"
    android:title="@string/fade_String" />

```

```
</menu>
```

Here is the modified code of **res/values/string.xml**.

```

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

    <string name="app_name">Animation</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="zoom_In_Out">Zoom In/Out</string>
    <string name="rotate_String">Clockwise/Anti Clockwise</string>
    <string name="fade_String">Fade In/Out</string>

</resources>

```

Here is the default code of **AndroidManifest.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.animation"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"

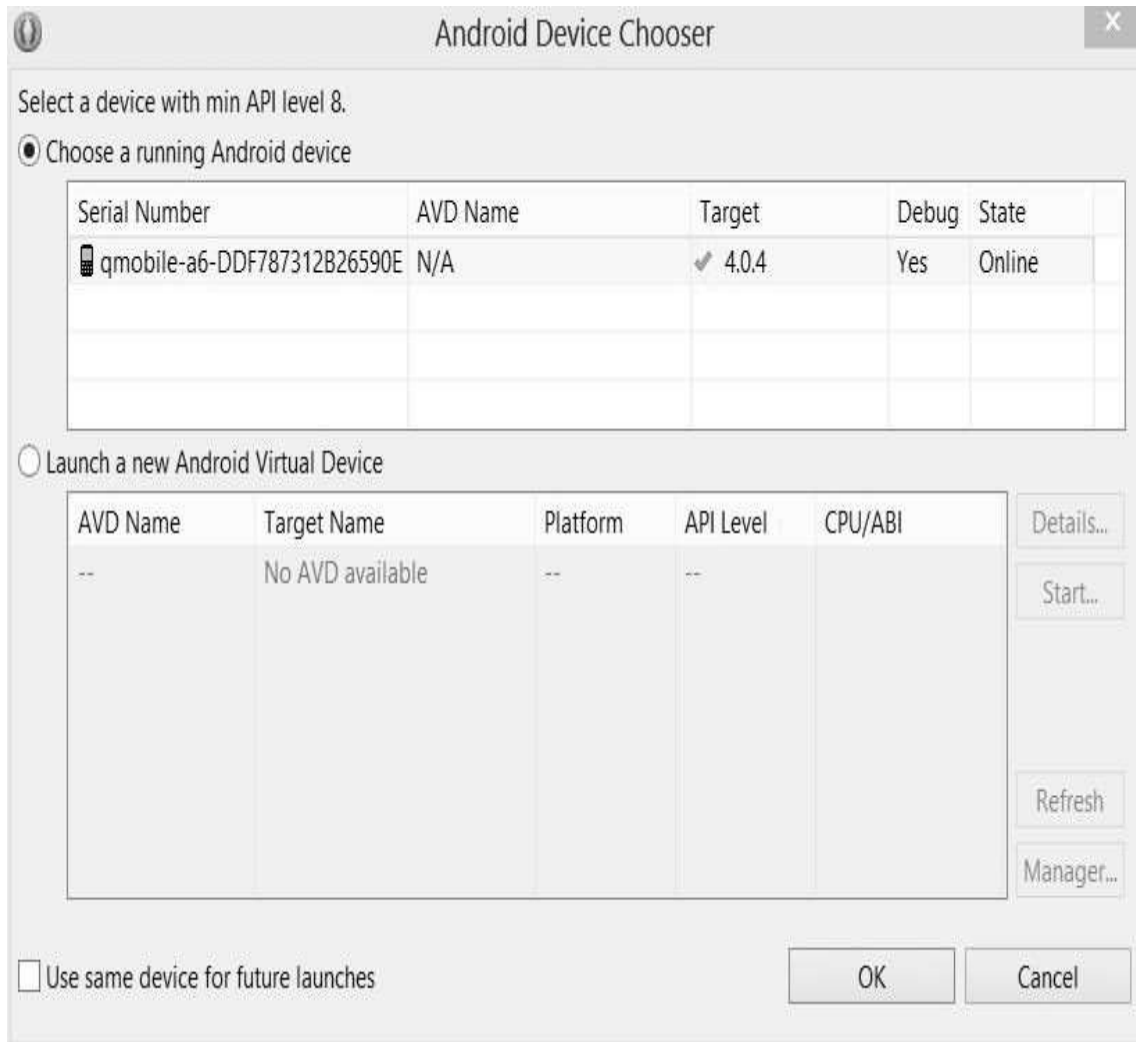
```

```
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.animation.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

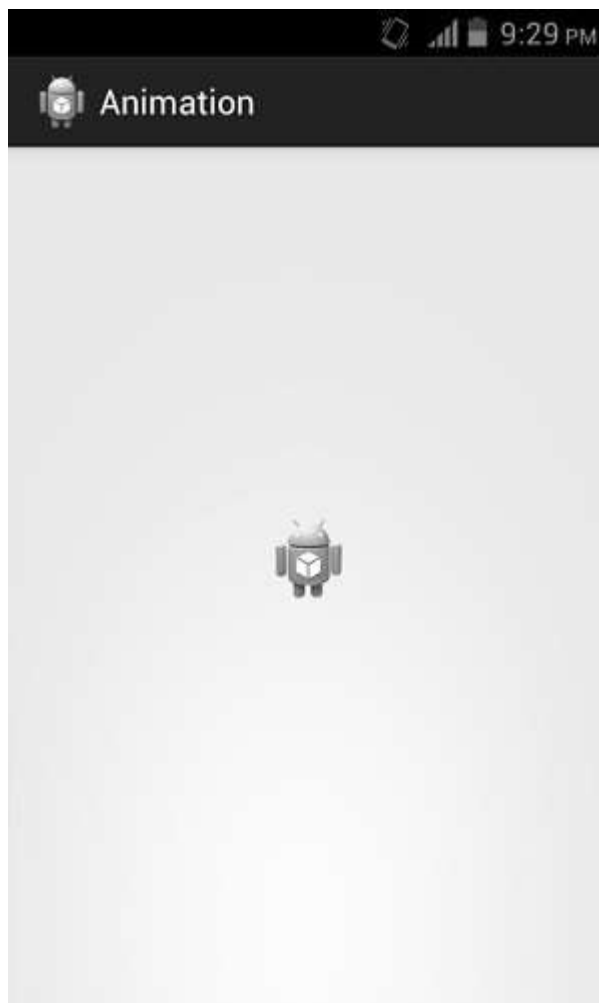
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

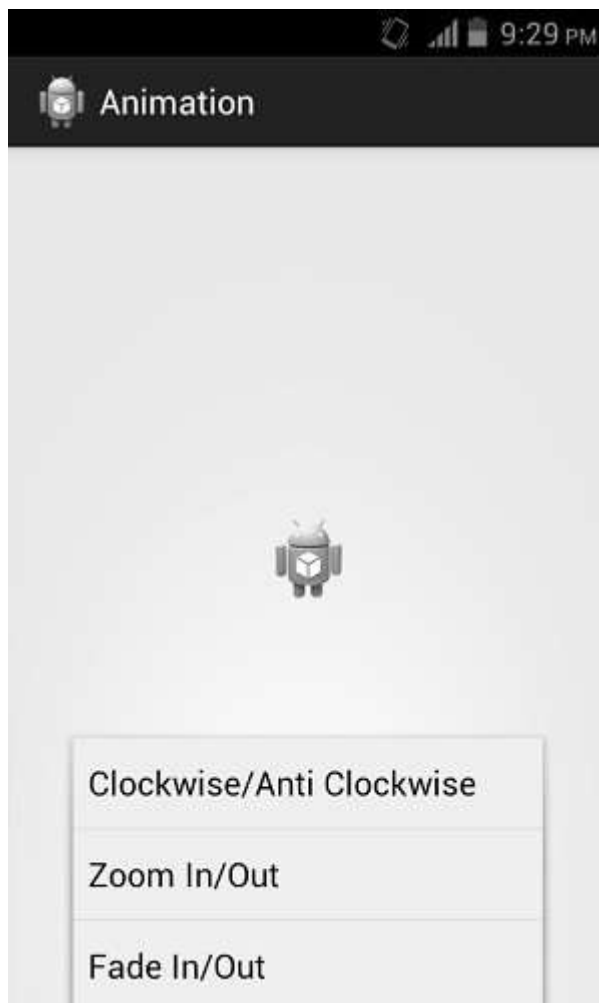
Let's try to run your Animation application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



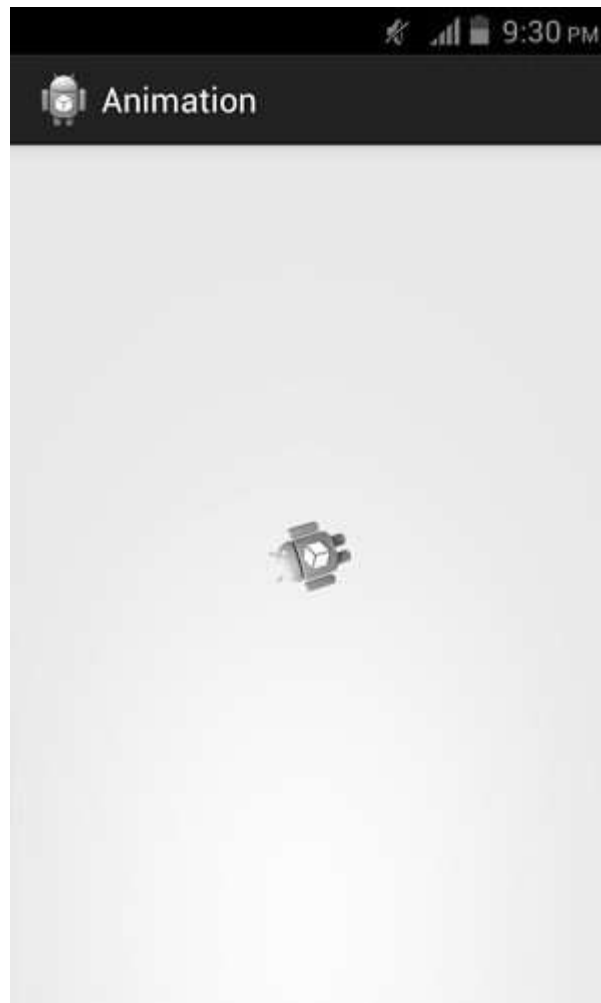
Now just select the menu from your mobile, and a menu would appear which would be something like this:



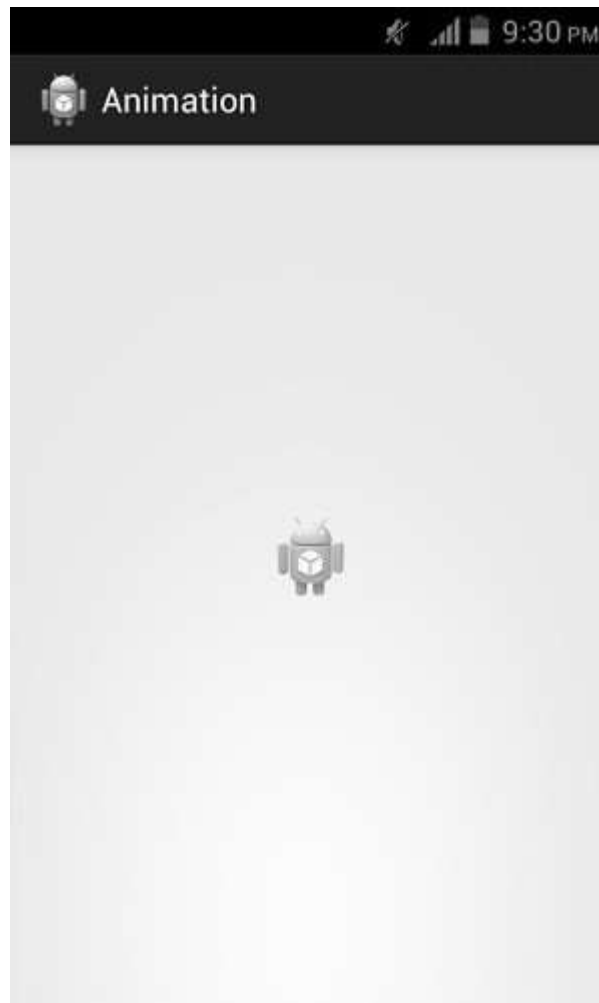
Now just select the Zoom in, Zoom out option from menu and an animation would appear which would be something like this:



Now just select the clockwise option from menu and an animation would appear which would be something like this:



Now just select the fade in/out option from menu and an animation would appear which would be something like this:



Note: If you run it in emulator, you may not experience smooth animation effect. You have to run it in your android mobile in order to experience the smooth animation.

28. AUDIO CAPTURE

Android has a built-in microphone through which you can capture audio and store it, or play it in your phone. There are many ways to do that but the most common way is through MediaRecorder class.

Android provides MediaRecorder class to record audio or video. To use MediaRecorder class, you will first create an instance of MediaRecorder class. Its syntax is given below.

```
MediaRecorder myAudioRecorder = new MediaRecorder();
```

Now you will set the source, output and encoding format and output file. Their syntax is given below.

```
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);  
myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);  
myAudioRecorder.setOutputFile(outputFile);
```

After specifying the audio source and format and its output file, we can then call the two basic methods prepare and start to start recording the audio.

```
myAudioRecorder.prepare();  
myAudioRecorder.start();
```

Apart from these methods, there are other methods listed in the MediaRecorder class that allows you more control over audio and video recording.

Sr.No	Method & description
1	setAudioSource() This method specifies the source of audio to be recorded.
2	setVideoSource() This method specifies the source of video to be recorded.
3	setOutputFormat() This method specifies the audio format in which audio to be stored.

4	setAudioEncoder() This method specifies the audio encoder to be used.
5	setOutputFile() This method configures the path to the file into which the recorded audio is to be stored.
6	stop() This method stops the recording process.
7	release() This method should be called when the recorder instance is needed.

Example:

This example provides demonstration of MediaRecorder class to capture audio and then MediaPlayer class to play that recorded audio.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as AudioCapture under a package com.example.audiocapture. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add AudioCapture code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Modify AndroidManifest.xml to add necessary permissions.
6	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.audiocapture/MainActivity.java**

```
package com.example.audiocapture;

import java.io.File;
import java.io.IOException;

import android.media.MediaPlayer;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.os.Environment;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

    private MediaRecorder myAudioRecorder;
    private String outputFile = null;
    private Button start,stop,play;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        start = (Button)findViewById(R.id.button1);
        stop = (Button)findViewById(R.id.button2);
        play = (Button)findViewById(R.id.button3);

        stop.setEnabled(false);
        play.setEnabled(false);
        outputFile = Environment.getExternalStorageDirectory().
            getAbsolutePath() + "/myrecording.3gp";;
```

```
myAudioRecorder = new MediaRecorder();
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);

myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
myAudioRecorder.setOutputFile(outputFile);

}

public void start(View view){
    try {
        myAudioRecorder.prepare();
        myAudioRecorder.start();
    } catch (IllegalStateException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    start.setEnabled(false);
    stop.setEnabled(true);
    Toast.makeText(getApplicationContext(), "Recording started",
    Toast.LENGTH_LONG).show();

}

public void stop(View view){
    myAudioRecorder.stop();
    myAudioRecorder.release();
    myAudioRecorder = null;
    stop.setEnabled(false);
    play.setEnabled(true);
}
```

```

        Toast.makeText(getApplicationContext(), "Audio recorded
        successfully",
        Toast.LENGTH_LONG).show();
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    public void play(View view) throws IllegalArgumentException,
    SecurityException, IllegalStateException, IOException{

    MediaPlayer m = new MediaPlayer();
    m.setDataSource(outputFile);
    m.prepare();
    m.start();
    Toast.makeText(getApplicationContext(), "Playing audio",
    Toast.LENGTH_LONG).show();

    }

}

```

Here is the content of **activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

```

```
tools:context=".MainActivity" >

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="32dp"
    android:text="@string/Recording"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<ImageView
    android:id="@+id/imageView1"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="37dp"
    android:scaleType="fitXY"
    android:src="@android:drawable/presence_audio_online" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/imageView1"
    android:layout_marginTop="67dp"
    android:layout_toLeftOf="@+id/imageView1"
    android:onClick="start"
    android:text="@string/start" />

<Button
```

```

        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/button1"
        android:layout_alignBottom="@+id/button1"
        android:layout_alignRight="@+id/textView1"
        android:layout_marginRight="40dp"
        android:onClick="stop"
        android:text="@string/stop" />

<Button
    android:id="@+id/button3"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button2"
    android:layout_centerHorizontal="true"
    android:onClick="play"
    android:text="@string/play" />

</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">AudioCapture</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Recording">Android Audio Recording Application</string>
    <string name="start">start</string>
    <string name="stop">stop</string>
    <string name="play">play</string>

```



```
</resources>
```

Here is the content of **AndroidManifest.xml**.


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.audiocapture"
    android:versionCode="1"
    android:versionName="1.0" >

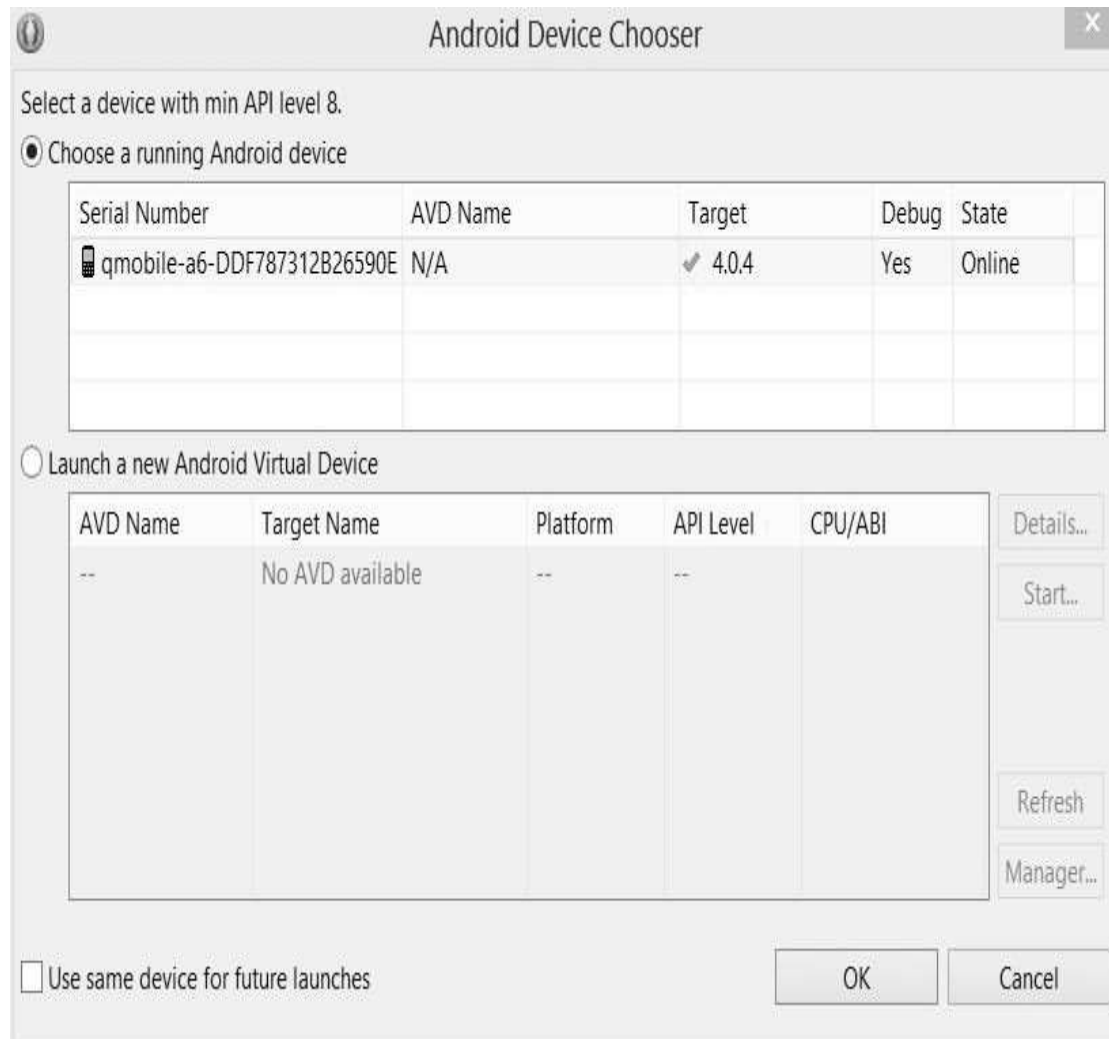
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="17" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.audiocapture.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

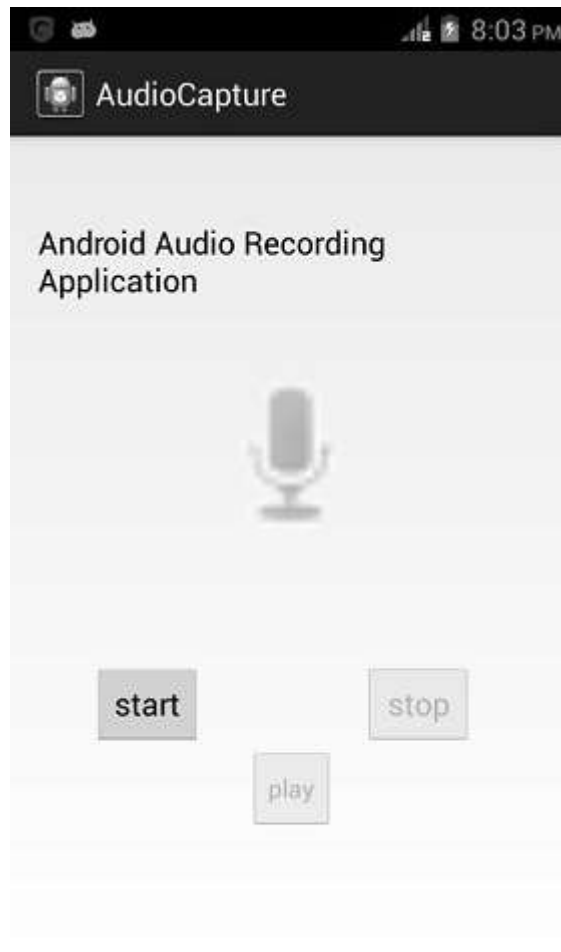
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

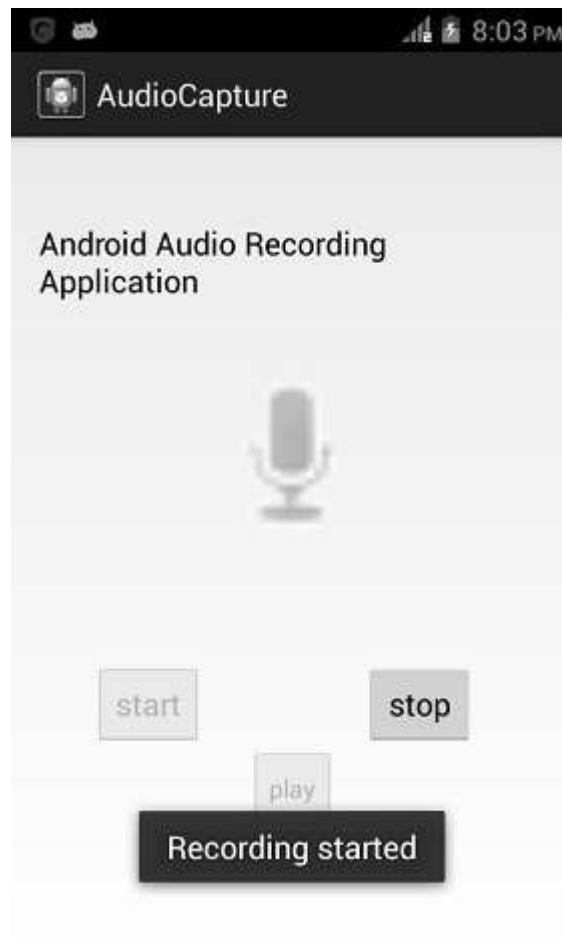
Let's try to run your AndroidCapture application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



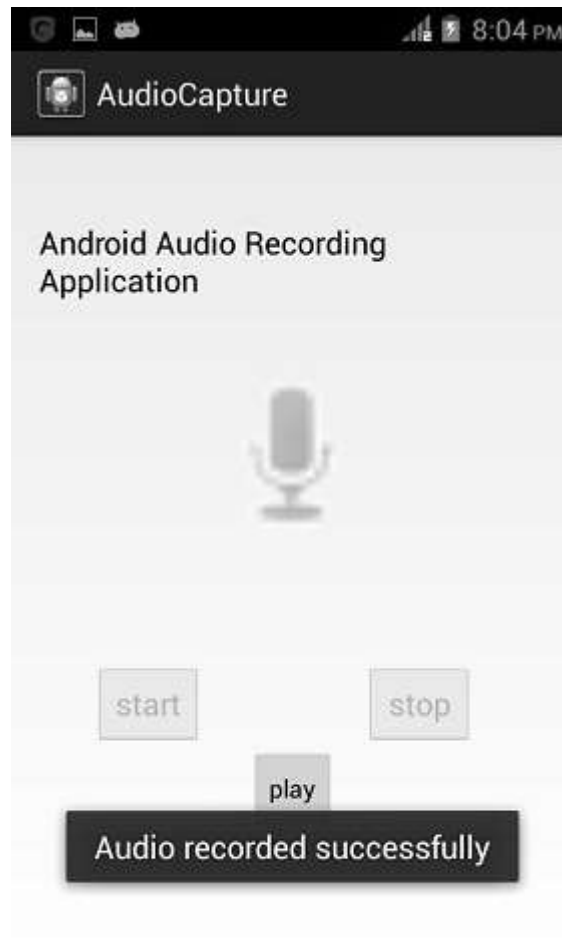
Select your mobile device as an option and then check your mobile device which will display following screen:



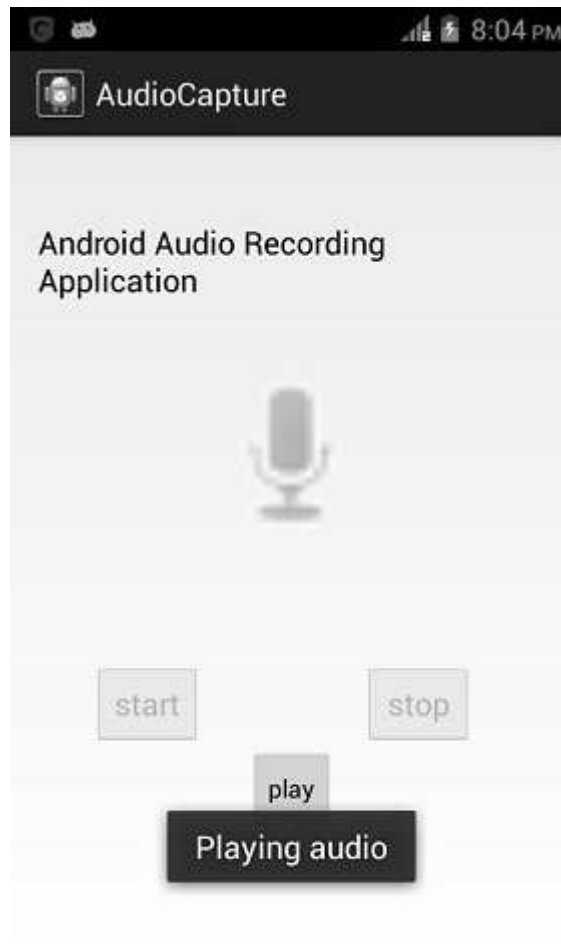
Now by default you will see that the **stop** and **play** buttons are disabled. Just press the start button and your application will start recording the audio. It will display the following screen:



Now just press **stop** button and it will save the recorded audio to external sd card. When you click on stop button, the following screen would appear:



Now just press the play button and recorded audio will just start playing on the device. The following message appears when you click on **play** button:



29. AUDIO MANAGER

You can easily control your ringer volume and ringer profile i.e. (silent, vibrate, loud etc.) in android. Android provides AudioManager class that provides access to these controls.

In order to use AudioManager class, you have to first create an object of AudioManager class by calling the **getSystemService()** method. Its syntax is given below.

```
private AudioManager myAudioManager;  
myAudioManager = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
```

Once you instantiate the object of AudioManager class, you can use **setRingerMode** method to set the audio or ringer profile of your device. Its syntax is given below.

```
myAudioManager.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
```

The method setRingerMode takes an integer number as a parameter. For each mode, an integer number is assigned that will differentiate between different modes. The possible modes are.

Sr.No	Mode & Description
1	RINGER_MODE_VIBRATE This Mode sets the device at vibrate mode.
2	RINGER_MODE_NORMAL This Mode sets the device at normal (loud) mode.
3	RINGER_MODE_SILENT This Mode sets the device at silent mode.

Once you have set the mode, you can call the **getRingerMode()** method to get the set state of the system. Its syntax is given below.

```
int mod = myAudioManager.getRingerMode();
```

Apart from the `getRingerMode` method, there are other methods available in the `AudioManager` class to control the volume and other modes. They are listed below:

Sr.No	Method & description
1	adjustVolume(int direction, int flags) This method adjusts the volume of the most relevant stream.
2	getMode() This method returns the current audio mode.
3	getStreamMaxVolume(int streamType) This method returns the maximum volume index for a particular stream.
4	getStreamVolume(int streamType) This method returns the current volume index for a particular stream.
5	isMusicActive() This method checks whether any music is active.
6	startBluetoothSco() This method Starts bluetooth SCO audio connection.
7	stopBluetoothSco() This method stops bluetooth SCO audio connection.

Example:

The below example demonstrates the use of `AudioManager` class. It creates a basic application that allows you to set different ringer modes for your device.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as <code>AudioManager</code> under a package <code>com.example.audiomanager</code> . While

	creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add AudioManager code.
3	Modify layout XML file res/layout/activity_main.xml to add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Modify AndroidManifest.xml to add necessary permissions.
6	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.audiomanager/MainActivity.java**

```

package com.example.audiomanager;

import android.media.AudioManager;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {

    private Button Vibrate, Ring, Silent, Mode;
    private TextView Status;
    private AudioManager myAudioManager;
    @Override
    protected void onCreate(Bundle savedInstanceState) {

```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

Vibrate = (Button)findViewById(R.id.button2);
Ring = (Button)findViewById(R.id.button4);
Silent = (Button)findViewById(R.id.button3);
Mode = (Button)findViewById(R.id.button1);
Status = (TextView)findViewById(R.id.textView2);

myAudioManager =
(AudioManager)getSystemService(Context.AUDIO_SERVICE);

}

public void vibrate(View view){
    myAudioManager.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
}
public void ring(View view){
    myAudioManager.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
}
public void silent(View view){
    myAudioManager.setRingerMode(AudioManager.RINGER_MODE_SILENT);
}
public void mode(View view){
    int mod = myAudioManager.getRingerMode();
    if(mod == AudioManager.RINGER_MODE_NORMAL){
        Status.setText("Current Status: Ring");
    }
    else if(mod == AudioManager.RINGER_MODE_SILENT){
        Status.setText("Current Status: Silent");
    }
    else if(mod == AudioManager.RINGER_MODE_VIBRATE){
        Status.setText("Current Status: Vibrate");
    }
}
```

```

    }
    else{

    }
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"

```

```
android:text="@string/audio"  
android:textAppearance="?android:attr/textAppearanceLarge" />
```

```
<Button  
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBaseline="@+id/button3"  
    android:layout_alignBottom="@+id/button3"  
    android:layout_alignRight="@+id/textView1"  
    android:onClick="vibrate"  
    android:text="@string/Vibrate" />
```

```
<Button  
    android:id="@+id/button3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentBottom="true"  
    android:layout_alignParentLeft="true"  
    android:layout_marginBottom="144dp"  
    android:layout_marginLeft="40dp"  
    android:layout_toLeftOf="@+id/button2"  
    android:onClick="silent"  
    android:text="@string/Silent" />
```

```
<Button  
    android:id="@+id/button4"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBaseline="@+id/button1"  
    android:layout_alignBottom="@+id/button1"  
    android:layout_toRightOf="@+id/button1"  
    android:onClick="ring"  
    android:text="@string/Ring" />
```

```

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/button2"
    android:layout_alignLeft="@+id/button3"
    android:layout_marginBottom="15dp"
    android:onClick="mode"
    android:text="@string/Mode" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="46dp"
    android:text="@string/Status"
    android:textAppearance="?android:attr/textAppearanceMedium" />

</RelativeLayout>

```

Here is the content of **Strings.xml**

```

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

    <string name="app_name">AudioManager</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="audio">Set Audio Profiles</string>
    <string name="Ring">Ring</string>
    <string name="Vibrate">Vibrate</string>
    <string name="Silent">Silent</string>

```

```

<string name="Mode">Current Mode</string>
<string name="Status">Current Status</string>

</resources>

```

Here is the content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.audiomanager"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

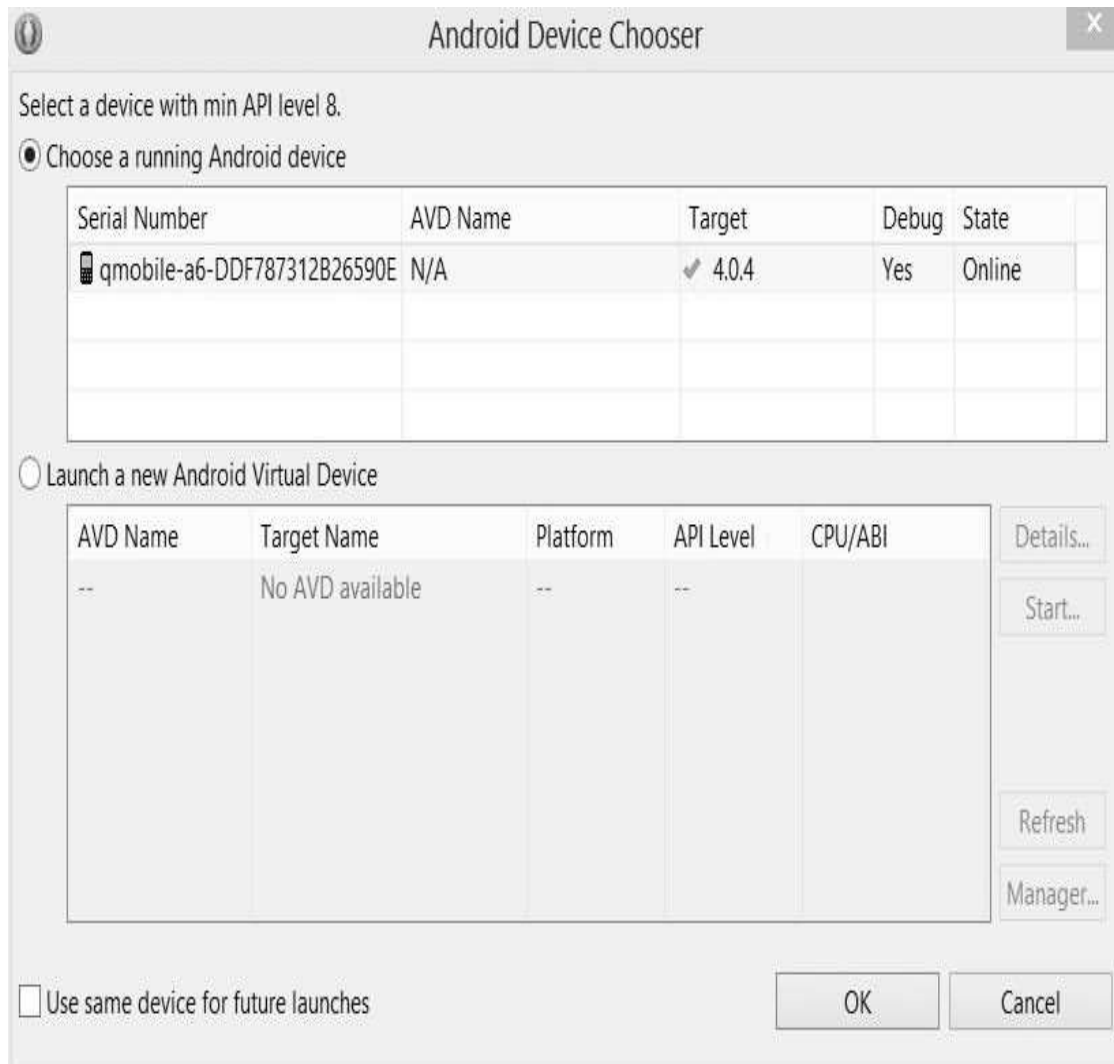
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.audiomanager.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

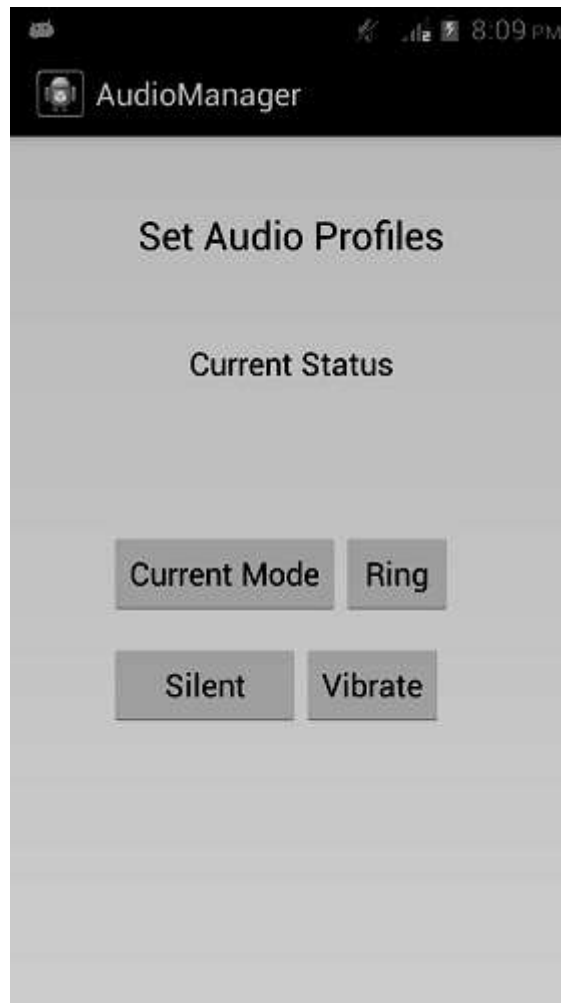
</manifest>

```

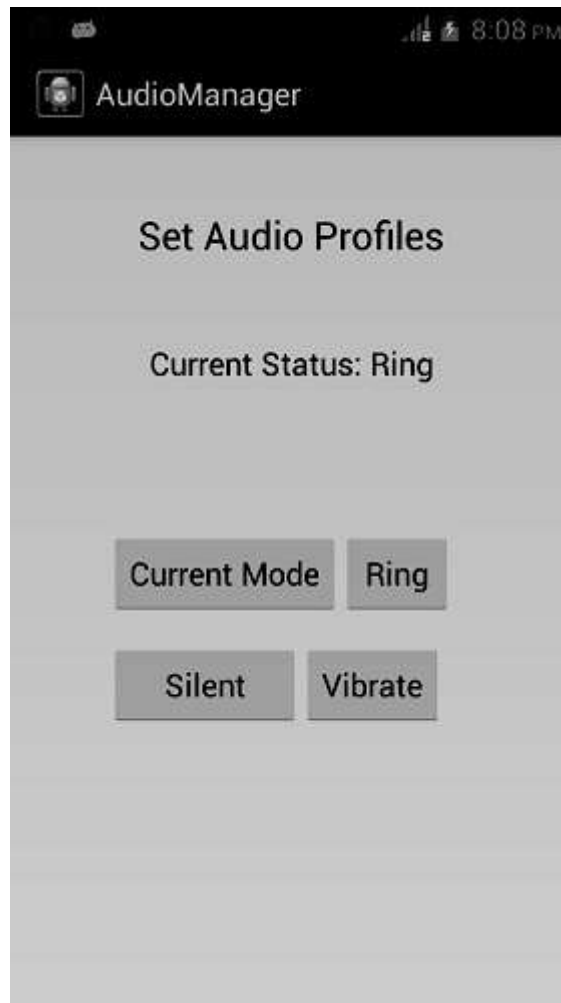
Let's try to run your Androidmanager application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



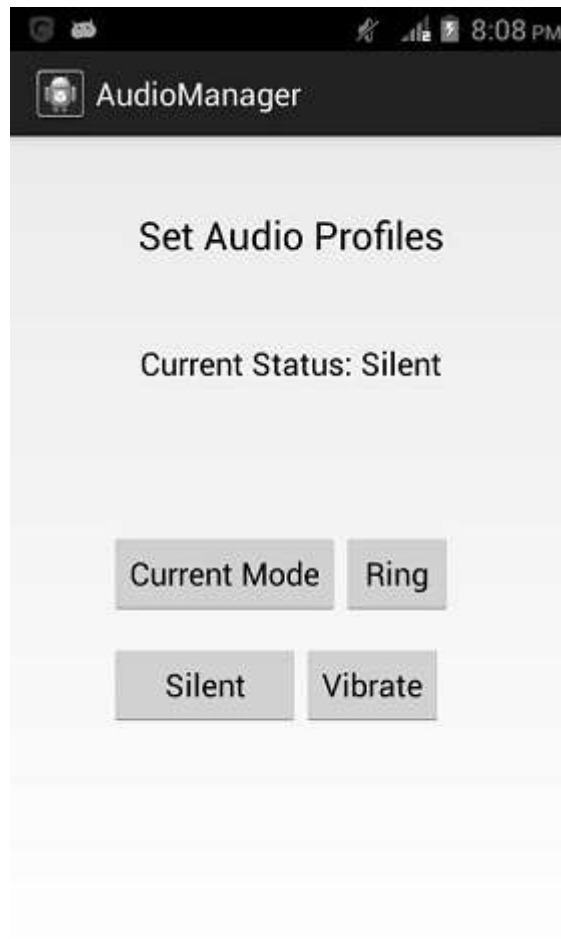
Select your mobile device as an option and then check your mobile device which will display following screen:



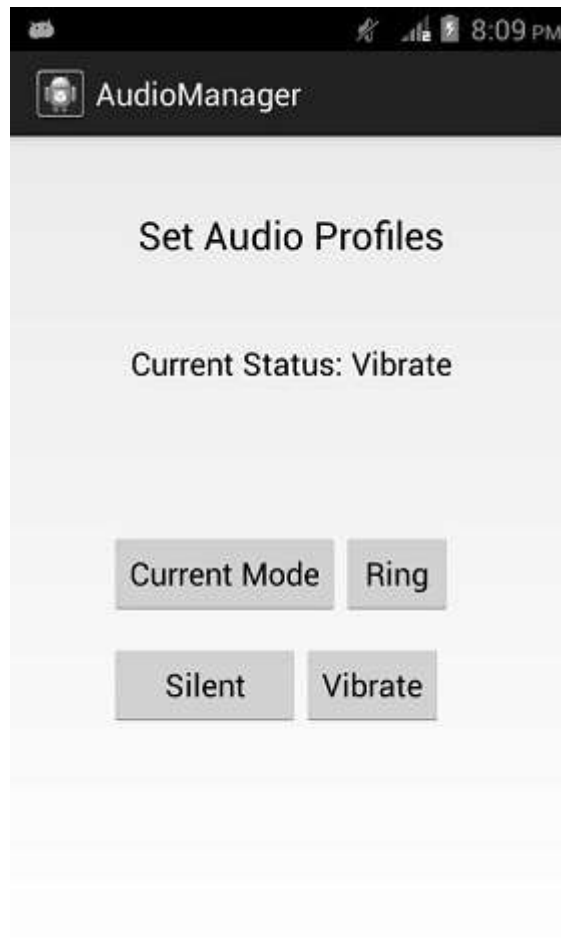
Now just select the **ring** button and then press the **current mode** button to see that if it's status has been set.



Now press the **silent** button and then press the **current mode** button to see that if it is set or not. It will display the following screen:



Now press the **vibrate** button and then press the **current mode** button to see that if it is set or not. It will display the following screen:



30. AUTOCOMPLETE

If you want to get suggestions, when you type in an editable text field, you can do this via `AutoCompleteTextView`. It provides suggestions automatically when the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box.

In order to use `AutoCompleteTextView` you have to first create an `AutoCompleteTextView` Field in the xml. Its syntax is given below.

```
<AutoCompleteTextView
    android:id="@+id/autoCompleteTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="65dp"
    android:ems="10" >
```

After that, you have to get a reference of this textview in java. Its syntax is given below.

```
private AutoCompleteTextView actv;
actv = (AutoCompleteTextView) findViewById(R.id.autoCompleteTextView1);
```

The next thing you need to do is to specify the list of suggestion items to be displayed. You can specify the list items as a string array in java or in strings.xml. Its syntax is given below.

```
String[] countries = getResources().
getStringArray(R.array.list_of_countries);
ArrayAdapter adapter = new ArrayAdapter
(this, android.R.layout.simple_list_item_1, countries);
actv.setAdapter(adapter);
```

The array adapter class is responsible for displaying the data as list in the suggestion box of the text field. The **setAdapter** method is used to set the adapter of the `autoCompleteTextView`. Apart from these methods, the other methods of `AutoComplete` are listed below.

Sr.No	Method & description
1	getAdapter() This method returns a filterable list adapter used for auto completion.
2	getCompletionHint() This method returns optional hint text displayed at the bottom of the matching list.
3	getDropDownAnchor() This method returns the id for the view that the auto-complete drop down list is anchored to.
4	getListSelection() This method returns the position of the dropdown view selection, if there is one.
5	isPopupShowing() This method indicates whether the popup menu is showing.
6	setText(CharSequence text, boolean filter) This method sets text except that it can disable filtering.
7	showDropDown() This method displays the drop down on screen.

Example:

The below example demonstrates the use of AutoCompleteTextView class. It creates a basic application that allows you to type in and it displays suggestions on your device.

To experiment with this example, you need to run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it

	as AutoComplete under a package com.example.autocomplete. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add AutoCompleteTextView code.
3	Modify layout XML file res/layout/activity_main.xml to add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Modify AndroidManifest.xml to add necessary permissions.
6	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.autocomplete/MainActivity.java**

```
package com.example.autocomplete;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.AdapterView;
import android.widget.AutoCompleteTextView;
import android.widget.MultiAutoCompleteTextView;

public class MainActivity extends Activity {

    private AutoCompleteTextView actv;
    private MultiAutoCompleteTextView mactv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

        setContentView(R.layout.activity_main);

String[] countries = getResources().
getStringArray(R.array.list_of_countries);
ArrayAdapter adapter = new ArrayAdapter
(this,android.R.layout.simple_list_item_1,countries);

actv = (AutoCompleteTextView)
findViewById(R.id.autoCompleteTextView1);
mactv = (MultiAutoCompleteTextView) findViewById
(R.id.multiAutoCompleteTextView1);

actv.setAdapter(adapter);
mactv.setAdapter(adapter);

mactv.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Here is the content of **activity_main.xml**

```

<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"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity" >

<AutoCompleteTextView
    android:id="@+id/autoCompleteTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="65dp"
    android:ems="10" >

    <requestFocus />
</AutoCompleteTextView>

<MultiAutoCompleteTextView
    android:id="@+id/multiAutoCompleteTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/autoCompleteTextView1"
    android:layout_centerVertical="true"
    android:ems="10" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"

    android:layout_height="wrap_content"
```



```

        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="@string/auto_complete"
        android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/multiAutoCompleteTextView1"
    android:layout_alignParentLeft="true"
    android:layout_marginBottom="19dp"
    android:text="@string/multi_auto_complete"
    android:textAppearance="?android:attr/textAppearanceMedium" />

</RelativeLayout>

```

Here is the content of **Strings.xml**

```

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

    <string name="app_name">AutoComplete</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="auto_complete">AutoComplete</string>
    <string name="multi_auto_complete">Multi AutoComplete</string>
    <string-array name="list_of_countries">
        <item >USA</item>
        <item >Uk</item>
        <item >Canada</item>
        <item >Australia</item>
        <item >France</item>
        <item >Italy</item>

        <item >China</item>
    </string-array>

```

```

    <item >Japan</item>
    <item >Spain</item>
  </string-array>

</resources>

```

Here is the content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.autocomplete"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

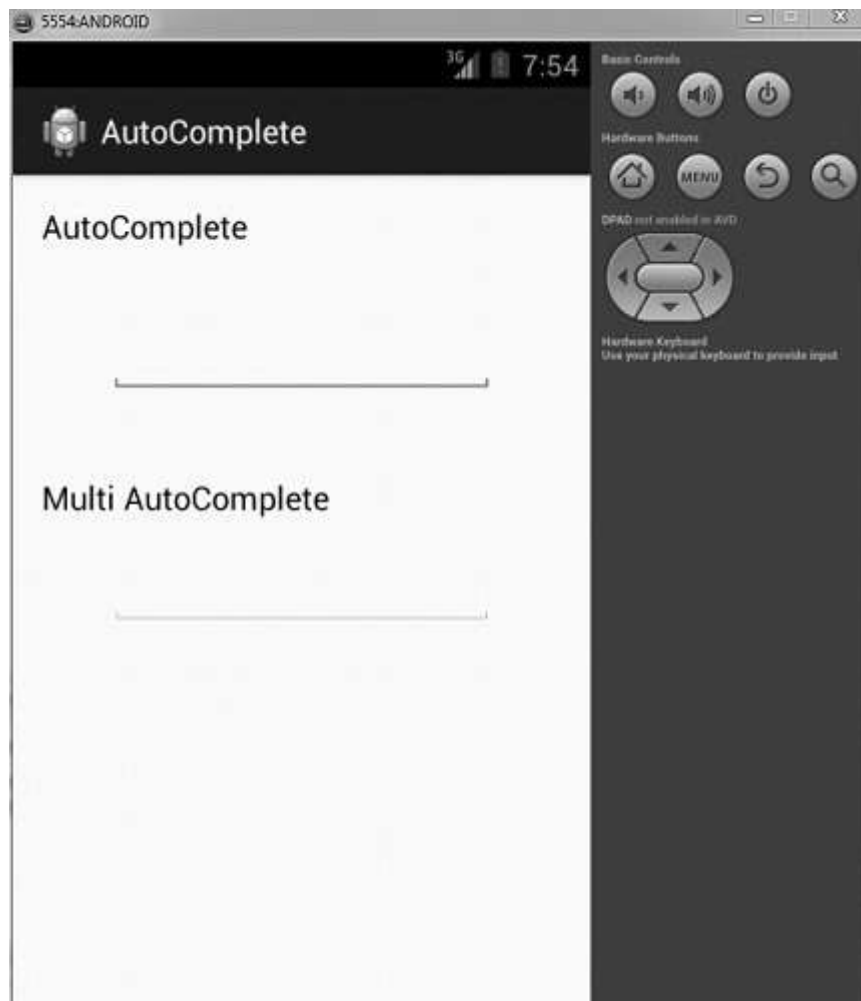
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.autocomplete.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

```

```
</manifest>
```

Let's try to run your Androidmanager application. We assume, you have connected 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 will install this application in your AVD and your AVD will display following screen.



Now just type in the text view to see suggestions of the country. As we type two letters which are **ca**, it shows us suggestion of Canada:



The multiAutoCompleteTextView demonstrates suggestions for not only a word but for whole text. As after writing first word, when we start writing the second word, it displays the suggestions. This can be shown in the picture below.



31. BEST PRACTICES

There are some practices that you can follow while developing android application. These are suggested by the android itself and they keep on improving with respect to time.

These best practices include interaction design features, performance, security and privacy, compatibility, testing, distributing and monetizing tips. They are narrowed down and are listed as below.

Best Practices - User input

Every text field is intended for a different job. For example, some text-fields are for text and some are for numbers. If it is for numbers then it is better to display the numeric keypad when that text-field is focused. Its syntax is.

```
<EditText
    android:id="@+id/phone"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="phone" />
```

Other than that if your field is for password, then it must show a password hint, so that the user can easily remember the password. It can be achieved as:

```
<EditText
    android:id="@+id/password"
    android:hint="@string/password_hint"
    android:inputType="textPassword" />
```

Best Practices - Background jobs

There are certain jobs in an application that are run in the application background. Their job might be to fetch some thing from the internet, playing music etc. It is recommended that the long awaiting tasks should not be done in the UI thread and rather in the background by services or AsyncTask.

AsyncTask Vs Services.

Both are used for doing background tasks, but the service is not affected by most user interface life cycle events, so it continues to run in circumstances that would shut down an AsyncTask.

Best Practices - Performance

Your application performance should be up to the mark. But it should perform differently; not on the front end, but on the back end when the device is connected to a power source or charging. Charging could be from USB and from wire cable.

When your device is charging itself, it is recommended to update your application settings if any, such as maximizing your refresh rate whenever the device is connected. It can be done as follows:

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);
// Are we charging / charged? Full or charging.
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
// How are we charging? From AC or USB.
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED,
-1);
```

Best Practices - Security and privacy

It is very important that your application should be secure. Not only the application, but the user data and the application data should also be secured. The security can be increased by the following factors:

- Use internal storage rather than external for storing applications files
- Use content providers wherever possible
- Use SSL when connecting to the web
- Use appropriate permissions for accessing different functionalities of device.

Example:

The below example demonstrates some of the best practices you should follow when developing android application. It creates a basic application that allows you to specify how to use text fields and how to increase performance by checking the charging status of the phone.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as BestPractices under a package com.example.autocomplete. While creating this project, make sure you Target SDK and Compile With at

	the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add the code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Modify AndroidManifest.xml to add necessary permissions.
6	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.bestpractices/MainActivity.java**

```

package com.example.bestpractices;

import android.os.BatteryManager;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

    private Button Check;
    private BatteryManager battery;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

```



```
        setContentView(R.layout.activity_main);
        Check = (Button)findViewById(R.id.button1);
    }

    public void check(View view){
        IntentFilter ifilter = new
        IntentFilter(Intent.ACTION_BATTERY_CHANGED);
        Intent batteryStatus = registerReceiver(null, ifilter);
        int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS,
                                                -1);

        boolean isCharging = status ==
        BatteryManager.BATTERY_STATUS_CHARGING ||
        status == BatteryManager.BATTERY_STATUS_FULL;
        // How are we charging?
        int chargePlug =
        batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED,
        -1);
        boolean usbCharge = chargePlug ==
        BatteryManager.BATTERY_PLUGGED_USB;
        boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;

        if(usbCharge){
            Toast.makeText(getApplicationContext(),"Mobile is
            charging on USB",Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(getApplicationContext(),"Mobile is
            charging on AC",Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
```

```

        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="22dp"
        android:layout_marginTop="20dp"
        android:text="@string/username"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/message"
        android:layout_width="wrap_content"

```

```
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:ems="10"
        android:inputType="textCapSentences|textAutoCorrect" >

    <requestFocus />
</EditText>

<EditText
    android:id="@+id/password"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="34dp"
    android:ems="10"
    android:hint="@string/password_hint"
    android:inputType="textPassword" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/textView1"
    android:layout_below="@+id/message"
    android:layout_marginTop="50dp"
    android:text="@string/password"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

    android:layout_below="@+id/password"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="34dp"
    android:onClick="check"
    android:text="@string/check" />

```

```
</RelativeLayout>
```

Here is the content of **Strings.xml**

```

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

    <string name="app_name">BestPractices</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="username">Username</string>
    <string name="password">Password</string>
    <string name="password_hint">Hello world!</string>
    <string name="check">Charging check</string>

</resources>

```

Here is the content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.bestpractices"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


    <application

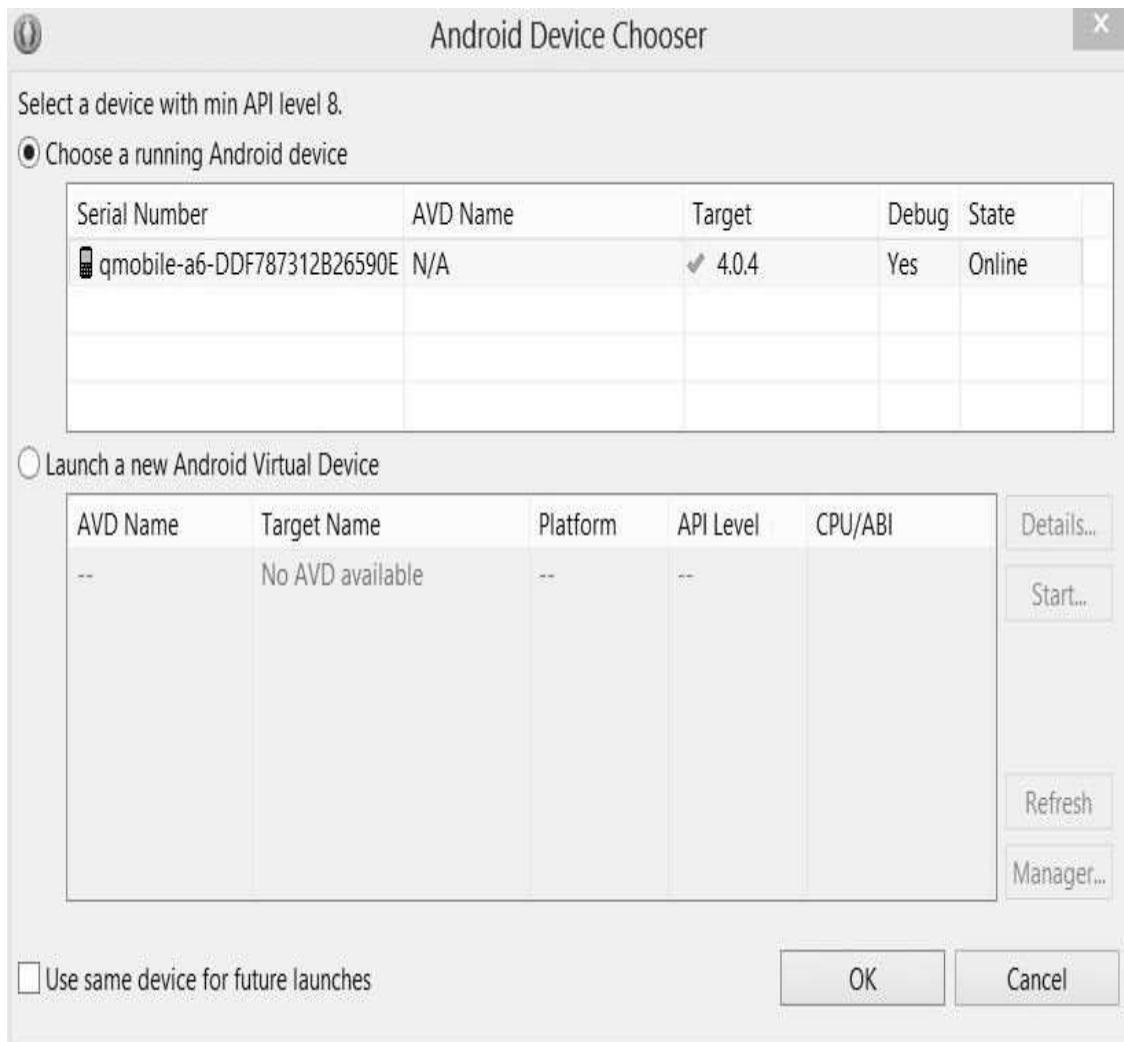
```

```
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.bestpractices.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

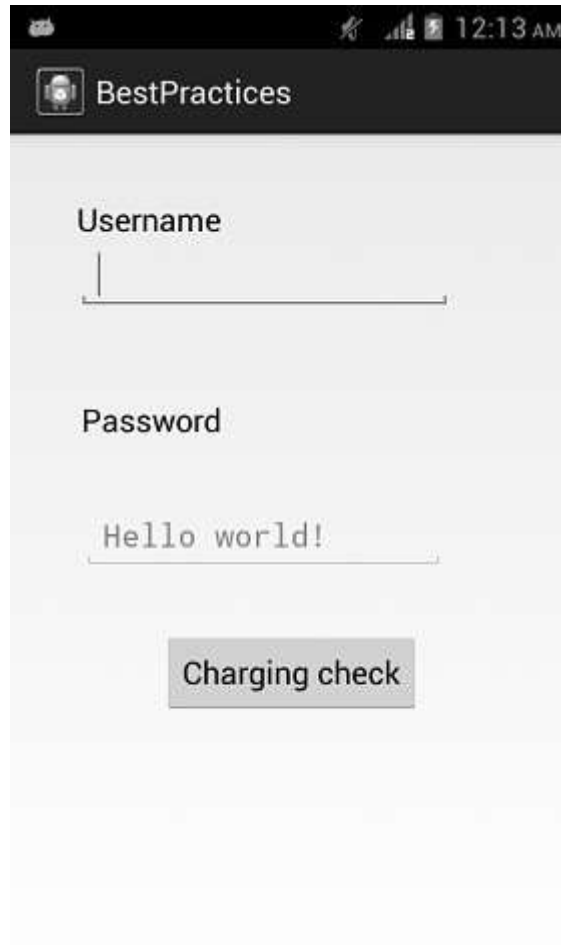
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

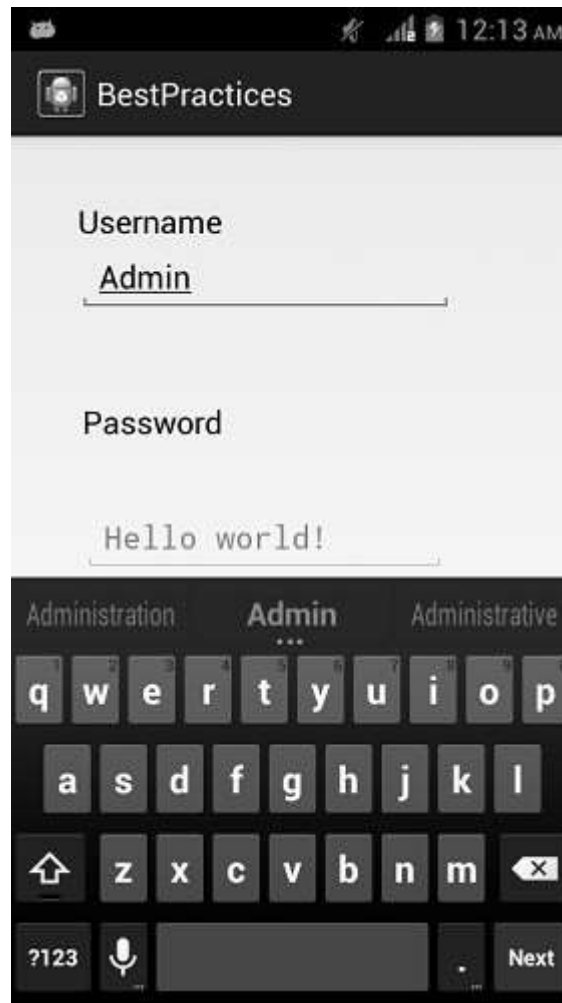
Let's try to run your BestPractices application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



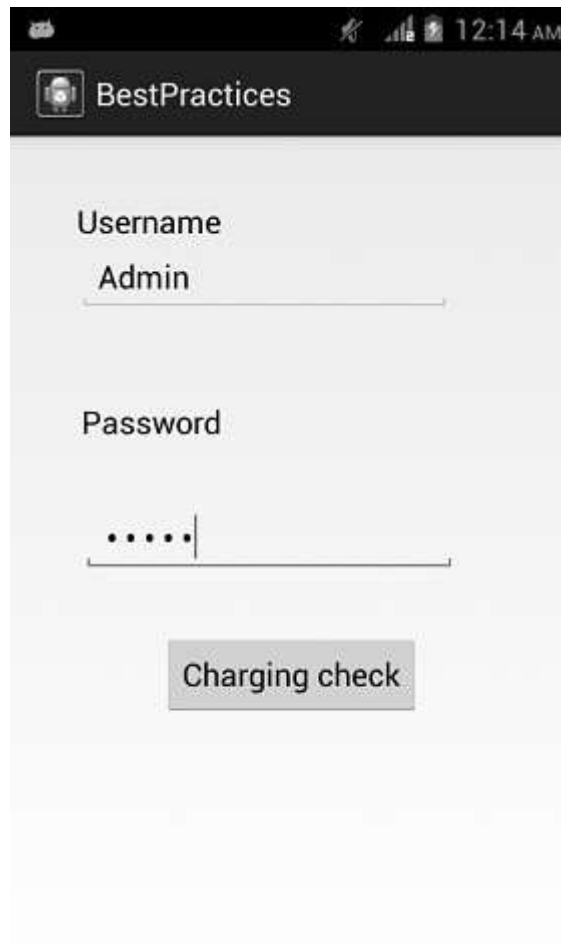
Select your mobile device as an option and then check your mobile device which will display following screen.



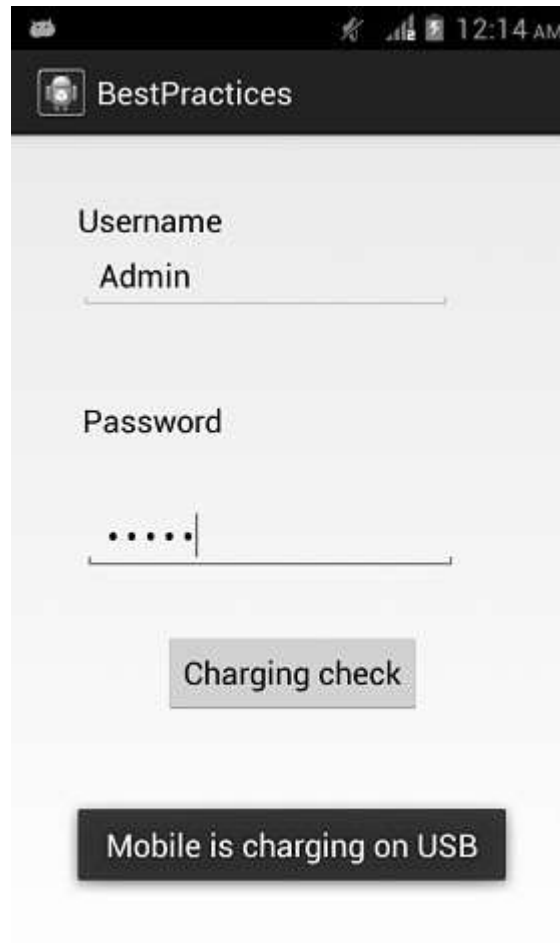
Now just type on the username field and you will see the built in android suggestions from the dictionary will start coming up. This is shown below:



Now you will see the hint in the password field. It would disappear as soon as you start writing in the field. It is shown below.



In the end, just connect your device to AC cable or USB cable and press on **charging check** button. In our case, we connect it with a PC via USB cable so it shows the following message:



32. BLUETOOTH

Among many ways, Bluetooth is a way to send or receive data between two different devices. Android platform includes support for the Bluetooth framework that allows a device to wirelessly exchange data with other Bluetooth devices.

Android provides Bluetooth API to perform these different operations.

- Scan for other Bluetooth devices
- Get a list of paired devices
- Connect to other devices through service discovery

Android provides BluetoothAdapter class to communicate with Bluetooth. Create an object of this calling by calling the static method getDefaultAdapter(). Its syntax is given below.

```
private BluetoothAdapter BA;  
BA = BluetoothAdapter.getDefaultAdapter();
```

In order to enable the Bluetooth of your device, call the intent with the following Bluetooth constant ACTION_REQUEST_ENABLE. Its syntax is.

```
Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);  
startActivityForResult(turnOn, 0);
```

Apart from this constant, there are other constant provided by the API, that supports different tasks. They are listed below:

Sr.No	Constant & description
1	ACTION_REQUEST_DISCOVERABLE This constant is used to turn on discovering of Bluetooth.
2	ACTION_STATE_CHANGED This constant will notify that Bluetooth state has been changed.
3	ACTION_FOUND This constant is used for receiving information about each device that is discovered.

Once you enable the Bluetooth, you can get a list of paired devices by calling getBondedDevices() method. It returns a set of bluetooth devices. Its syntax is.

```
private Set<BluetoothDevice>pairedDevices;
pairedDevices = BA.getBondedDevices();
```

Apart from the pairedDevices, there are other methods in the API that gives more control over Bluetooth. They are listed below.

Sr.No	Method & description
1	enable() This method enables the adapter if not enabled.
2	isEnabled() This method returns true if adapter is enabled.
3	disable() This method disables the adapter.
4	getName() This method returns the name of the Bluetooth adapter.
5	setName(String name) This method changes the Bluetooth name.
6	getState() This method returns the current state of the Bluetooth Adapter.
7	startDiscovery() This method starts the discovery process of the Bluetooth for 120 seconds.

Example:

This example provides demonstration of BluetoothAdapter class to manipulate Bluetooth and show list of paired devices by the Bluetooth.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as AudioCapture under a package com.example.audiocapture. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add the code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Modify AndroidManifest.xml to add necessary permissions.
6	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.bluetooth/MainActivity.java**

```
package com.example.bluetooth;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;

import android.os.Bundle;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
```

```
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private Button On,Off,Visible,list;
    private BluetoothAdapter BA;
    private Set<BluetoothDevice>pairedDevices;
    private ListView lv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        On = (Button)findViewById(R.id.button1);
        Off = (Button)findViewById(R.id.button2);
        Visible = (Button)findViewById(R.id.button3);
        list = (Button)findViewById(R.id.button4);

        lv = (ListView)findViewById(R.id.listView1);

        BA = BluetoothAdapter.getDefaultAdapter();
    }

    public void on(View view){
        if (!BA.isEnabled()) {
            Intent turnOn = new
            Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(turnOn, 0);
            Toast.makeText(getApplicationContext(),"Turned on"
            ,Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(getApplicationContext(),"Already on",
```

```

        Toast.LENGTH_LONG).show();
    }
}

public void list(View view){
    pairedDevices = BA.getBondedDevices();

    ArrayList list = new ArrayList();
    for(BluetoothDevice bt : pairedDevices)
        list.add(bt.getName());

    Toast.makeText(getApplicationContext(),"Showing Paired Devices",
    Toast.LENGTH_SHORT).show();
    final ArrayAdapter adapter = new ArrayAdapter
    (this,android.R.layout.simple_list_item_1, list);
    lv.setAdapter(adapter);

}

public void off(View view){
    BA.disable();
    Toast.makeText(getApplicationContext(),"Turned off" ,
    Toast.LENGTH_LONG).show();
}

public void visible(View view){
    Intent getVisible = new Intent(BluetoothAdapter.
    ACTION_REQUEST_DISCOVERABLE);
    startActivityForResult(getVisible, 0);

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

```

```

    }
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ScrollView
        android:id="@+id/scrollView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical" >

            <TextView
                android:id="@+id/textView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"

```



```
        android:text="@string/app_name"
        android:textAppearance="?android:attr/textAppearanceLarge" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="on"
    android:text="@string/on" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="visible"
    android:text="@string/Visible" />

<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="list"
    android:text="@string/List" />

<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="off"
    android:text="@string/off" />

<ListView
    android:id="@+id/listView1"
    android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:visibility="visible" >

</ListView>

</LinearLayout>
</ScrollView>

</RelativeLayout>

```

Here is the content of **Strings.xml**

```

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

    <string name="app_name">Bluetooth</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="on">Turn On</string>
    <string name="off">Turn Off</string>
    <string name="Visible">Get Visible</string>
    <string name="List">List Devices</string>

</resources>

```

Here is the content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.bluetooth"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


```

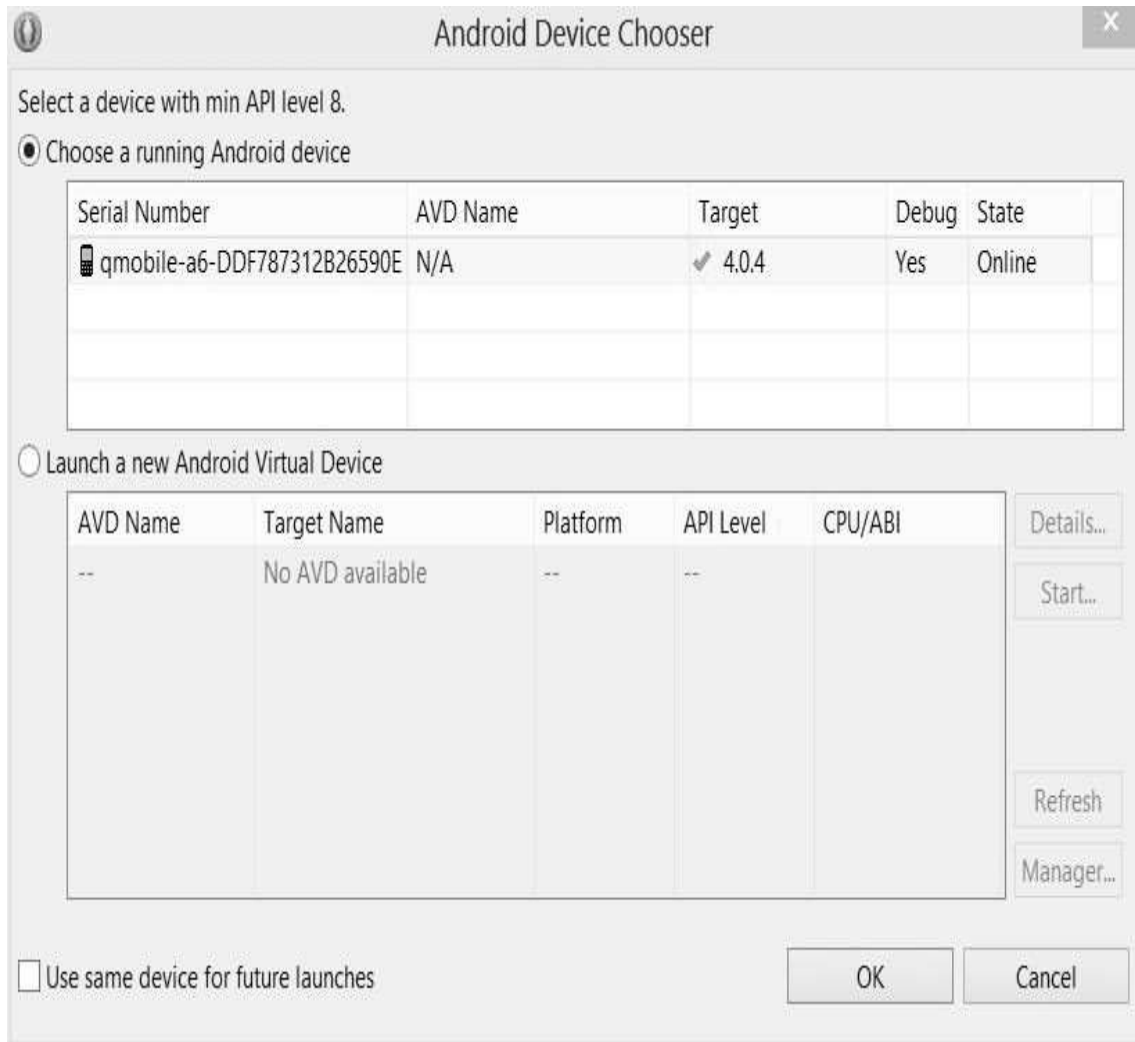
```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.bluetooth.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

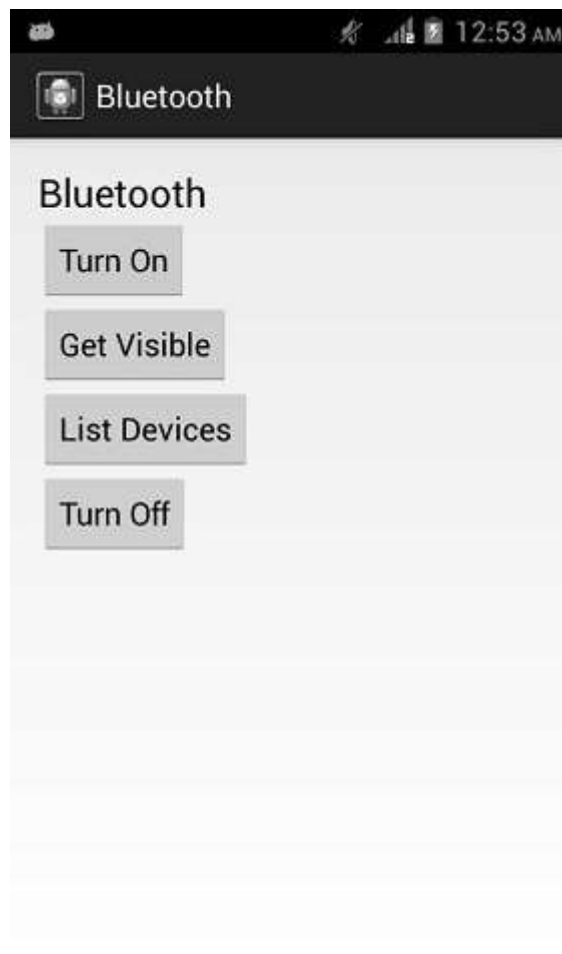
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

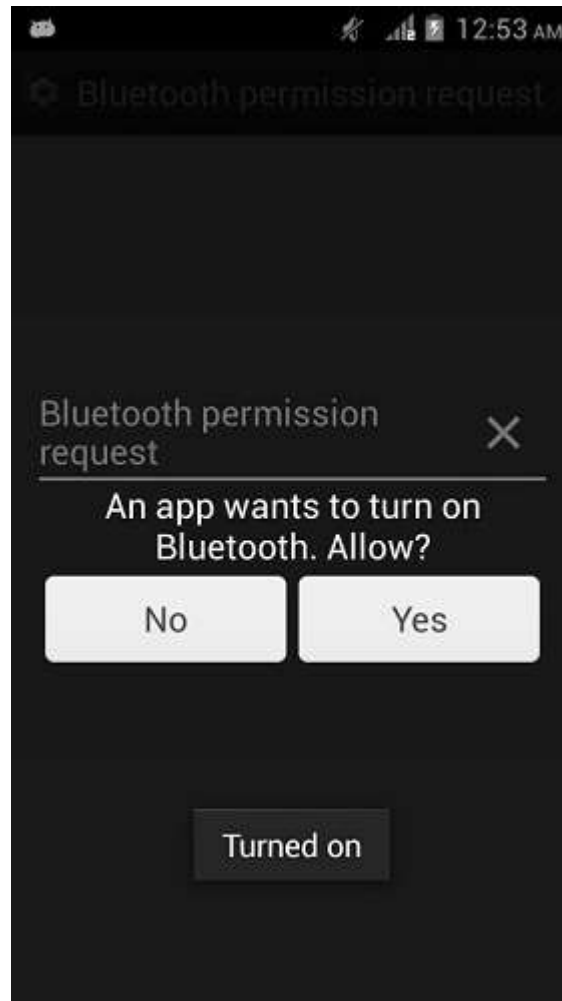
Let's try to run your AndroidCapture application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



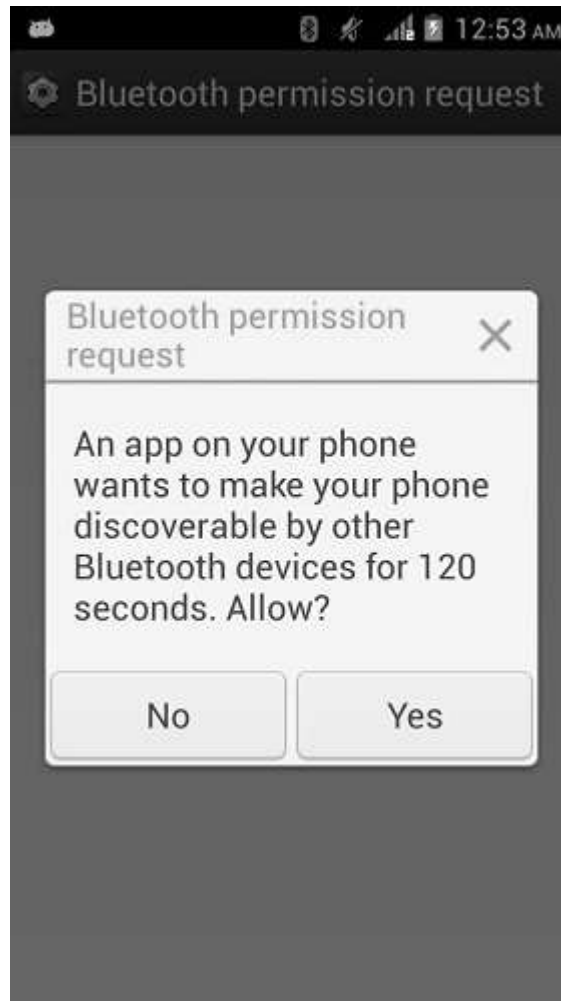
Select your mobile device as an option and then check your mobile device which will display following screen:



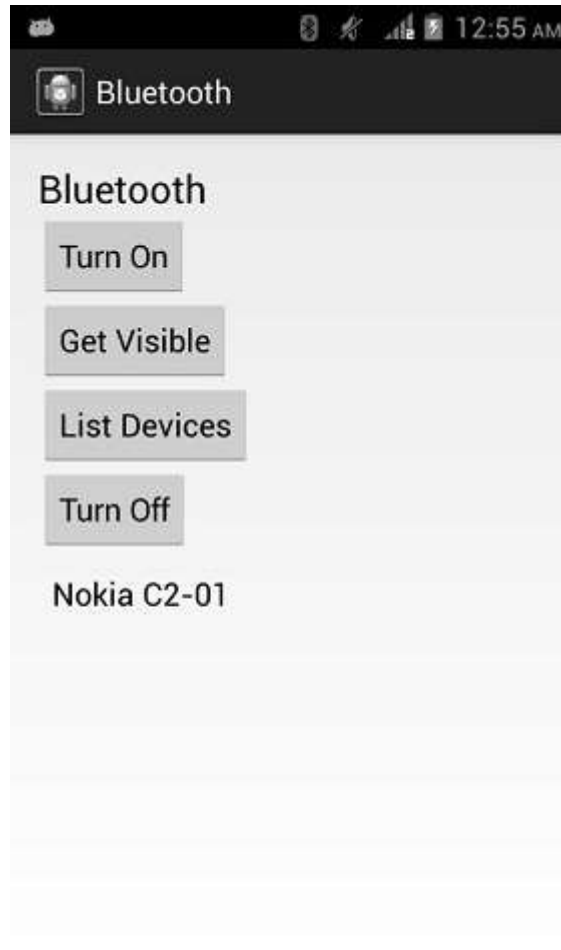
Now select **Turn On** to turn on the bluetooth. But as you select it, your Bluetooth will not be turned on. In fact, it will ask your permission to enable the Bluetooth.



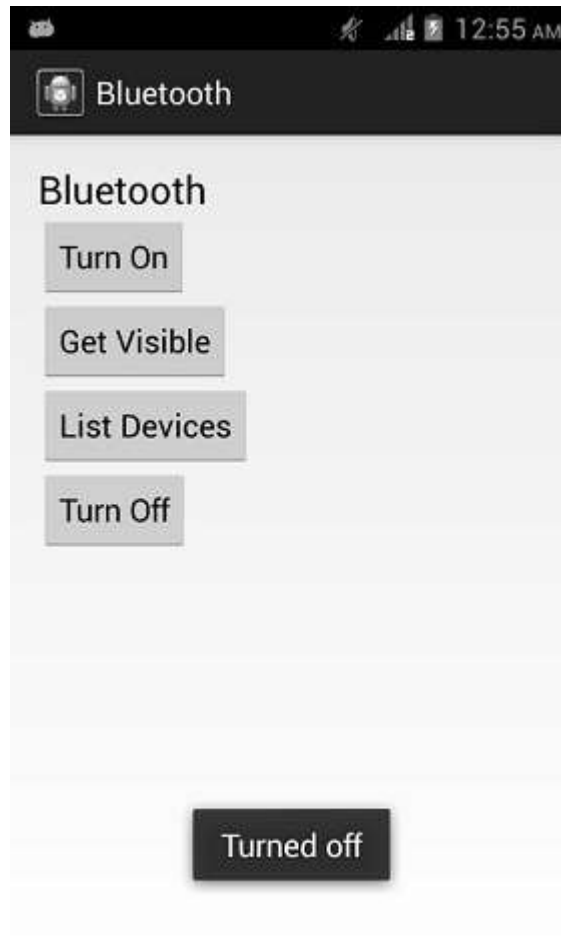
Now just select the Get Visible button to turn on your visibility. The following screen would appear asking your permission to turn on discovery for 120 seconds.



Now just select the List Devices option. It will list down the paired devices in the list view. In our case, we have only one paired device. It is shown below:



Now just select the Turn off button to switch off the Bluetooth. Following message would appear when you switch off the bluetooth indicating the successful switching off of Bluetooth.



33. CAMERA

These are the following two ways, in which you can use camera in your application

- Using existing android camera application in our application
- Directly using Camera API provided by android in our application

Using existing android camera application in our application

You will use `MediaStore.ACTION_IMAGE_CAPTURE` to launch an existing camera application installed on your phone. Its syntax is given below:

```
Intent intent = new  
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
```

Apart from the above, there are other available Intents provided by MediaStore. They are listed as follows:

Sr.No	Intent type and description
1	ACTION_IMAGE_CAPTURE_SECURE It returns the image captured from the camera, when the device is secured.
2	ACTION_VIDEO_CAPTURE It calls the existing video application in android to capture video.
3	EXTRA_SCREEN_ORIENTATION It is used to set the orientation of the screen to vertical or landscape.
4	EXTRA_FULL_SCREEN It is used to control the user interface of the ViewImage.
5	INTENT_ACTION_VIDEO_CAMERA This intent is used to launch the camera in the video mode.
6	EXTRA_SIZE_LIMIT

It is used to specify the size limit of video or image capture size.
--

Now you will use the function *startActivityForResult()* to launch this activity and wait for its result. Its syntax is given below:

<code>startActivityForResult(intent, 0)</code>
--

This method has been defined in the **activity** class. We are calling it from main activity. There are methods defined in the activity class that does the same job, but used when you are not calling from the activity but from somewhere else. They are listed below:

Sr.No	Activity function description
1	startActivityForResult(Intent intent, int requestCode, Bundle options) It starts an activity, but can take extra bundle of options with it.
2	startActivityFromChild(Activity child, Intent intent, int requestCode) It launches the activity when your activity is child of any other activity.
3	startActivityFromChild(Activity child, Intent intent, int requestCode, Bundle options) It work same as above, but it can take extra values in the shape of bundle with it.
4	startActivityFromFragment(Fragment fragment, Intent intent, int requestCode) It launches activity from the fragment you are currently inside.
5	startActivityFromFragment(Fragment fragment, Intent intent, int requestCode, Bundle options) It not only launches the activity from the fragment, but can take extra values with it.

No matter which function you used to launch the activity, they all return the result. The result can be obtained by overriding the function *onActivityResult*

Example:

Here is an example that shows how to launch the existing camera application to capture an image and display the result in the form of bitmap.

To experiment with this example, you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Camera under a package com.example.camera. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add intent code to launch the activity and result method to receive the output.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required. Here we add only imageView and a textView.
4	Modify res/values/strings.xml to define required constant values.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.camera/MainActivity.java**.

```
package com.example.camera;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;
```

```
public class MainActivity extends Activity {

    ImageView imgFavorite;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imgFavorite = (ImageView)findViewById(R.id.imageView1);
        imgFavorite.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                open();
            }
        });
    }

    public void open(){
        Intent intent = new
        Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(intent, 0);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        // TODO Auto-generated method stub
        super.onActivityResult(requestCode, resultCode, data);
        Bitmap bp = (Bitmap) data.getExtras().get("data");
        imgFavorite.setImageBitmap(bp);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
```

```

    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Following will be the content of **res/layout/activity_main.xml** file:

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginLeft="34dp"
        android:layout_marginTop="36dp"
        android:contentDescription="@string/hello_world"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignRight="@+id/imageView1"
        android:text="@string/tap"

```

```

        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Following will be the content of **res/values/strings.xml** to define one new constants

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Camera</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="tap">Tap the image to open the camera!!</string>
</resources>

```

Following is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.camera"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.camera.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>

```


```

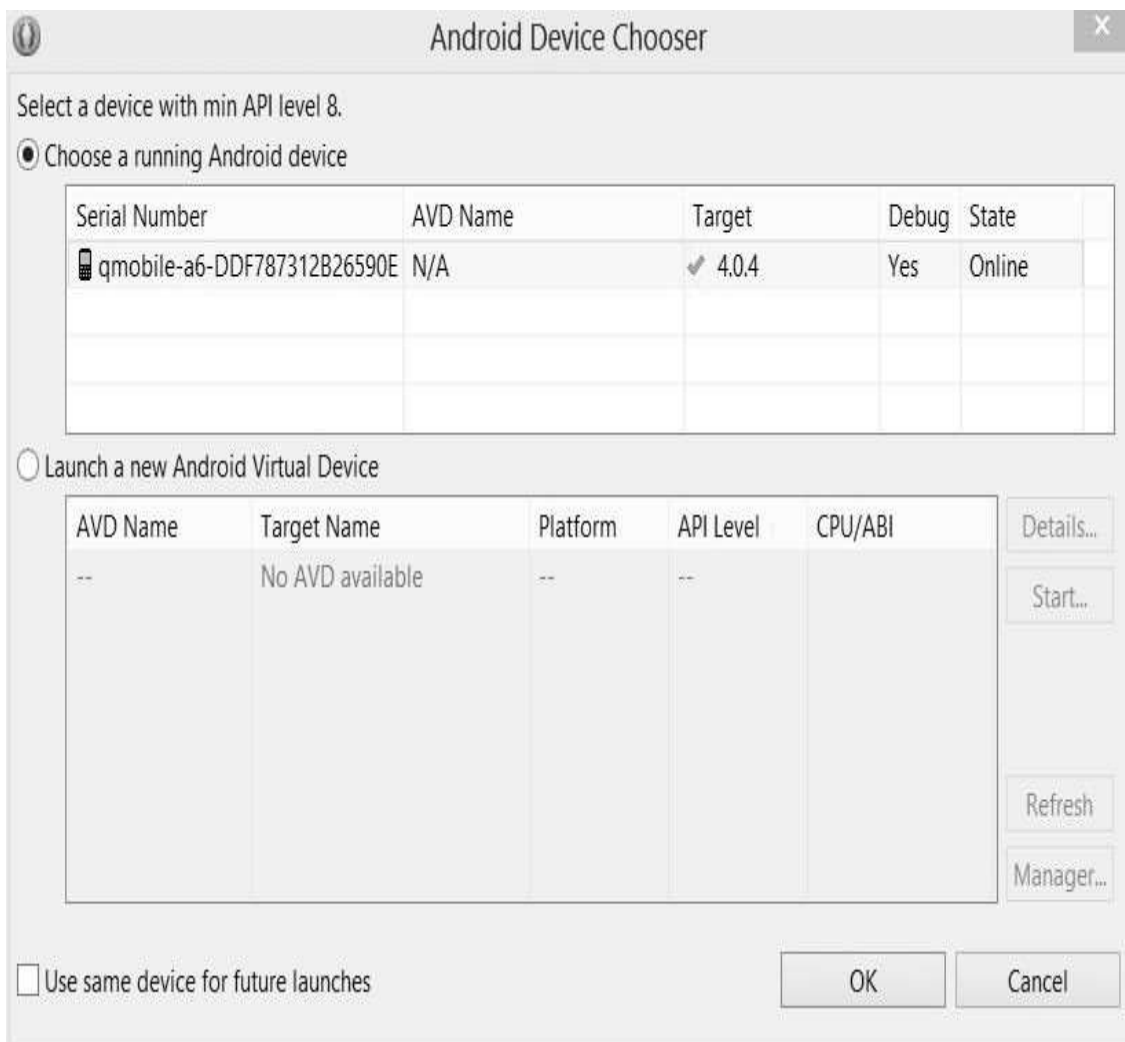
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER"
        />
    </intent-filter>
</activity>
</application>

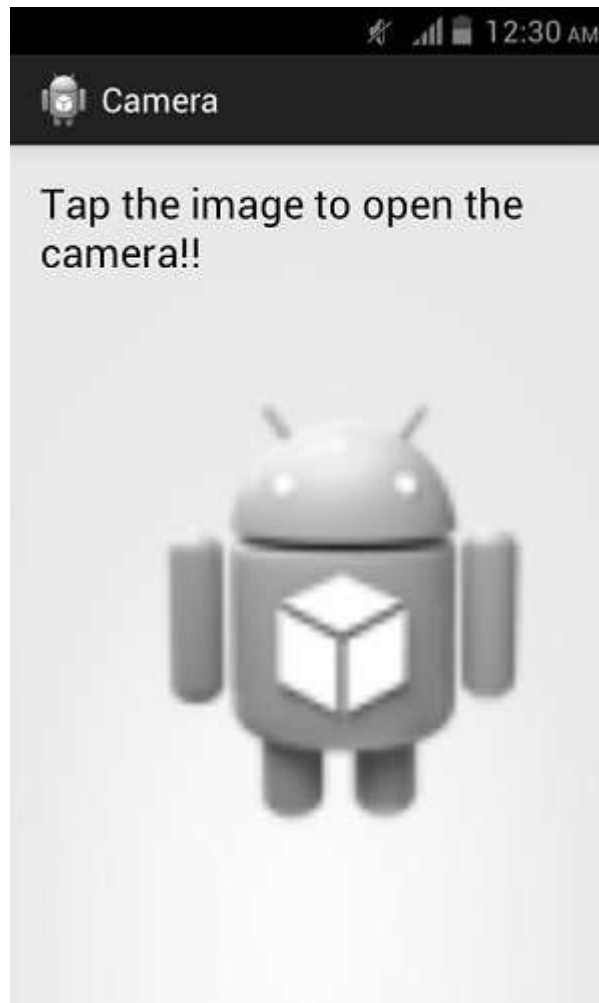
</manifest>

```

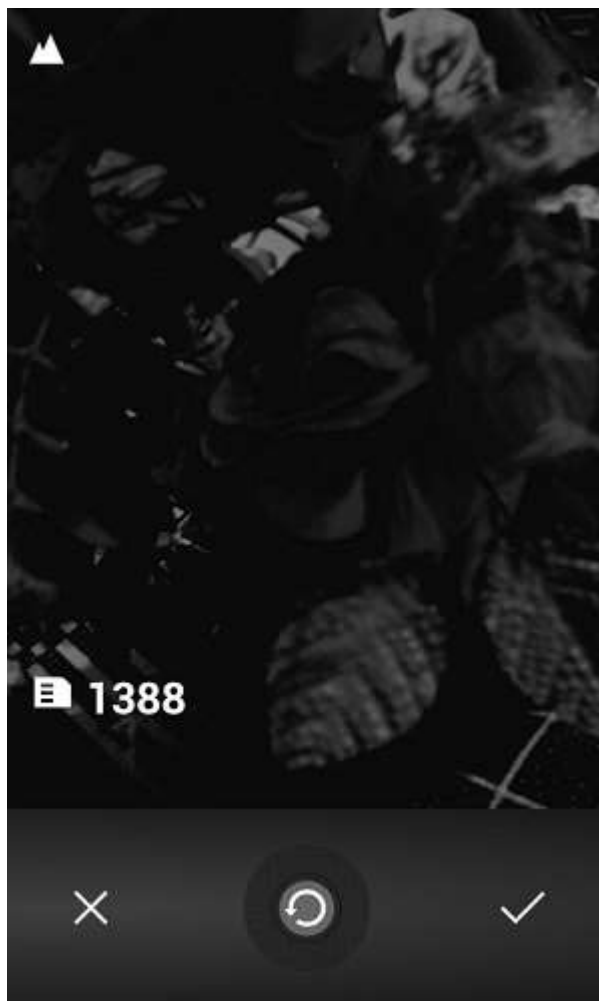
Let's try to run your Camera application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



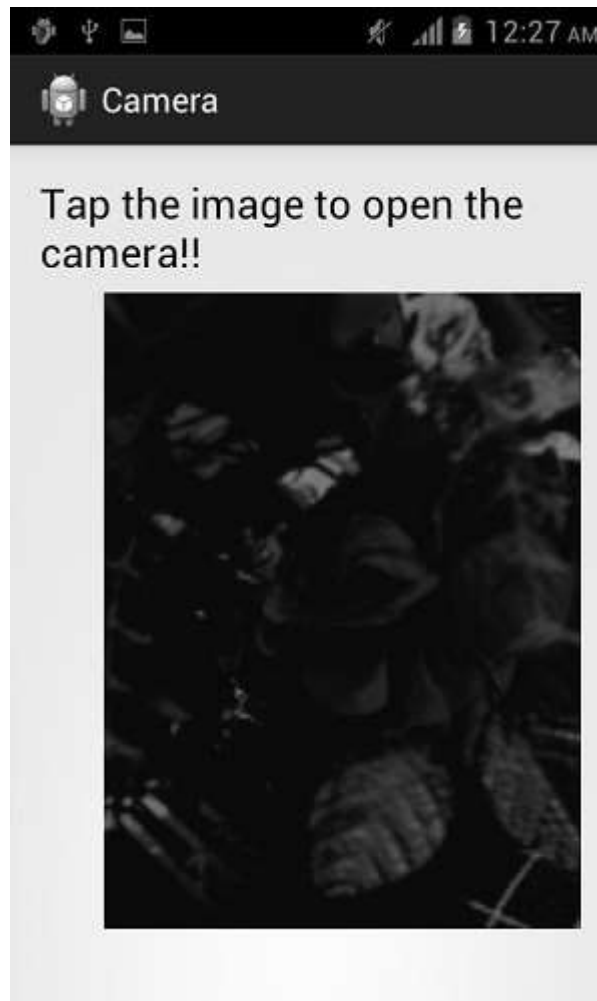
Select your mobile device as an option and then check your mobile device which will display the following screen:



Now just tap on the image of android icon and the camera will be opened. Just capture a picture. After capturing it, two buttons will appear asking you to discard it or to keep it:



Just press the tic (green) button and you will be brought back to your application with the captured image in place of android icon.



Directly using Camera API Provided by Android in our Application

We will be using the camera API to integrate the camera in our application.

First you will need to initialize the camera object using the static method provided by the api called *Camera.open*. Its syntax is:

```
Camera object = null;
object = Camera.open();
```

Apart from the above function, there are other functions provided by the Camera class that are listed below:

Sr.No	Method & Description
-------	----------------------

1	getCameraInfo(int cameraId, Camera.CameraInfo cameraInfo) It returns the information about a particular camera.
2	getNumberOfCameras() It returns an integer number defining of cameras available on device.
3	lock() It is used to lock the camera, so no other application can access it.
4	release() It is used to release the lock on camera, so other applications can access it.
5	open(int cameraId) It is used to open particular camera when multiple cameras are supported.
6	enableShutterSound(boolean enabled) It is used to enable/disable default shutter sound of image capture.

Now you need to make a separate class and extend it with SurfaceView and implement SurfaceHolder interface.

The two classes that have been used have the following purpose:

Class	Description
Camera	It is used to control the camera and take images or capture video from the camera.
SurfaceView	This class is used to present a live camera preview to the user.

You have to call the preview method of the camera class to start the preview of the camera to the user.

```
public class ShowCamera extends SurfaceView implements
SurfaceHolder.Callback {
```

```

private Camera theCamera;

public void surfaceCreated(SurfaceHolder holder) {
    theCamera.setPreviewDisplay(holder);
    theCamera.startPreview();
}
public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int
arg3){
}
public void surfaceDestroyed(SurfaceHolder arg0) {
}
}

```

Apart from the preview there are other options of the camera that can be set using the other functions provided by the Camera API.

Sr.No	Method & Description
1	startFaceDetection() This function starts the face detection in the camera.
2	stopFaceDetection() It is used to stop the face detection which is enabled by the above function.
3	startSmoothZoom(int value) It takes an integer value and zoom the camera very smoothly to that value.
4	stopSmoothZoom() It is used to stop the zoom of the camera.
5	stopPreview() It is used to stop the preview of the camera to the user.
6	takePicture(Camera.ShutterCallback shutter,

	<p>Camera.PictureCallback raw, Camera.PictureCallback jpeg) It is used to enable/disable default shutter sound of image capture.</p>
--	--

Example:

Following example demonstrates the usage of the camera API in the application.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, because camera is not supported by the emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Camera under a package com.example.camera1. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add the respective code of camera and get references to the XML components.
3	Create a new ShowCamera.java file to extend it with SurfaceView and implement the SurfaceHolder interface.
4	Modify layout XML file res/layout/activity_main.xml add any GUI component if required. Here we add only FrameView and a button and an ImageView.
5	Modify res/values/strings.xml to define required constant values.
6	Modify AndroidManifest.xml as shown below to add the necessary permissions for camera.
7	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.camera1/MainActivity.java**.

```
package com.example.camera1;
```

```
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private Camera cameraObject;
    private ShowCamera showCamera;
    private ImageView pic;
    public static Camera isCameraAvailiable(){
        Camera object = null;
        try {
            object = Camera.open();
        }
        catch (Exception e){
        }
        return object;
    }

    private PictureCallback capturedIt = new PictureCallback() {

        @Override
        public void onPictureTaken(byte[] data, Camera camera) {
```

```

        Bitmap bitmap = BitmapFactory.decodeByteArray(data , 0, data
        .length);
        if(bitmap==null){
            Toast.makeText(getApplicationContext(), "not taken",
            Toast.LENGTH_SHORT).show();
        }
        else
        {
            Toast.makeText(getApplicationContext(), "taken",
            Toast.LENGTH_SHORT).show();
        }
        cameraObject.release();
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    pic = (ImageView)findViewById(R.id.imageView1);
    cameraObject = isCameraAvailiable();
    showCamera = new ShowCamera(this, cameraObject);
    FrameLayout preview = (FrameLayout)
    findViewById(R.id.camera_preview);
    preview.addView(showCamera);
}
public void snapIt(View view){
    cameraObject.takePicture(null, null, capturedIt);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

```



```

    }
}

```

Create the new java file called as **src/com.example.camera1/ShowCamera.java**. and add the following code:

```

package com.example.camera1;

import java.io.IOException;

import android.content.Context;
import android.hardware.Camera;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class ShowCamera extends SurfaceView implements
SurfaceHolder.Callback {

    private SurfaceHolder holdMe;
    private Camera theCamera;

    public ShowCamera(Context context, Camera camera) {
        super(context);
        theCamera = camera;
        holdMe = getHolder();
        holdMe.addCallback(this);
    }

    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int
arg3) {
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        try {

```

```

        theCamera.setPreviewDisplay(holder);
        theCamera.startPreview();
    } catch (IOException e) {
    }
}

@Override
public void surfaceDestroyed(SurfaceHolder arg0) {
}
}

```

Modify the content of the **res/layout/activity_main.xml** :

```

<?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="horizontal" >

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="0.30"
        android:orientation="vertical" >

        <FrameLayout
            android:id="@+id/camera_preview"
            android:layout_width="fill_parent"
            android:layout_height="199dp" />

        <Button
            android:id="@+id/button_capture"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"

```

```

        android:onClick="snapIt"
        android:text="@string/Capture" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="fitXY"
        android:src="@drawable/ic_launcher" />

</LinearLayout>
< /LinearLayout>

```

Modify the content of the **res/values/string.xml**

```

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

    <string name="app_name">Camera1</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Capture">Capture</string>

</resources>

```

Modify the content of the **AndroidManifest.xml** and add the necessary permissions as shown below.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.camera1"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"

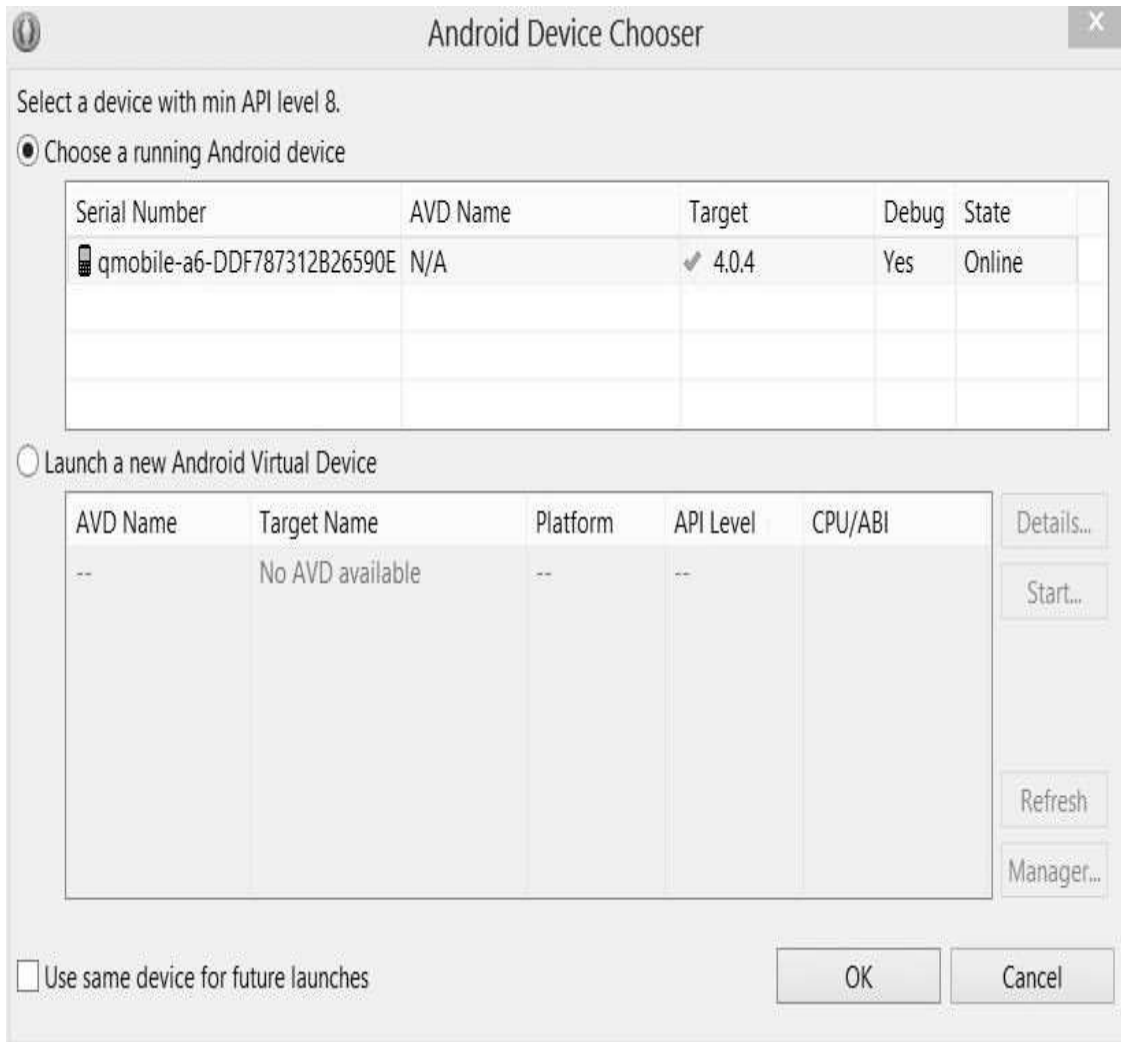
```

```
        android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.camera1.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Let's try to run your SendSMSDemo application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



The camera would start showing its preview in the upper half panel. Just click the capture button. You can now either store the captured image, upload it to the web or either discard it.

34. CLIPBOARD

Android provides the clipboard framework for copying and pasting different types of data. The data could be text, images, binary stream data or other complex data types.

Android provides the library of ClipboardManager and ClipData and ClipData.item to use the copying and pasting framework. To use clipboard framework, you need to put data into clip object, and then put that object into system wide clipboard.

In order to use clipboard, you need to instantiate an object of ClipboardManager by calling the **getSystemService()** method. Its syntax is given below:

```
ClipboardManager myClipboard;  
myClipboard = (ClipboardManager)getSystemService(CLIPBOARD_SERVICE);
```

Copying data

The next thing you need to do is to instantiate the ClipData object by calling the respective type of data method of the ClipData class. In case of text data, the **newPlainText** method will be called. After that you have to set that data as the clip of the Clipboard Manager object. Its syntax is given below:

```
ClipData myClip;  
String text = "hello world";  
myClip = ClipData.newPlainText("text", text);  
myClipboard.setPrimaryClip(myClip);
```

The ClipData object can take these three form and following functions are used to create those forms.

Sr.No	ClipData Form & Method
1	Text newPlainText(label, text) Returns a ClipData object whose single ClipData.Item object contains a text string.
2	URI

	<p><code>newUri(resolver, label, URI)</code></p> <p>Returns a <code>ClipData</code> object whose single <code>ClipData.Item</code> object contains a URI.</p>
3	<p>Intent</p> <p><code>newIntent(label, intent)</code></p> <p>Returns a <code>ClipData</code> object whose single <code>ClipData.Item</code> object contains an Intent.</p>

Pasting data

In order to paste the data, we will first get the clip by calling the **getPrimaryClip()** method. And from that clip we will get the item in `ClipData.Item` object. And from the object we will get the data. Its syntax is given below:

```
ClipData abc = myClipboard.getPrimaryClip();
ClipData.Item item = abc.getItemAt(0);
String text = item.getText().toString();
```

Apart from these methods, there are other methods provided by the `ClipboardManager` class for managing clipboard framework. These methods are listed below:

Sr.No	Method & description
1	<p>getPrimaryClip()</p> <p>This method just returns the current primary clip on the clipboard.</p>
2	<p>getPrimaryClipDescription()</p> <p>This method returns a description of the current primary clip on the clipboard but not a copy of its data.</p>
3	<p>hasPrimaryClip()</p> <p>This method returns true if there is currently a primary clip on the clipboard.</p>
4	<p>setPrimaryClip(ClipData clip)</p>

	This method sets the current primary clip on the clipboard.
5	setText(CharSequence text) This method can be directly used to copy text into the clipboard.
6	getText() This method can be directly used to get the copied text from the clipboard.

Example:

Here is an example demonstrating the use of ClipboardManager class. It creates a basic copy paste application that allows you to copy the text and then paste it via clipboard.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Clipboard under a package com.example.clipboard. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results

Following is the content of the modified main activity file **src/com.example.clipboard/MainActivity.java**.

```
package com.example.clipboard;
```

```
import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.ClipData;
import android.content.ClipboardManager;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    private ClipboardManager myClipboard;
    private ClipData myClip;
    private EditText copyField,pasteField;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myClipboard =
            (ClipboardManager) getSystemService(CLIPBOARD_SERVICE);
        copyField = (EditText)findViewById(R.id.editText1);
        pasteField = (EditText)findViewById(R.id.editText2);

    }

    @SuppressWarnings("NewApi")
    public void copy(View view){
        String text = copyField.getText().toString();
        myClip = ClipData.newPlainText("text", text);
        myClipboard.setPrimaryClip(myClip);
        Toast.makeText(getApplicationContext(), "Text Copied",
```

```

        Toast.LENGTH_SHORT).show();
    }

    @SuppressWarnings("NewApi")
    public void paste(View view){
        ClipData abc = myClipboard.getPrimaryClip();
        ClipData.Item item = abc.getItemAt(0);
        String text = item.getText().toString();
        pasteField.setText(text);
        Toast.makeText(getApplicationContext(), "Text Pasted",
            Toast.LENGTH_SHORT).show();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="25dp"
    android:layout_marginTop="19dp"
    android:text="@string/copytext"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="20dp"
    android:ems="10" >

    <requestFocus />
</EditText>

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_centerVertical="true"
    android:text="@string/pastetext"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<Button
```

```

    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/editText1"
    android:layout_marginLeft="65dp"
    android:layout_marginTop="20dp"
    android:onClick="copy"
    android:text="@string/copy" />

```

```
<EditText
```

```

    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="39dp"
    android:ems="10" />

```

```
<Button
```

```

    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button1"
    android:layout_below="@+id/editText2"
    android:layout_marginTop="34dp"
    android:onClick="paste"
    android:text="@string/paste" />

```

```
</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

```

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

```

```

<string name="app_name">Clipboard</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="copy">Copy Text</string>
<string name="paste">Paste Text</string>
<string name="copytext">Text to copy</string>
<string name="pastetext">Copied Text</string>
</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.clipboard"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

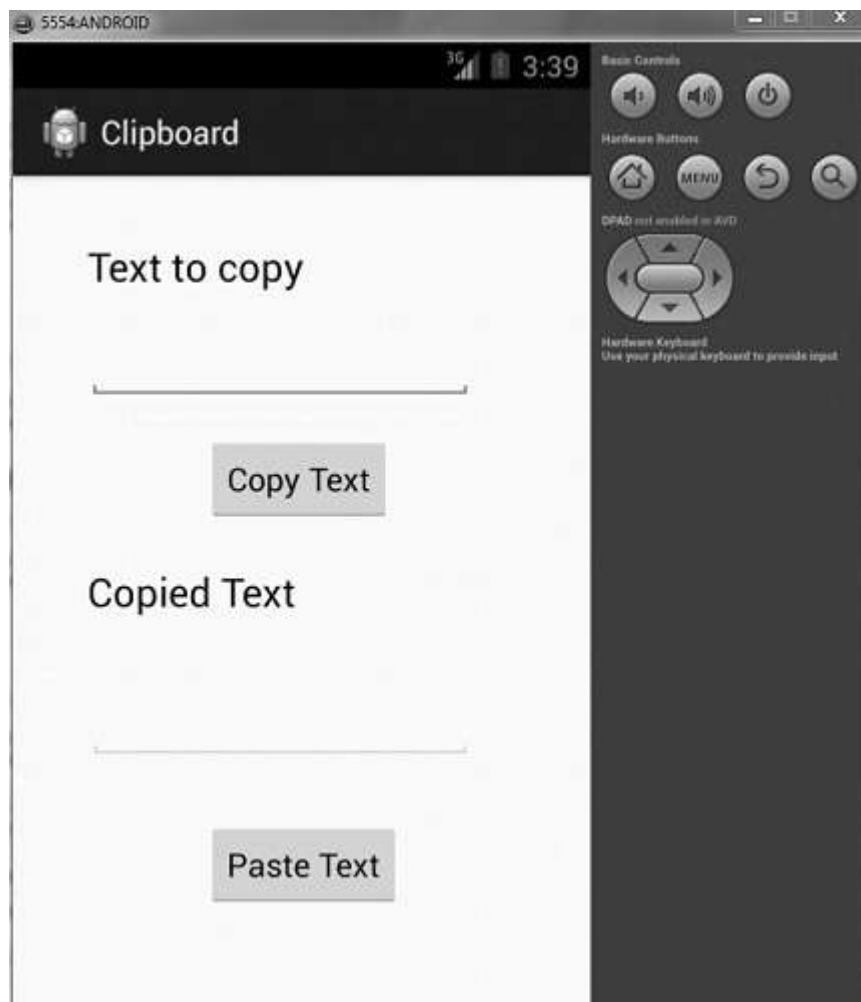
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.clipboard.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

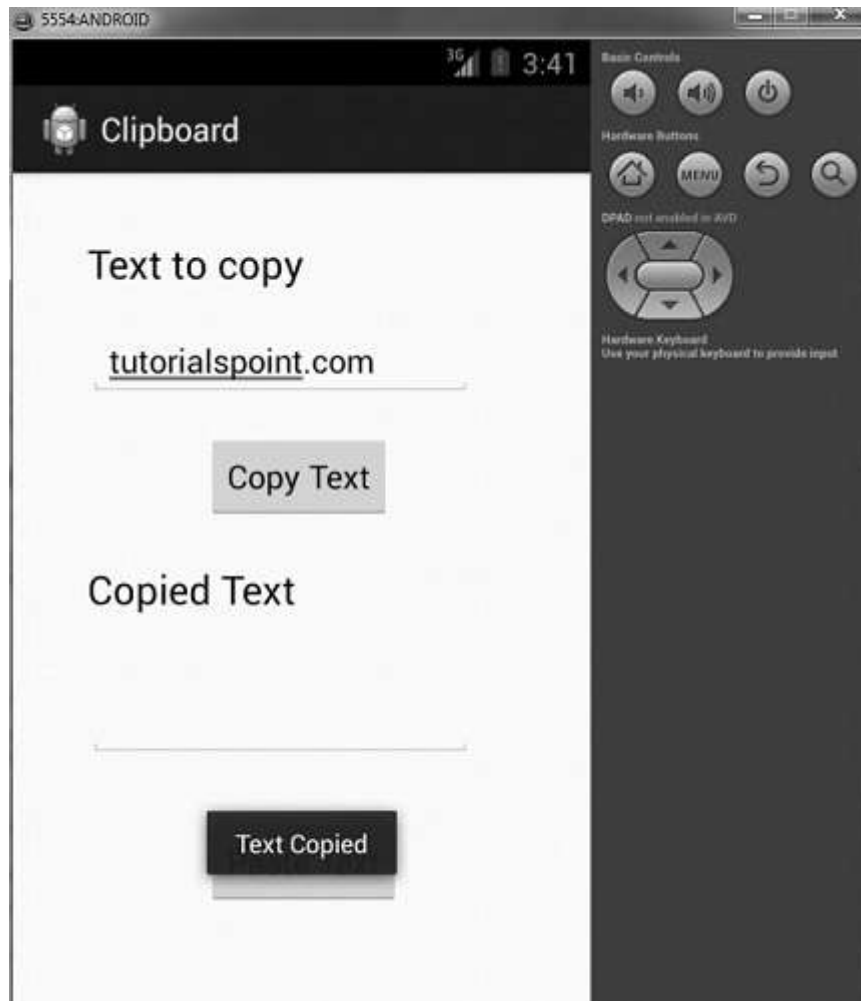
```

```
</activity>  
</application>  
  
</manifest>
```

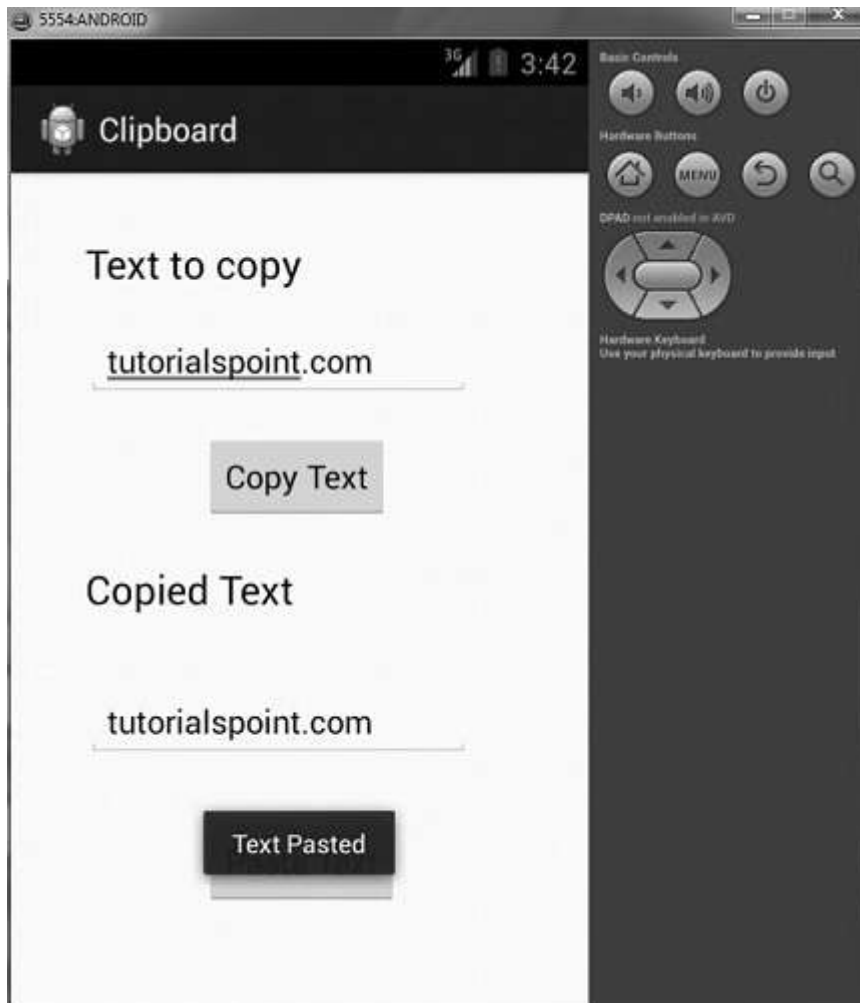
Let's try to run our Clipboard application we just modified. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything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



Now just enter any text in the **Text to copy** field and then select the **copy text** button. The following notification will be displayed which is shown below:



Now just press the paste button, and you will see the text which is copied is now pasted in the field of Copied Text. It is shown below:



35. CUSTOM FONTS

In android, you can define your own custom fonts for the strings in your application. You just need to download the required font from the internet, and then place it in assets/fonts folder.

After putting fonts in the assets folder under fonts folder, you can access it in your java code through Typeface class. First, get the reference of the text view in the code. Its syntax is given below:

```
TextView tx = (TextView)findViewById(R.id.textview1);
```

The next thing you need to do is to call static method of Typeface class **createFromAsset()** to get your custom font from assets. Its syntax is given below:

```
Typeface custom_font = Typeface.createFromAsset(getAssets(), "fonts/font name.ttf");
```

The last thing you need to do is to set this custom font object to your TextView Typeface property. You need to call **setTypeface()** method to do that. Its syntax is given below:

```
tx.setTypeface(custom_font);
```

Apart from these Methods, there are other methods defined in the Typeface class, that you can use to handle Fonts more effectively.

Sr.No	Method & description
1	create(String familyName, int style) Creates a Typeface object given a family name, and option style information.
2	create(Typeface family, int style) Creates a Typeface object that best matches the specified existing Typeface and the specified Style.
3	createFromFile(String path) Creates a new Typeface from the specified font file.

4	<p>defaultFromStyle(int style)</p> <p>Returns one of the default Typeface objects, based on the specified style.</p>
5	<p>getStyle()</p> <p>Returns the Typeface's intrinsic style attributes.</p>

Example:

Here is an example demonstrating the use of Typeface to handle CustomFont. It creates a basic application that displays a custom font that you have specified in the fonts file.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as CustomFonts under a package com.example.customfonts. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Download a font from internet and put it under assets/fonts folder.
3	Modify src/MainActivity.java file to add necessary code.
4	Modify the res/layout/activity_main to add respective XML components.
5	Modify the res/values/string.xml to add necessary string components.
6	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.customfonts/MainActivity.java**.

```

package com.example.customfonts;

import android.app.Activity;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView tx = (TextView)findViewById(R.id.hello);
        Typeface custom_font = Typeface.createFromAsset(getAssets(),
            "fonts/Erika Type.ttf");
        tx.setTypeface(custom_font);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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/hello"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="70dip"
    android:text="@string/hello_world" />

</LinearLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">CustomFonts</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.customfonts"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"

```

```
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.customfonts.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

Let's try to run our Custom Font 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 following Emulator window:



As you can see that the text that appeared on the AVD does not have a default android font, rather it has the custom font that you have specified in the fonts folder.

Note: You need to take care of the size and the character supported by the font, when using custom fonts.

36. DATA BACKUP

Android allows you to backup your application data to remote "cloud" storage, in order to provide a restore point for the application data and settings. You can only backup your application data. To access the other applications data, you need to root your phone.

In order to make a data backup application, you need to register your application with google backup service. This has been explained in the example. After registering, you have to specify its key in the AndroidManifest.XML

```
<application
    android:allowBackup="true"
    android:backupAgent="MyBackupPlace">

    <meta-data
        android:name="com.google.android.backup.api_key"
        android:value="AEdPqrEAAAAIEr1xFByGgNz2ywBeQb6TsmLpp5Ksh1PW-ZSexg"
    />
</application>
```

Android provides **BackUpAgentHelper** class to handle all the operations of data backup. To use this class, you have to extend your class with it. Its syntax is given below:

```
public class MyBackUpPlace extends BackupAgentHelper {

}
```

The persistent data that you want to backup is in either of the two forms. Either it could be SharedPreferences or it could be File. Android supports both types of backup in the respective classes of **SharedPreferencesBackupHelper** and **FileBackupHelper**.

In order to use **SharedPreferenceBackupHelper**, you need to instantiate its object with the name of your sharedPreferences File. Its syntax is given below:

```
static final String File_Name_Of_Preferences = "myPreferences";
SharedPreferencesBackupHelper helper = new
SharedPreferencesBackupHelper(this, File_Name_Of_Preferences);
```


The last thing you need to do is to call `addHelper` method by specifying the backup key string, and the helper object. Its syntax is given below:

```
addHelper(PREFS_BACKUP_KEY, helper);
```

The `addHelper` method will automatically add a helper to a given data subset to the agent's configuration.

Apart from these methods, there are other methods defined in the `BackupAgentHelper` class. They are defined below:

Sr.No	Method & description
1	<p>onBackup(ParcelFileDescriptor oldState, BackupDataOutput data, ParcelFileDescriptor newState)</p> <p>Run the backup process on each of the configured handlers.</p>
2	<p>onRestore(BackupDataInput data, int appVersionCode, ParcelFileDescriptor newState)</p> <p>Run the restore process on each of the configured handlers.</p>

The methods of the `SharedPreferencesBackUpHelper` class are listed below.

Sr.No	Method & description
1	<p>performBackup(ParcelFileDescriptor oldState, BackupDataOutput data, ParcelFileDescriptor newState)</p> <p>Backs up the configured <code>SharedPreferences</code> groups.</p>
2	<p>restoreEntity(BackupDataInputStream data)</p> <p>Restores one entity from the restore data stream to its proper shared preferences file store.</p>

Example:

The following example demonstrates the use of `BackupAgentHelper` class to create backup of your application data.

To experiment with this example, you need to run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Backup under a package com.example.backup. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Register your application with Google backup service.
3	Modify the AndroidManifest to add respective necessary key and other components.
4	Create backup agent class with the name you specify at AndroidManifest.XML
5	Run the application and verify the results.

Register you android application with google backup service. To do that, [visit this link](#). You must agree to the terms of service, and then enter the application package name. It is shown below:

Then click on Register with android backup service. It would give you your key, along with your AndroidManifest code to copy. Just copy the key. It is shown below:

Your key is:

```
AEdPqrEAAAIEr1xFBYGgNz2ywBeQb6TsmLpp5Ksh1PW-2Sxg
```

Once you copy the key, you need to write it in your AndroidManifest.XML file. Its code is given below:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```

package="com.example.backup"
android:versionCode="1"
android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:backupAgent="MyBackUpPlace"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.backup.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <meta-data
        android:name="com.google.android.backup.api_key"
        android:value="AEdPqrEAAAIEr1xFByGgNz2ywBeQb6TsmLpp5Ksh1PW-
        ZSexg" />

</application>

</manifest>

```

Here is the code of BackUpAgentHelper class. The name of the class should be the same as you specified in the backupAgent tag under application in AndroidManifest.XML

```
package com.example.backup;

import android.app.backup.BackupAgentHelper;
import android.app.backup.SharedPreferencesBackupHelper;

public class MyBackUpPlace extends BackupAgentHelper {

    static final String File_Name_Of_Preferences = "myPreferences";
    static final String PREFS_BACKUP_KEY = "backup";

    @Override
    public void onCreate() {
        SharedPreferencesBackupHelper helper = new
        SharedPreferencesBackupHelper(this,
        File_Name_Of_Preferences);
        addHelper(PREFS_BACKUP_KEY, helper);
    }
}
```

Test your BackupAgent

Once you've implemented your backup agent, you can test the backup and restore functionality with the following procedure, using bmgr.

Install your application on a suitable Android system image.

If you are using the emulator, create and use an AVD with Android 2.2 (API Level 8).

If you are using a device, the device must be running Android 2.2 or greater and must have built-in Google Play.

Ensure data backup is enabled

If using the emulator, you can enable backup with the following command from your SDK tools/ path:

```
adb shell bmgr enable true
```

If using a device, open the system Settings, select Privacy, then enable Back up my data and Automatic restore.

Performing backup

For testing purposes, you can also make a request with the following bmgr command:

```
adb shell bmgr backup your.package.name
```

Initiate a backup operation by typing the following command.

```
adb shell bmgr run
```

This forces the Backup Manager to perform all backup requests that are in its queue.

Uninstall and reinstall your application

Uninstall the application with the following command:

```
adb uninstall your.package.name
```

Then reinstall the application and verify the results.

37. DEVELOPER TOOLS

The android developer tools lets you create interactive and powerful application for android platform. The tools can be generally categorized into two types.

- SDK tools
- Platform tools

SDK tools

SDK tools are generally platform independent and are required no matter which android platform you are working on. When you install the Android SDK into your system, these tools get automatically installed. The list of SDK tools has been given below:

Sr.No	Tool & description
1	android This tool lets you manage AVDs, projects, and the installed components of the SDK.
2	ddms This tool lets you debug Android applications.
3	Draw 9-Patch This tool allows you to easily create a NinePatch graphic using a WYSIWYG editor.
4	emulator This tools let you test your applications without using a physical device.
5	mksdcard Helps you create a disk image (external sdcard storage) that you can use with the emulator.
6	proguard Shrinks, optimizes, and obfuscates your code by removing unused

	code.
7	<p>sqlite3</p> <p>Lets you access the SQLite data files created and used by Android applications.</p>
8	<p>traceview</p> <p>Provides a graphical viewer for execution logs saved by your application.</p>

We will discuss three important tools here that are android, ddms and sqlite3.

Android

Android is a development tool that lets you perform these tasks:

- Manage Android Virtual Devices (AVD)
- Create and update Android projects
- Update your sdk with new platform add-ons and documentation

```
android [global options] action [action options]
```

DDMS

DDMS stands for Dalvik Debug Monitor Server that provides many services on the device. The service could include message formation, call spoofing, capturing screenshot, exploring internal threads and file systems etc.

Running DDMS

From eclipse click on **Window, Open Perspective, Other ... DDMS**. Or simply look on the left most top corner and click on ddms.

How it works

In android, each application runs in its own process and each process run in the virtual machine. Each VM exposes a unique port that a debugger can attach to.

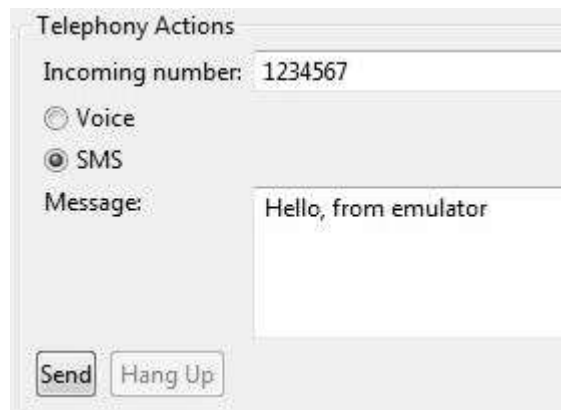
When DDMS starts, it connects to adb. When a device is connected, a VM monitoring service is created between adb and DDMS, which notifies DDMS when a VM on the device is started or terminated.

Using DDMS

You can use DDMS for many tasks. For example, here we are using it to make sms, make call, and capture screenshot.

Making SMS

In the DDMS, select the Emulator Control tab. In the emulator control tab, click on SMS and start typing the SMS and then the incoming number. It is shown in the picture below.

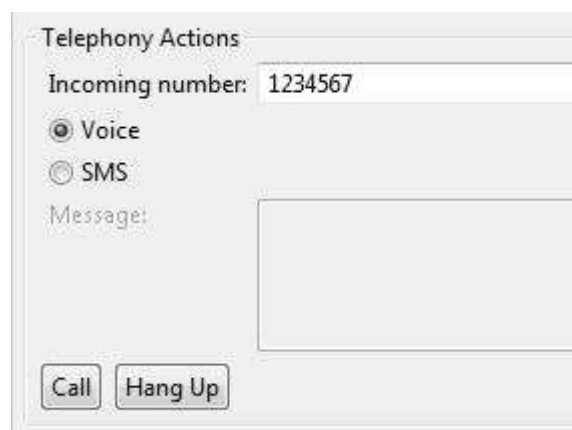


Now click on send button, and you will see an sms notification in the emulator window. It is shown below:

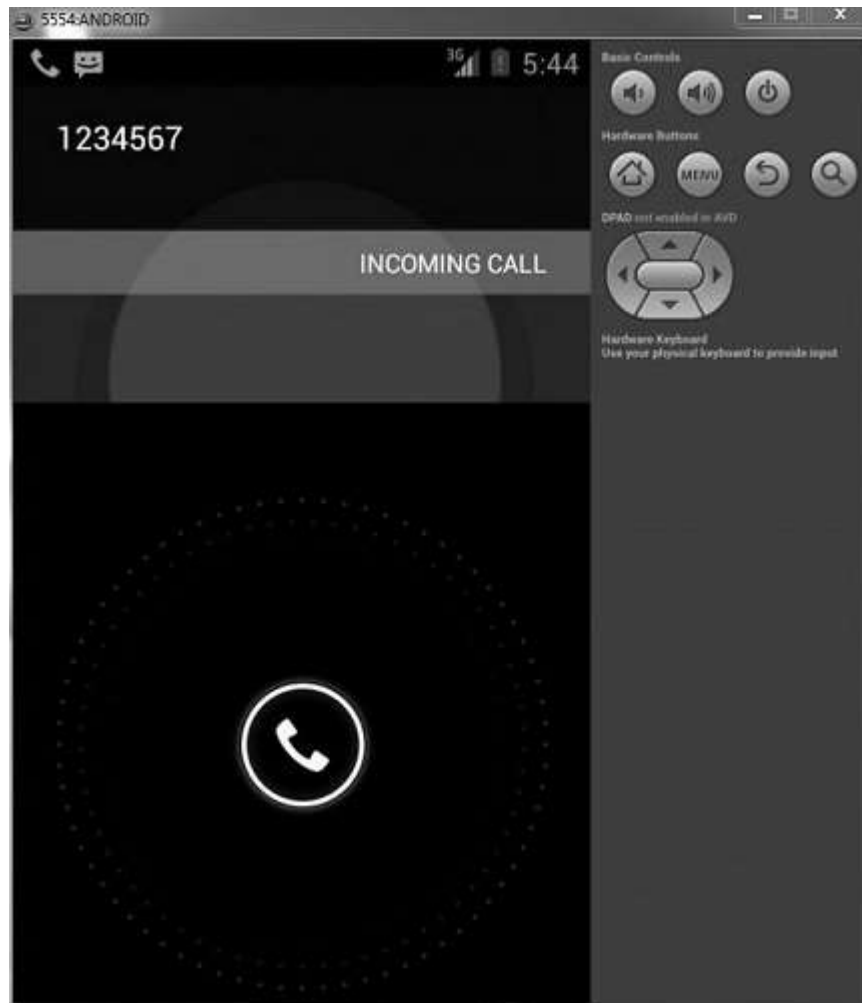


Making Call

In the DDMS, select the Emulator Control tab. In the emulator control tab, click on voice and then start typing the incoming number. It is shown in the picture below:

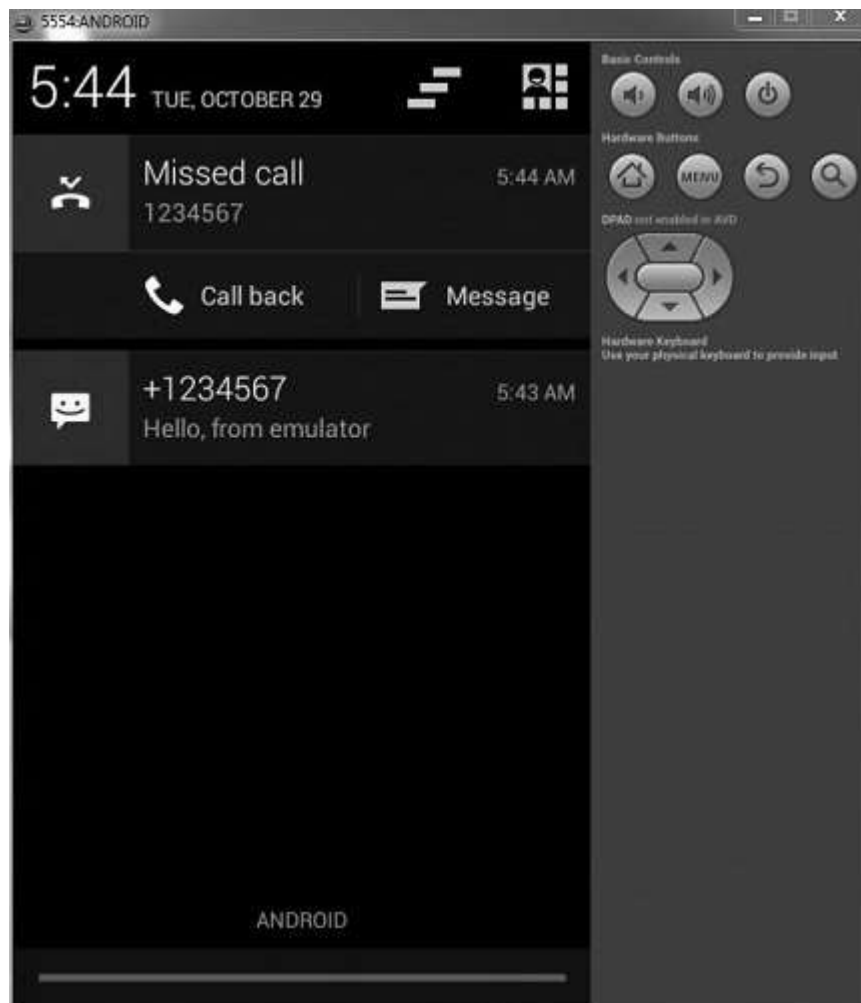


Now click on the call button to make a call to your emulator. It is shown below:



Now click on hang-up in the eclipse window to terminate the call.

The fake sms and call can be viewed from the notification by just dragging the notification window to the center using mouse. It is shown below:



Capturing ScreenShot

You can also capture screenshot of your emulator. For this look for the camera icon on the right side under Devices tab. Just point your mouse over it and select it.

As soon as you select it, it will start the screen capturing process and will capture whatever screen of the emulator is currently active. It is shown below:



The eclipse orientation can be changed using Ctrl + F11 key. Now you can save the image or rotate it and then select done to exit the screen capture dialog.

Sqlite3

Sqlite3 is a command line program which is used to manage the SQLite databases created by Android applications. The tool also allow us to execute the SQL statements on the fly.

There are two way through which you can use SQLite, either from remote shell or you can use locally.

Use Sqlite3 from a remote shell.

Enter a remote shell by entering the following command:

```
adb [-d|-e|-s {}] shell
```

From a remote shell, start the sqlite3 tool by entering the following command:

```
sqlite3
```

Once you invoke sqlite3, you can issue sqlite3 commands in the shell. To exit and return to the adb remote shell, enter exit or press CTRL+D.

Using Sqlite3 directly

Copy a database file from your device to your host machine.

```
adb pull
```

Start the sqlite3 tool from the /tools directory, specifying the database file:

```
sqlite3
```

Platform tools

The platform tools are customized to support the features of the latest android platform.

The platform tools are typically updated every time you install a new SDK platform. Each update of the platform tools is backward compatible with older platforms.

Some of the platform tools are listed below:

- Android Debug bridge (ADB)
- Android Interface definition language (AIDL)
- aapt, dexdump, and dex etc.

38. EMULATOR

Emulator lets you emulate the real device with all its functionalities without purchasing the real device. Android emulator lets you emulate different android configurations by creating android virtual devices.

We are going to explore different functionalities in the emulator that are present in the real android device in this chapter.

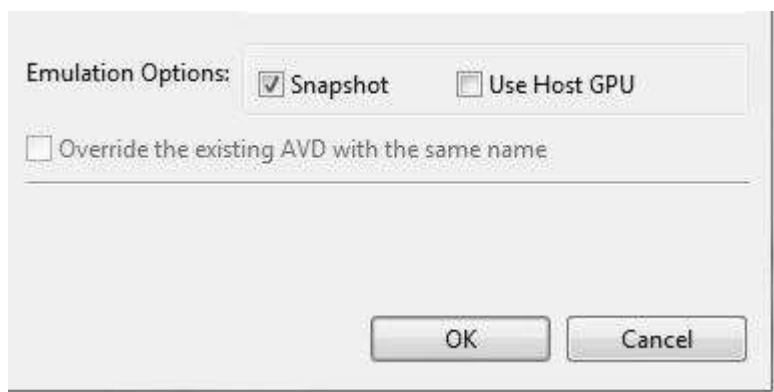
Creating AVD

If you want to emulate a real device, first create an AVD with the same device configurations as real device, then launch this AVD from AVD manager.

Creating Snapshots

Creating snapshots mean saving an emulator state to a file that enables the emulator to be started quickly the next time you try to launch it. One of the biggest advantage of creating snapshots is that it saves the boot up time.

In order to create snapshot, check mark the option of snapshot while creating your AVD. It is shown below:



The first time you launch the emulator, it will take the usual time of loading. But when you close it and start it again, you will see a considerable amount of time reduction in appearing of emulator.

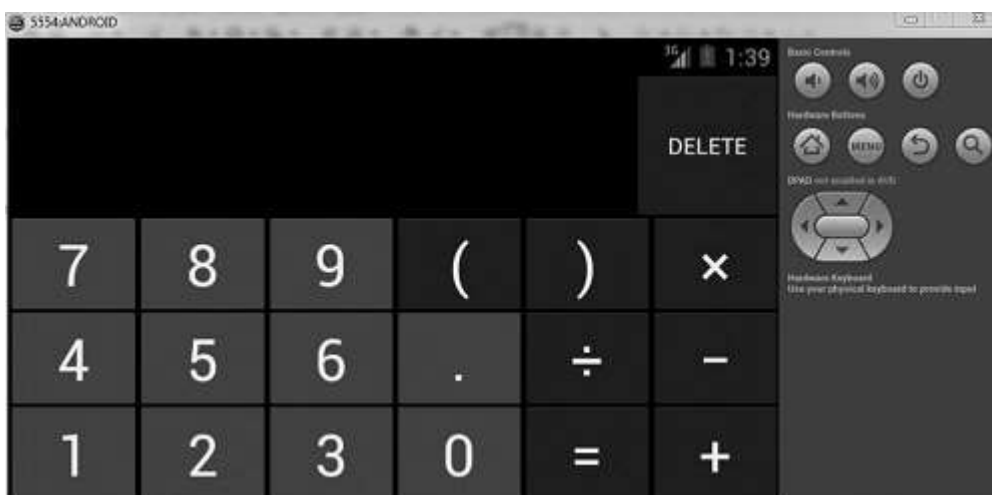
Changing Orientation

Usually by default when you launch the emulator, its orientation is vertical, but you can change its orientation by pressing Ctrl+F11 key from keyboard.

First launch the emulator. It is shown in the picture below:



Once it is launched, press **Ctrl+F11** key to change its orientation. It is shown below:



Emulator Commands.

Apart from just orientation commands, there are other very useful commands of emulator that you should keep in mind while using emulator. They are listed below:

Sr.No	Command & description
1	Home Shifts to main screen
2	F2 Toggles context sensitive menu
3	F3 Brings out call log
4	F4 End call
5	F5 Search
6	F6 Toggle trackball mode
7	F7 Power button
8	F8 Toggle data network
9	Ctrl+F5 Ring Volume up
10	Ctrl+F6

	Ring Volume down
--	------------------

Emulator - Sending SMS

You can emulate sending SMS to your emulator. There are two ways to do that. You can do that from DDMS which can be found in Eclipse, or from Telnet (Network utility found in windows).

Sending SMS through Telnet.

Telnet is not enabled by default in windows. You have to enable it to use it. Once enabled you can go to command prompt and start telnet by typing telnet.

In order to send SMS, note down the AVD number which can be found on the title bar of the emulator. It could be like this 5554 etc. etc. Once noted, type this command in command prompt.

```
telnet localhost 5554
```

Press enter when you type the command. It is shown below in the figure.



```
C:\Users>telnet localhost 5554
```

You will see that you are now connected to your emulator. Now type this command to send message.

```
sms send +1234567 your sms goes here
```

Once you type this command, hit enter. Now look at the AVD. You will receive a notification displaying that you got a new text message. It is shown below:



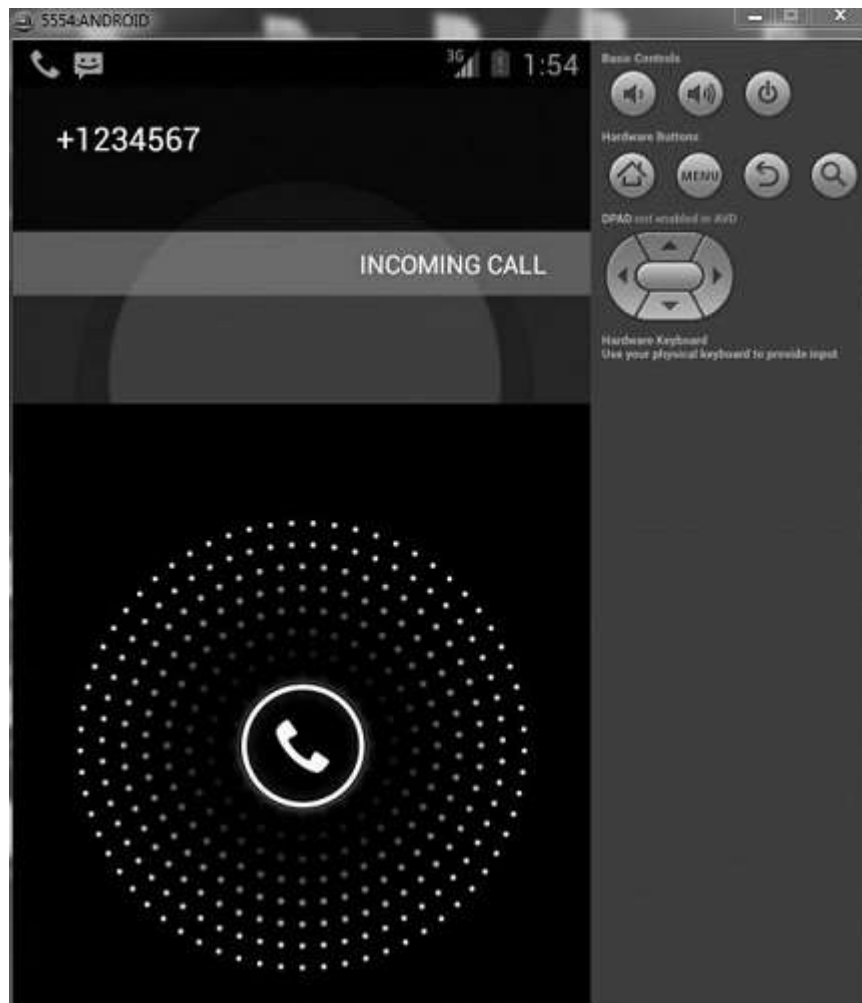
Emulator - Making Call

You can easily make phone calls to your emulator using telnet client. You need to connect to your emulator from telnet. It is discussed in the sending sms topic above.

After that you will type this command in the telnet window to make a call. Its syntax is given below:

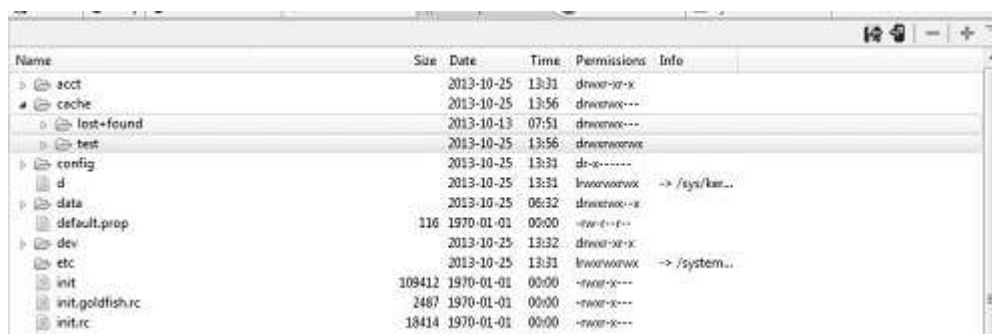
```
gsm call +1234567
```

Once you type this command, hit enter. Now look at the AVD. You will receive a call from the number you put in the command. It is shown below:



Emulator - Transferring files

You can easily transfer files into the emulator and vice versa. To do that, you need to select the DDMS utility in Eclipse. After that select the file explorer tab. It is shown below:



Browse through the explorer and make new folder, view existing contents etc. etc.

39. FACEBOOK INTEGRATION

Android allows your application to connect to Facebook and share data or any kind of updates on Facebook. This chapter is about integrating Facebook into your application.

There are two ways through which you can integrate Facebook and share something from your application. These ways are listed below:

- Facebook SDK
- Intent Share

Integrating Facebook SDK

This is the first way of connecting with Facebook. You have to register your application and then receive some Application Id, and then you have to download the Facebook SDK and add it to your project. The steps are listed below:

Generating application signature

You have to generate a key signature, but before you generate it, make sure you have SSL installed, otherwise you have to download SSI. It can be downloaded [here](#).

Now open command prompt and redirect to your java jre folder. Once you reach there, type this command exactly. You have to replace the path in the inverted commas with your keystore path which you can find in eclipse by selecting the window tab and selecting the preferences tab and then selecting the build option under android from left side.

```
keytool -exportcert -alias androiddebugkey -keystore "your path" |  
openssl sha1 -binary | openssl base64
```

Once you enter it, you will be prompted for password. Give android as the password and then copy the key that is given to you. It is shown in the image below:

```
C:\Program Files\Java\jre7\bin>keytool -exportcert -alias androiddebugkey -keystore "C:\Users\      \.android\debug.keystore" |  
openssl sha1 -binary | openssl base64  
Enter keystore password: android  
gr0ix/+LESdxzlWE
```

Registering your application

Now create a new Facebook application at developers.facebook.com/apps and fill all the information. It is shown below:

The screenshot shows the 'Create New App' form with the following details:

- App Name: TutorialsPoint (Valid)
- App Namespace: sampletutorialspoint
- App Category: Education (Choose a sub-category)
- Buttons: Continue, Cancel
- Footer: By proceeding, you agree to the Facebook Platform Policies

Now move to the native android app section and fill in your project and class name and paste the hash that you copied in step 1. It is shown below:

The screenshot shows the 'Native Android App' configuration form with the following details:

- Package Name: com.example.facebook
- Class Name: com.example.facebook.MainActivity
- Key Hashes: gr0ix/+LESdxz1WE
- Facebook Login: Disabled
- Deep Linking: Disabled

If everything works fine, you will receive an application ID with the secret. Just copy the application id and save it somewhere. It is shown in the image below:

The screenshot shows the application details for 'TutorialsPoint' with the following information:

- App ID: 577787852259051
- App Secret: 57ab9abdbf2f9a6fb1f081ba8a4eb093 (reset)
- Status: This app is in Sandbox Mode (Only visible to Admins, Developers and Testers)

Downloading SDK and integrating it

Download Facebook sdk [here](#). Import this into eclipse. Once imported, right click on your facebook project and click on properties. Click on android, click on add button and select Facebook sdk as the project. Click ok.

Creating facebook login application

Once everything is complete, you can run the samples, that comes with SDK or create your own application. To login, you need to call **openActiveSession** method and implement its callback. Its syntax is given below:

```

// start Facebook Login
Session.openActiveSession(this, true, new Session.StatusCallback() {

    // callback when session changes state
    public void call(Session session, SessionState state, Exception
exception)
    {
        if (session.isOpened()) {
            // make request to;2 the /me API
            Request.executeMeRequestAsync(session, new Request.
GraphUserCallback() {

                // callback after Graph API response with user object
                @Override
                public void onCompleted(GraphUser user, Response response) {
                    if (user != null) {
                        TextView welcome = (TextView) findViewById(R.id.welcome);
                        welcome.setText("Hello " + user.getName() + "!");
                    }
                }
            });
        }
    }
}

```

Intent share

Intent share is used to share data between applications. In this strategy, we will not handle the SDK stuff, but let the Facebook application handle it. We will simply call the facebook application and pass the data to share. This way, we can share something on Facebook.

Android provides intent library to share data between activities and applications. In order to use it as share intent, we have to specify the type of the share intent to **ACTION_SEND**. Its syntax is given below:

```

Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);

```

Next thing you need is to define the type of data to pass, and then pass the data. Its syntax is given below:

```
shareIntent.setType("text/plain");
shareIntent.putExtra(Intent.EXTRA_TEXT, "Hello, from tutorialspoint");
startActivity(Intent.createChooser(shareIntent, "Share your thoughts"));
```

Apart from these methods, there are other methods available that allows intent handling. They are listed below:

Sr.No	Method & description
1	addCategory(String category) This method adds a new category to the intent.
2	createChooser(Intent target, CharSequence title) Convenience function for creating a ACTION_CHOOSER Intent.
3	getAction() This method retrieve the general action to be performed, such as ACTION_VIEW.
4	getCategories() This method returns the set of all categories in the intent.nt and the current scaling event.
5	putExtra(String name, int value) This method add extended data to the intent.
6	toString() This method returns a string containing a concise, human-readable description of this object.

Example:

Here is an example demonstrating the use of IntentShare to share data on Facebook. It creates a basic application that allows you to share some text on Facebook.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as IntentShare under a package com.example.intentshare. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.intentshare/MainActivity.java**.

```
package com.example.intentshare;

import java.io.File;
import java.io.FileOutputStream;

import com.example.intentshare.R;

import android.app.Activity;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.view.Menu;
```



```
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private ImageView img;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        img = (ImageView) findViewById(R.id.imageView1);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void open(View view){
        Intent shareIntent = new Intent();
        shareIntent.setAction(Intent.ACTION_SEND);
        shareIntent.setType("text/plain");
        shareIntent.putExtra(Intent.EXTRA_TEXT, "Hello, from
        tutorialspoint");
        startActivity(Intent.createChooser(shareIntent, "Share your
        thoughts"));

    }

}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="98dp"
        android:layout_marginTop="139dp"
        android:onClick="open"
        android:src="@drawable/tp" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="48dp"
        android:text="@string/tap"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

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

    <string name="app_name">IntentShare</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="tap">Tap the button to share something</string>

</resources>
```

Following is the content of **AndroidManifest.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.intentshare"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.intentshare.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />


                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

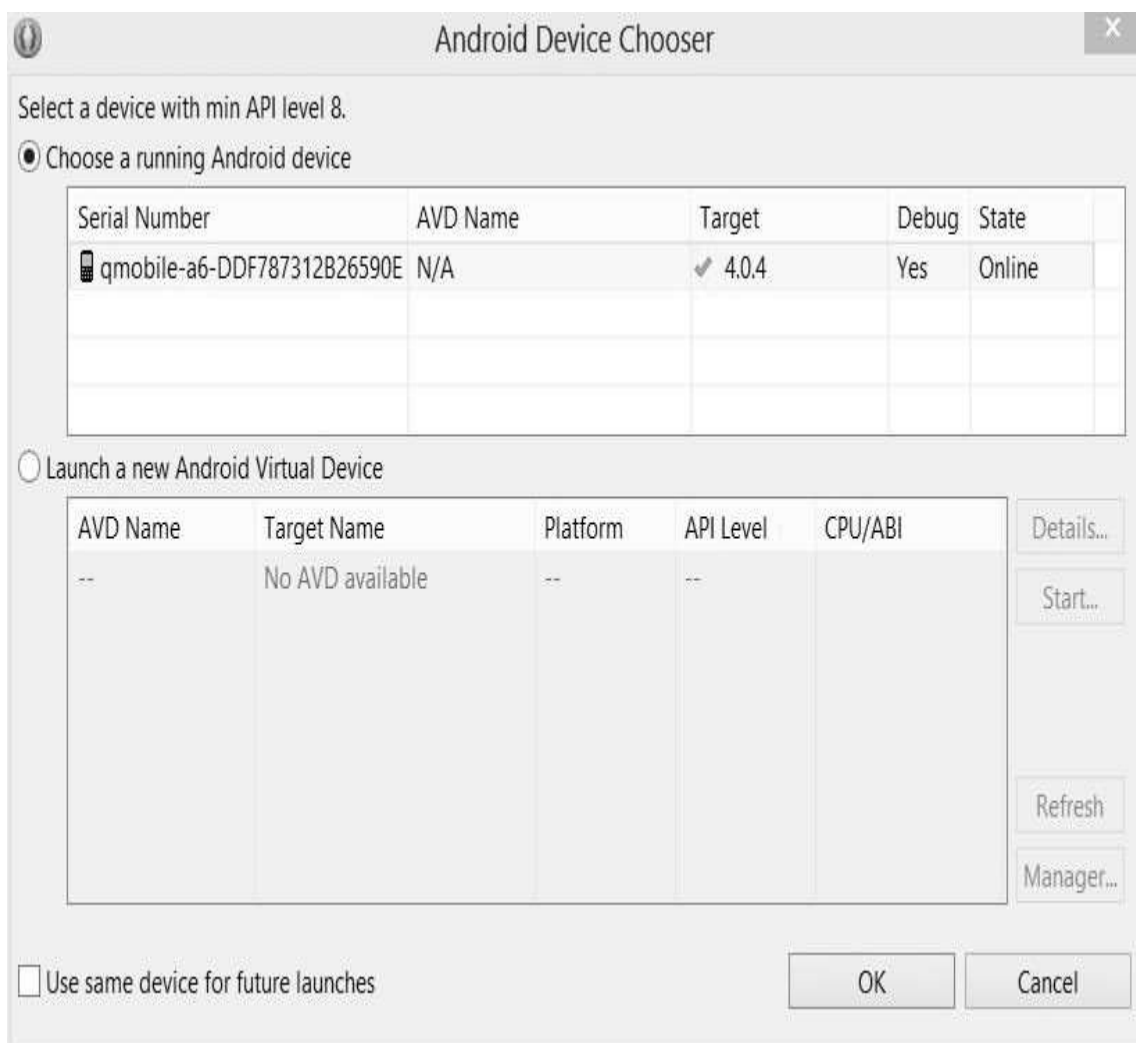
```

        </intent-filter>
    </activity>
</application>

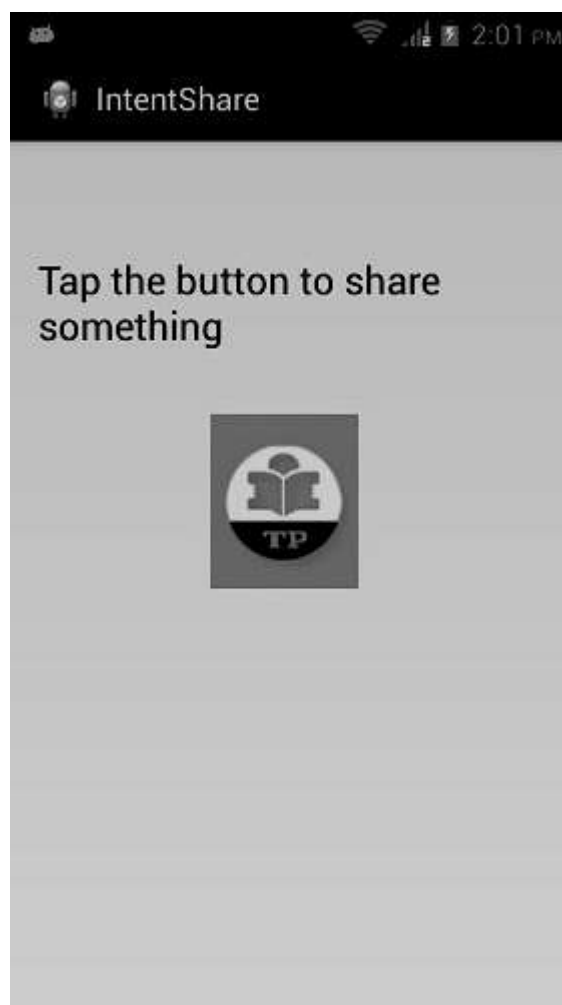
</manifest>

```

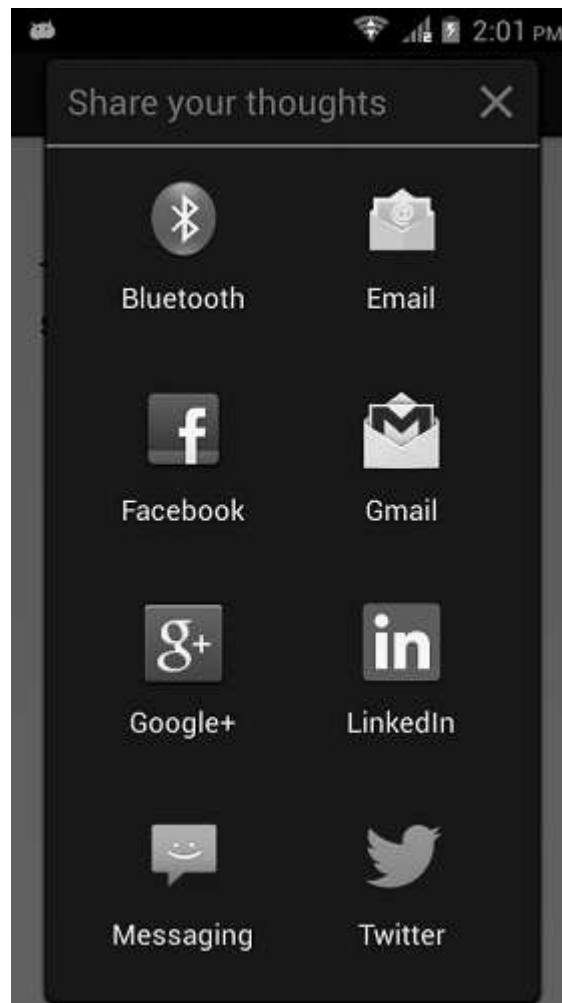
Let's try to run your IntentShare application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



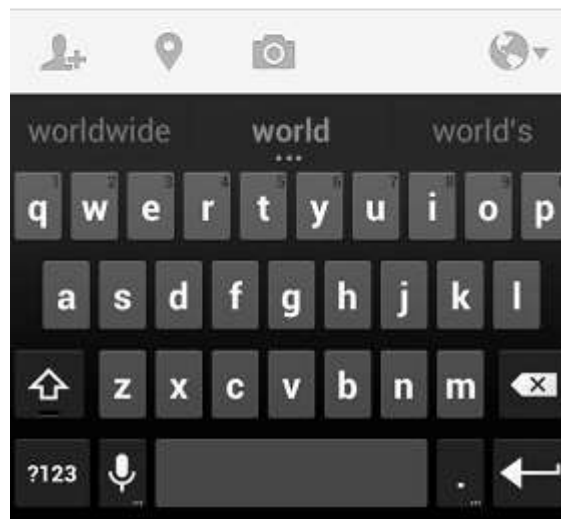
Select your mobile device as an option and then check your mobile device which will display your default screen:



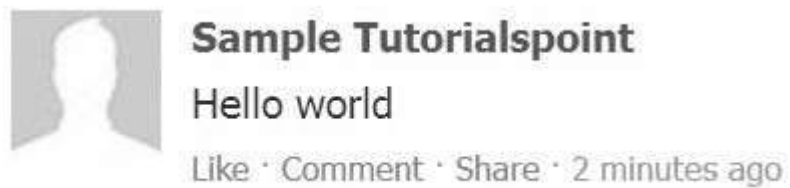
Now just tap on the image logo and you will see a list of share providers.



Now just select Facebook from that list and then write any message. It is shown in the image below:



Now just select the post button and then it would be posted on your wall. It is shown below:



40. GESTURES

Android provides special types of touch screen events such as pinch, double tap, scrolls, long presses and flinch. These are all known as gestures.

Android provides `GestureDetector` class to receive motion events and tell us that these events correspond to gestures or not. To use it, you need to create an object of `GestureDetector` and then extend another class with **`GestureDetector.SimpleOnGestureListener`** to act as a listener and override some methods. Its syntax is given below:

```
GestureDetector myG;
myG = new GestureDetector(this,new Gesture());

class Gesture extends GestureDetector.SimpleOnGestureListener{
public boolean onSingleTapUp(MotionEvent ev) {
}
public void onLongPress(MotionEvent ev) {
}
public boolean onScroll(MotionEvent e1, MotionEvent e2, float
distanceX,
float distanceY) {
}
public boolean onFling(MotionEvent e1, MotionEvent e2, float
velocityX,
float velocityY) {
}
}
}
```

Handling Pinch Gesture

Android provides **`ScaleGestureDetector`** class to handle gestures like pinch etc. In order to use it, you need to instantiate an object of this class. Its syntax is as follow:


```
ScaleGestureDetector SGD;
SGD = new ScaleGestureDetector(this,new ScaleListener());
```

The first parameter is the context and the second parameter is the event listener. We have to define the event listener and override a function **OnTouchEvent** to make it working. Its syntax is given below:

```
public boolean onTouchEvent(MotionEvent ev) {
    SGD.onTouchEvent(ev);
    return true;
}
private class ScaleListener extends
ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        float scale = detector.getScaleFactor();
        return true;
    }
}
```

Apart from the pinch gestures, there are other methods available that notify more about touch events. They are listed below:

Sr.No	Method & description
1	getTime() This method gets the event time of the current event being processed.
2	getFocusX() This method gets the X coordinate of the current gesture's focal point.
3	getFocusY() This method gets the Y coordinate of the current gesture's focal point.
4	getTimeDelta() This method returns the time difference in milliseconds between the previous accepted scaling event and the current scaling event.

5	isInProgress() This method returns true if a scale gesture is in progress.
6	onTouchEvent(MotionEvent event) This method accepts MotionEvent and dispatches events when appropriate.

Example:

Here is an example demonstrating the use of ScaleGestureDetector class. It creates a basic application that allows you to zoom in and out through pinch.

To experiment with this example, you can run this on an actual device or in an emulator with touch screen enabled.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Gestures under a package com.example.gestures. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.gestures/MainActivity.java**.

```
package com.example.gestures;

import android.app.Activity;
import android.graphics.Matrix;
```

```
import android.os.Bundle;
import android.view.Menu;
import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView img;
    private Matrix matrix = new Matrix();
    private float scale = 1f;
    private ScaleGestureDetector SGD;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        img = (ImageView)findViewById(R.id.imageView1);
        SGD = new ScaleGestureDetector(this,new ScaleListener());
    }

    @Override
    public boolean onTouchEvent(MotionEvent ev) {
        SGD.onTouchEvent(ev);
        return true;
    }

    private class ScaleListener extends ScaleGestureDetector.
SimpleOnScaleGestureListener {
    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        scale *= detector.getScaleFactor();
        scale = Math.max(0.1f, Math.min(scale, 5.0f));
        matrix.setScale(scale, scale);
        img.setImageMatrix(matrix);
    }
}
```

```

        return true;
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <ImageView
        android:id="@+id/imageView1"

```

```

        android:layout_width="match_parent"
        android:layout_height="match_parent"

        android:layout_below="@+id/textView1"
        android:scaleType="matrix"
        android:src="@android:drawable/sym_def_app_icon" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">Gestures</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Pinch to zoom in or out!</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.gestures"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

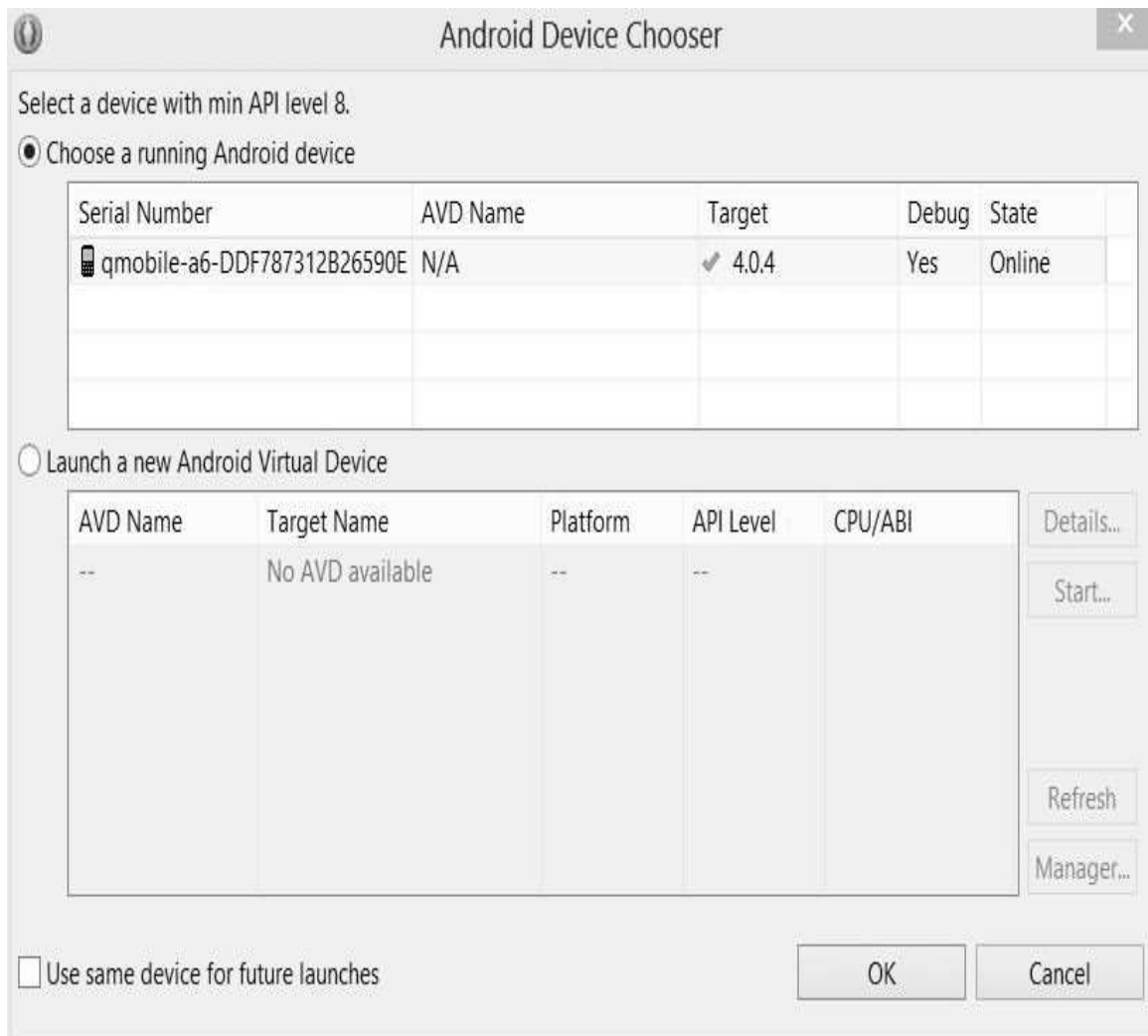
```

```
<activity
  android:name="com.example.gestures.MainActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

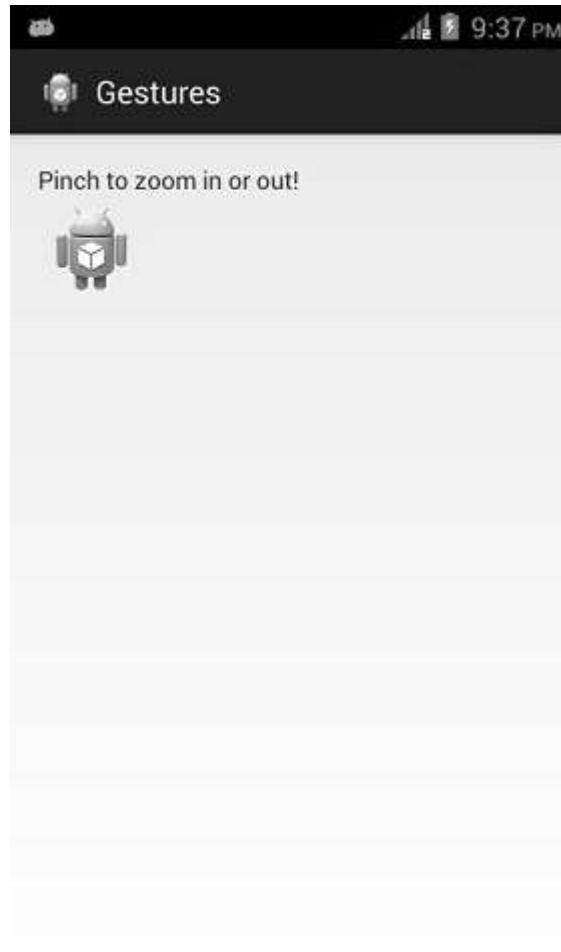
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
</application>

</manifest>
```

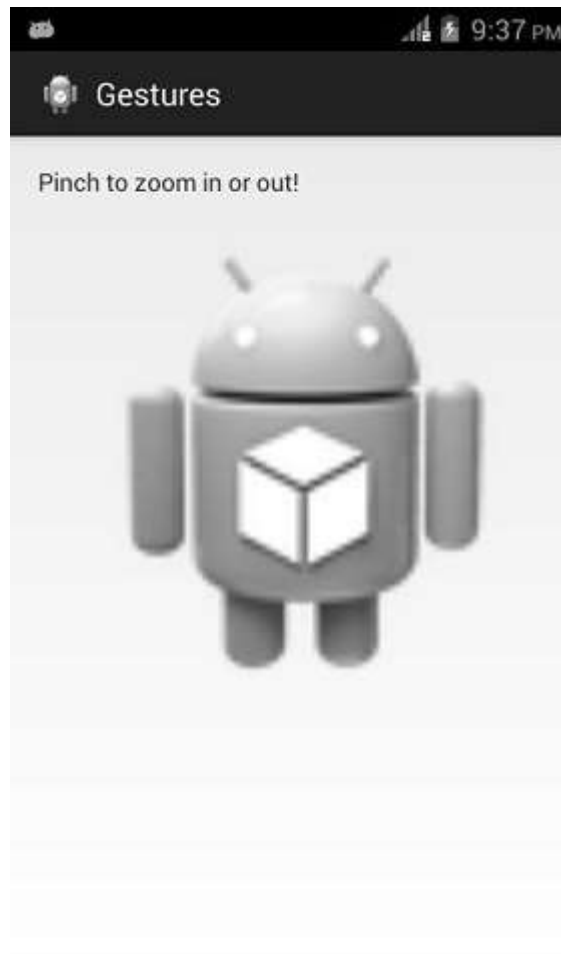
Let's try to run your Gestures application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



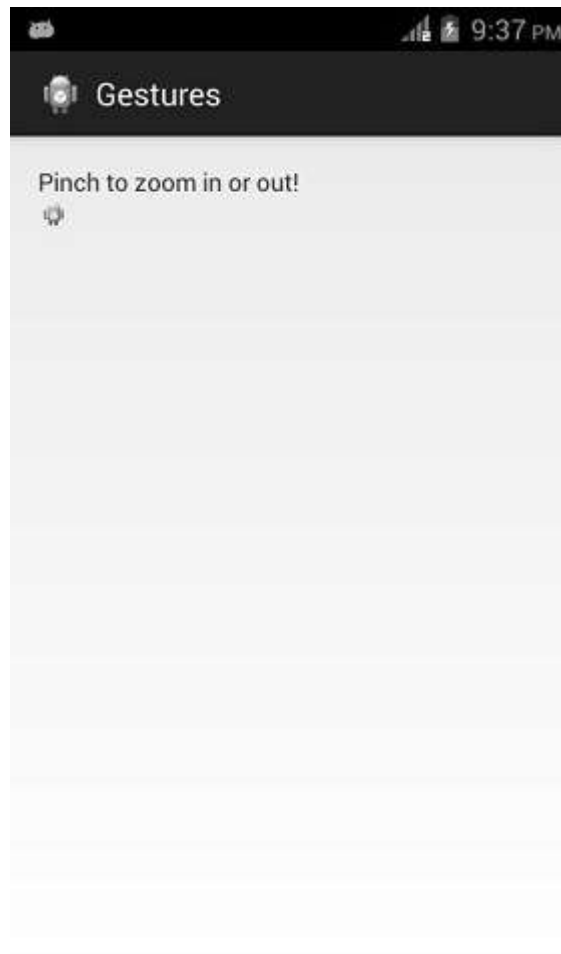
Select your mobile device as an option and then check your mobile device which will display your default screen:



Now just place two fingers over android screen, and separate them apart and you will see that the android image is zooming. It is shown in the image below:



Now again place two fingers over android screen, and try to close them and you will see that the android image is now shrinking. It is shown in the image below:



41. GOOGLE MAPS

Android allows us to integrate google maps in our application. You can show any location on the map, or can show different routes on the map etc. etc. You can also customize the map according to your choices.

Adding Google Map

Google provides this facility using google play services library which you have to download externally. After downloading, you have to integrate it with your project. In the end you have to integrate your application with google via google console. This is completely discussed in the example.

Google Map - Activity file

Google provides GoogleMap and MapFragment api to integrate map in your android application. In order to use GoogleMap, you have to create an object of GoogleMap and get the reference of map from the xml layout file. Its syntax is given below:

```
GoogleMap googleMap;  
  
googleMap = ((MapFragment)  
getFragmentManager().findFragmentById(R.id.map)).getMap();
```

Google Map - Layout file

Now you have to add the map fragment into xml layout file. Its syntax is given below:

```
<fragment  
    android:id="@+id/map"  
    android:name="com.google.android.gms.maps.MapFragment"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

Google Map - AndroidManifest file

The next thing you need to do is to add some permissions along with the Google Map API key in the AndroidManifest.XML file. Its syntax is given below:

```
<!--Permissions-->
```

```

<uses-permission    android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"
/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>

<!--Google MAP API key-->

<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />

```

Customizing Google Map

You can easily customize google map from its default view, and change it according to your demand.

Adding Marker

You can place a marker with some text over it displaying your location on the map. It can be done by via **addMarker()** method. Its syntax is given below:

```

final LatLng TutorialPoint = new LatLng(21, 57);
Marker TP = googleMap.addMarker(new
MarkerOptions().position(TutorialPoint).title("TutorialPoint"));

```

Changing Map Type

You can also change the type of the MAP. There are four different types of map and each give different view of the map. These types are Normal, Hybrid, Satellite and terrain. You can use them as below:

```

googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

```

Enable/Disable zoom

You can also enable or disable the zoom gestures in the map by calling the **setZoomControlsEnabled(boolean)** method. Its syntax is given below:

```
googleMap.getUiSettings().setZoomGesturesEnabled(true);
```

Apart from these customization, there are other methods available in the GoogleMap class that helps you more to customize the map. They are listed below:

Sr.No	Method & description
1	addCircle(CircleOptions options) This method adds a circle to the map.
2	addPolygon(PolygonOptions options) This method adds a polygon to the map.
3	addTileOverlay(TileOverlayOptions options) This method adds tile overlay to the map.
4	animateCamera(CameraUpdate update) This method moves the map according to the update with an animation.
5	clear() This method removes everything from the map.
6	getMyLocation() This method returns the currently displayed user location.
7	moveCamera(CameraUpdate update) This method repositions the camera according to the instructions defined in the update.
8	setTrafficEnabled(boolean enabled) This method toggles the traffic layer on or off.

9	snapshot(GoogleMap.SnapshotReadyCallback callback) This method takes a snapshot of the map.
10	stopAnimation() This method stops the camera animation if there is one in progress

Example:

Here is an example demonstrating the use of GoogleMap class. It creates a basic M application that allows you to navigate through the map.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	Integrate google maps in your application.
2	You will use Eclipse IDE to create an Android application and name it as GoogleMaps under a package com.example.googlemaps. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
3	Modify src/MainActivity.java file to add necessary code.
4	Modify the res/layout/activity_main to add respective XML components.
5	Modify AndroidManifest.xml to add necessary internet permission.
6	Run the application and choose a running android device and install the application on it and verify the results.

Integrating Google Maps

Integrating google maps in your application basically consists of these 4 steps.

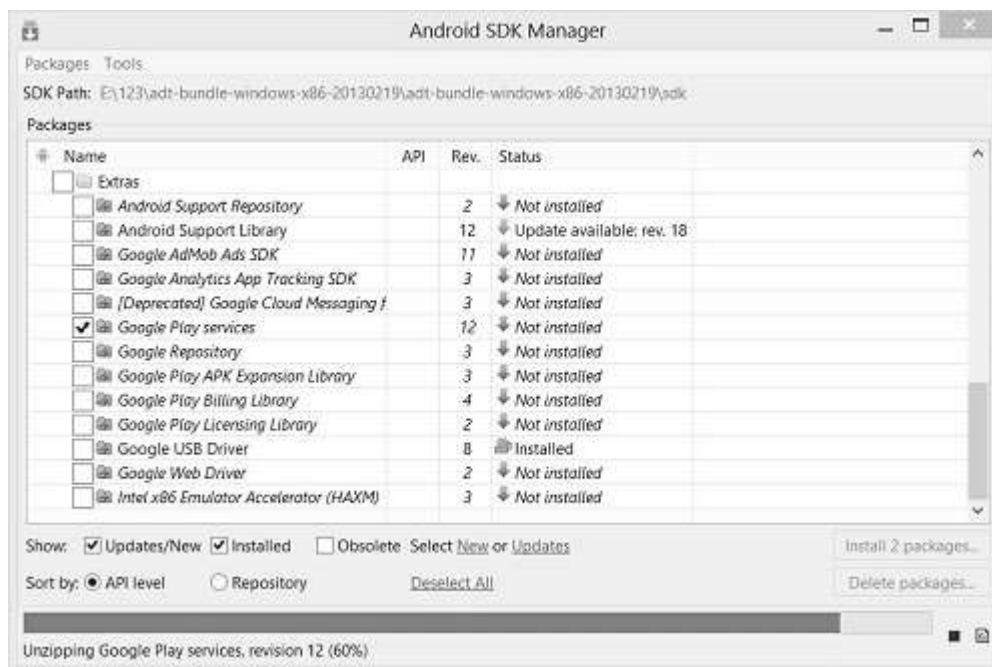
- Download and configure. Google Play Services SDK
- Obtain API key from google console
- Specify Android Manifest settings

Download and configure. Google Play Services SDK

Install Google services SDK

Open your SDK manager in the eclipse by clicking the Window and then selecting the Android SDK manager.

Navigate to the extras tab and select the Google play services and click on install this package. It would be like this.

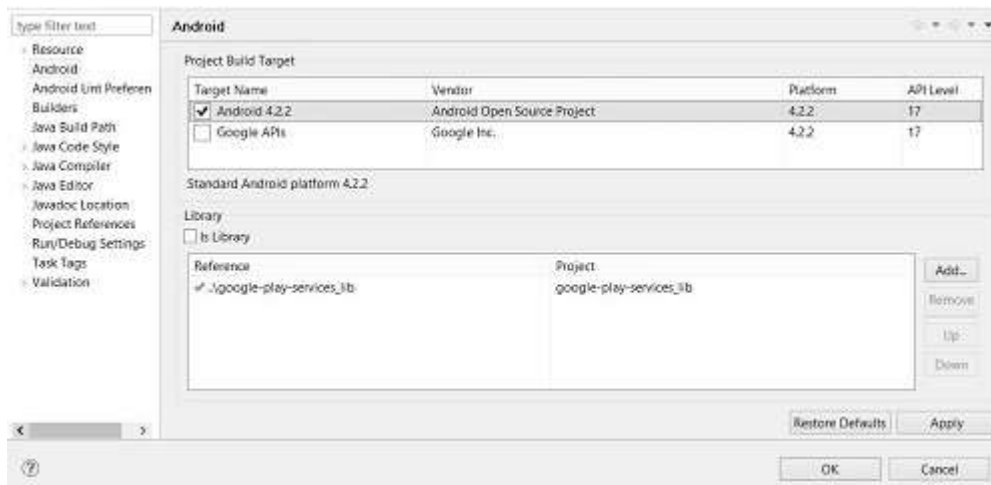


Import SDK to eclipse

After you download the SDK, click on file tab and select import option. Select existing android application code and press ok. Browse to your android folder and then sdk folder. In sdk folder expand extras folder. Expand google folder and select google play services.

Configure your project with SDK

After you import the SDK, you have to add it into your project. For this, right click on your eclipse project and select properties. Select android from left tab and then select add from right below panel and add the project. It would be like this



Obtaining the API key

This part is further divided into two steps. First you have to get an SHA1 fingerprint key from your pc and then you have to get map API key from google console.

Getting Certificate from KeyTool

You need to get a certificate key because you have to provide it to google console in order to get your API key for map.

Open your command prompt and move to the path where your java jre has been placed. Now type this command.

```
keytool -list -v -alias androiddebugkey -keystore %%Your path%% -storepass android -keypass android
```

Replace the percentage part of the command with the path which you will copy from by selecting the window tab and selecting the preferences tab and then selecting the build option under android from left side.

Copy the default debug keystore path and replace it in the command and hit enter. The following result would appear.

```
C:\Program Files (x86)\Java\jre7\bin>keytool -list -v -alias androiddebugkey -keystore C:\Users\...\.android\debug.keystore -storepass android -keypass android
Alias name: androiddebugkey
Creation date: Oct 6, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 352f55f0
Valid from: ...
Certificate fingerprints:
MD5: 9D:49:75:15:1A:FC:1E:23:98:62:C8:29:72:EC:D2:71
SHA1: AB:A9:D3:3E:92:D0:FE:4E:9D:21:87:8D:0D:61:18:8F:D8:91:E8:62:7D:3B:11:C9:AD:5E:CC:05:02:18:D9
SHA256: 14:8F:B7:40:BE:4B:83:2B:A9:13:CC:18:37:04:BB:12:45:F1:1E:B4:62:7D:3B:11:C9:AD:5E:CC:05:02:18:D9
Signature algorithm name: SHA256withRSA
Version: 3
Extensions:
```


Copy the SHA1 key because you need it in the next step.

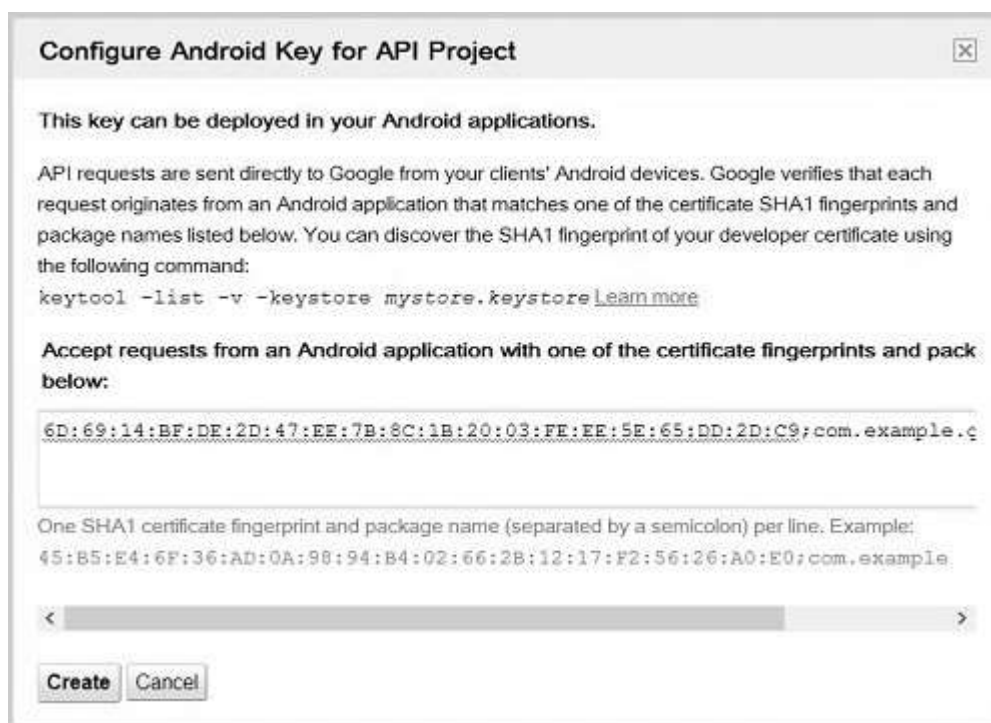
Getting key from Google Console

Open [Google Console](#) and sign in by clicking a new project.

Click on services from the left tab and then navigate to the Google Maps Android API v2. You have to turn them on like this



Now again go to the left tab and select API access. And click on create new android key. Now paste the key that you copied and put a semicolon and paste your project name and click create. It would be like this.



Now copy the API key that has been given to you by android, because you have to paste it into your manifest file.

Specify Android Manifest Settings

The final step is to add the API key to your application. Open your manifest file and place this code right before closing the application tag.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
```

```
android:value="API_KEY"/>
```

In the second line replace API_KEY with your api key and you are done. You need to add some permissions in your manifest too, which are given below in the manifest file.

Adding Google Maps to your application.

Following is the content of the modified main activity file **src/com.example.googlemaps/MainActivity.java**.

```
package com.example.googlemaps;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import android.os.Bundle;
import android.app.Activity;
import android.widget.Toast;

public class MainActivity extends Activity {
    static final LatLng TutorialPoint = new LatLng(21, 57);
    private GoogleMap googleMap;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        try {
            if (googleMap == null) {
                googleMap = ((MapFragment) getFragmentManager().
                    findFragmentById(R.id.map)).getMap();
            }
            googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
        }
    }
}
```

```

        Marker TP = googleMap.addMarker(new MarkerOptions().
            position(TutorialsPoint).title("TutorialsPoint"));

    } catch (Exception e) {
        e.printStackTrace();
    }

}

}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <fragment
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.MapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</RelativeLayout>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.googlemaps"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission

```

```
android:name="com.example.googlemaps.permission.MAPS_RECEIVE" />

<uses-sdk
    android:minSdkVersion="12"
    android:targetSdkVersion="17" />
<permission
    android:name="com.example.googlemaps.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />

<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.
gsf.permission.
READ_GSERVICES" />
<uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.
ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```


```
<activity
  android:name="com.example.googlemaps.MainActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>

<meta-data
  android:name="com.google.android.maps.v2.API_KEY"
  android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />

</application>

</manifest>
```

Let's try to run your GoogleMaps application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Now what you need to do is to tap on the balloon to see the text.



Now you can customize the google map according to your choice with the functions given in the GoogleMap API.

42. IMAGE EFFECTS

Android allows you to manipulate images by adding different kinds of effects on the images. You can easily apply image processing techniques to add certain kinds of effects on images. The effects could be brightness, darkness, grayscale conversion etc.

Android provides Bitmap class to handle images. This can be found under android.graphics.bitmap. There are many ways through which you can instantiate bitmap. We are creating a bitmap of image from the imageView.

```
private Bitmap bmp;
private ImageView img;
img = (ImageView)findViewById(R.id.imageView1);
BitmapDrawable abmp = (BitmapDrawable)img.getDrawable();
```

Now we will create bitmap by calling getBitmap() function of BitmapDrawable class. Its syntax is given below:

```
bmp = abmp.getBitmap();
```

An image is nothing but a two dimensional matrix. Same way you will handle a bitmap. An image consist of pixels. So you will get pixels from this bitmap and apply processing to it. Its syntax is as follows:

```
for(int i=0; i<bmp.getWidth(); i++){
    for(int j=0; j<bmp.getHeight(); j++){
        int p = bmp.getPixel(i, j);
    }
}
```

The getWidth() and getHeight() functions returns the height and width of the matrix. The getPixel() method returns the pixel at the specified index. Once you got the pixel, you can easily manipulate it according to your needs.

Apart from these methods, there are other methods that help us manipulate images more better.

Sr.No	Method & description
1	copy(Bitmap.Config config, boolean isMutable)

	This method copy this bitmap's pixels into the new bitmap.
2	createBitmap(DisplayMetrics display, int width, int height, Bitmap.Config config) Returns a mutable bitmap with the specified width and height.
3	createBitmap(int width, int height, Bitmap.Config config) Returns a mutable bitmap with the specified width and height.
4	createBitmap(Bitmap src) Returns an immutable bitmap from the source bitmap.
5	extractAlpha() Returns a new bitmap that captures the alpha values of the original.
6	getConfig() This method returns config, otherwise returns null.
7	getDensity() Returns the density for this bitmap
8	getRowBytes() Return the number of bytes between rows in the bitmap's pixels.
9	setPixel(int x, int y, int color) Writes the specified Color into the bitmap (assuming it is mutable) at the x,y coordinate.
10	setDensity(int density) This method specifies the density for this bitmap

Example:

The below example demonstrates some of the image effects on the bitmap. It creates a basic application that allows you to convert the picture into grayscale and much more.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as ImageEffects under a package com.example.imageeffects. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.imageeffects/MainActivity.java**.

```
package com.example.imageeffects;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView img;
    private Bitmap bmp;
```

```
private Bitmap operation;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    img = (ImageView)findViewById(R.id.imageView1);
    BitmapDrawable abmp = (BitmapDrawable)img.getDrawable();
    bmp = abmp.getBitmap();

}

public void gray(View view){
    operation= Bitmap.createBitmap(bmp.getWidth(),
    bmp.getHeight(),bmp.getConfig());

    double red = 0.33;
    double green = 0.59;
    double blue = 0.11;

    for(int i=0; i<bmp.getWidth(); i++){
        for(int j=0; j<bmp.getHeight(); j++){
            int p = bmp.getPixel(i, j);
            int r = Color.red(p);
            int g = Color.green(p);
            int b = Color.blue(p);

            r = (int) red * r;
            g = (int) green * g;
            b = (int) blue * b;

            operation.setPixel(i, j, Color.argb(Color.alpha(p), r, g, b));
        }
    }
}
```

```
    }
    img.setImageBitmap(operation);
}

public void bright(View view){
    operation= Bitmap.createBitmap bmp.getWidth(),
    bmp.getHeight(),bmp.getConfig());

    for(int i=0; i<bmp.getWidth(); i++){
        for(int j=0; j<bmp.getHeight(); j++){
            int p = bmp.getPixel(i, j);
            int r = Color.red(p);
            int g = Color.green(p);
            int b = Color.blue(p);
            int alpha = Color.alpha(p);

            r = 100 + r;
            g = 100 + g;
            b = 100 + b;
            alpha = 100 + alpha;

            operation.setPixel(i, j, Color.argb(alpha, r, g, b));
        }
    }
    img.setImageBitmap(operation);
}

public void dark(View view){
    operation= Bitmap.createBitmap bmp.getWidth(),
    bmp.getHeight(),bmp.getConfig());

    for(int i=0; i<bmp.getWidth(); i++){
```

```

        for(int j=0; j<bmp.getHeight(); j++){
            int p = bmp.getPixel(i, j);
            int r = Color.red(p);
            int g = Color.green(p);
            int b = Color.blue(p);
            int alpha = Color.alpha(p);

            r = r - 50;
            g = g - 50;
            b = b - 50;
            alpha = alpha -50;
            operation.setPixel(i, j, Color.argb(Color.alpha(p), r, g,
            b));

        }
    }

    img.setImageBitmap(operation);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity" >
```

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button1"
    android:layout_alignBottom="@+id/button1"
    android:layout_alignParentRight="true"
    android:layout_marginRight="19dp"
    android:onClick="dark"
    android:text="@string/dark" />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="87dp"
    android:layout_marginRight="17dp"
    android:layout_toLeftOf="@+id/button3"
    android:onClick="gray"
    android:text="@string/gray" />
```

```
<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/button2"
        android:layout_alignBottom="@+id/button2"
        android:layout_centerHorizontal="true"
        android:onClick="bright"
        android:text="@string/bright" />

<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="114dp"
    android:src="@drawable/ic_launcher" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">ImageEffects</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="gray">Gray</string>
    <string name="bright">bright</string>
    <string name="dark">dark</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

```


```
package="com.example.imageeffects"
android:versionCode="1"
android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.imageeffects.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```


Let's try to run our Image Effects 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 following Emulator window:



Now if you will look at your device screen, you will see an image of android along with three buttons.

Now just select the gray button that will convert your image into grayscale and will update the UI. It is shown below:



Now tap on the bright button, that will add some value to each pixel of the image and thus makes an illusion of brightness. It is shown below:



Now tap on the dark button, that will subtract some value to each pixel of the image and thus makes an illusion of dark. It is shown below:



43. IMAGE SWITCHER

Sometimes you don't want an image to appear abruptly on the screen, rather you want to apply some kind of animation to the image when it transitions from one image to another. This is supported by android in the form of ImageSwitcher.

An image switcher allows you to add some transitions on the images through the way they appear on screen. In order to use image Switcher, you need to define its XML component first. Its syntax is given below:

```
<ImageSwitcher
    android:id="@+id/imageSwitcher1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" >
</ImageSwitcher>
```

Now we create an instance of ImageSwitcher in java file and get a reference of this XML component. Its syntax is given below:

```
private ImageSwitcher imageSwitcher;
imageSwitcher = (ImageSwitcher)findViewById(R.id.imageSwitcher1);
```

The next thing we need to do is implement the ViewFactory interface and implement unimplemented method that returns an imageView. Its syntax is below:

```
imageSwitcher.setImageResource(R.drawable.ic_launcher);
imageSwitcher.setFactory(new ViewFactory() {
    public View makeView() {
        ImageView myView = new ImageView(getApplicationContext());
        return myView;
    }
})
```

The last thing you need to do is to add Animation to the ImageSwitcher. You need to define an object of Animation class through AnimationUtilities class by calling a static method loadAnimation. Its syntax is given below:

```

Animation in =
AnimationUtils.loadAnimation(this,android.R.anim.slide_in_left);
imageSwitcher.setInAnimation(in);
imageSwitcher.setOutAnimation(out);
    
```

The method setInAnimaton sets the animation of the appearance of the object on the screen whereas setOutAnimation does the opposite. The method loadAnimation() creates an animation object.

Apart from these methods, there are other methods defined in the ImageSwitcher class. They are defined below:

Sr.No	Method & description
1	<p>setImageDrawable(Drawable drawable)</p> <p>Sets an image with image switcher. The image is passed in the form of bitmap.</p>
2	<p>setImageResource(int resid)</p> <p>Sets an image with image switcher. The image is passed in the form of integer id.</p>
3	<p>setImageURI(Uri uri)</p> <p>Sets an image with image switcher. The image is passed in the form of URI.</p>
4	<p>ImageSwitcher(Context context, AttributeSet attrs)</p> <p>Returns an image switcher object with already setting some attributes passed in the method.</p>
5	<p>onInitializeAccessibilityEvent (AccessibilityEvent event)</p> <p>Initializes an AccessibilityEvent with information about this View which is the event source.</p>
6	<p>onInitializeAccessibilityNodeInfo (AccessibilityNodeInfo info)</p> <p>Initializes an AccessibilityNodeInfo with information about this view</p>

Example:

The below example demonstrates some of the image switcher effects on the bitmap. It creates a basic application that allows you to view the animation effects on the images.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as ImageSwitcher under a package com.example.imageswitcher. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.imageswithcer/MainActivity.java**.

```
package com.example.imageswitcher;

import android.app.ActionBar.LayoutParams;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageButton;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.Toast;
```

```
import android.widget.ViewSwitcher.ViewFactory;

public class MainActivity extends Activity {

    private ImageButton img;
    private ImageSwitcher imageSwitcher;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        img = (ImageButton)findViewById(R.id.imageButton1);
        imageSwitcher = (ImageSwitcher)findViewById(R.id.imageSwitcher1);

        imageSwitcher.setFactory(new ViewFactory() {

            @Override
            public View makeView() {
                ImageView myView = new ImageView(getApplicationContext());
                myView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                myView.setLayoutParams(new ImageSwitcher.LayoutParams(LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
                return myView;
            }
        });
    }

    public void next(View view){
        Toast.makeText(getApplicationContext(), "Next Image",
            Toast.LENGTH_LONG).show();
        Animation in = AnimationUtils.loadAnimation(this,
            android.R.anim.slide_in_left);
    }
}
```



```

        Animation out = AnimationUtils.loadAnimation(this,
        android.R.anim.slide_out_right);
        imageSwitcher.setInAnimation(in);
        imageSwitcher.setOutAnimation(out);
        imageSwitcher.setImageResource(R.drawable.ic_launcher);
    }
    public void previous(View view){
        Toast.makeText(getApplicationContext(), "previous Image",
        Toast.LENGTH_LONG).show();
        Animation in = AnimationUtils.loadAnimation(this,
        android.R.anim.slide_out_right);
        Animation out = AnimationUtils.loadAnimation(this,
        android.R.anim.slide_in_left);
        imageSwitcher.setInAnimation(out);
        imageSwitcher.setOutAnimation(in);
        imageSwitcher.setImageResource(R.drawable.ic_launcher);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"

```

```
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"

android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity" >

<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="54dp"
    android:onClick="next"
    android:src="@android:drawable/ic_menu_send" />

<ImageSwitcher
    android:id="@+id/imageSwitcher1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" >
</ImageSwitcher>

<ImageButton
    android:id="@+id/imageButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="85dp"
    android:onClick="previous"
    android:src="@android:drawable/ic_menu_revert" />
```

```
</RelativeLayout>
```

Following is the content of **AndroidManifest.xml** file.


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.imageswitcher"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.imageswitcher.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Let's try to run our Image Switcher 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 following Emulator window:



Now if you will look at your device screen, you will see the two buttons.

Now just select the upper button with right arrow. An image would appear from right and move towards left. It is shown below:



Now tap on the below button, that will bring back the previous image with some transition. It is shown below:



44. INTERNAL STORAGE

Android provides many kinds of storage for applications to store their data. The storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection.

We are going to look at the internal storage in this chapter. Internal storage is the storage of the private data on the device memory.

By default these files are private and are accessed by only your application and get deleted, when user delete your application.

Writing file

In order to use internal storage to write some data in the file, call the `openFileOutput()` method with the name of the file and the mode. The mode could be private, public etc. Its syntax is given below:

```
FileOutputStream fOut = openFileOutput("file name  
here",MODE_WORLD_READABLE);
```

The method `openFileOutput()` returns an instance of `FileOutputStream`. So you receive it in the object of `FileInputStream`. After that you can call `write` method to write data on the file. Its syntax is given below:

```
String str = "data";  
fOut.write(str.getBytes());  
fOut.close();
```

Reading file

In order to read from the file you just created, call the `openFileInput()` method with the name of the file. It returns an instance of `FileInputStream`. Its syntax is given below:

```
FileInputStream fin = openFileInput(file);
```

After that, you can call `read` method to read one character at a time from the file and then you can print it. Its syntax is given below:

```
int c;  
String temp="";  
while( (c = fin.read()) != -1){
```

```

    temp = temp + Character.toString((char)c);
}
//string temp contains all the data of the file.
fin.close();

```

Apart from the methods of write and close, there are other methods provided by the **FileOutputStream** class for better writing files. These methods are listed below:

Sr.No	Method & description
1	FileOutputStream(File file, boolean append) This method constructs a new FileOutputStream that writes to file.
2	getChannel() This method returns a write-only FileChannel that shares its position with this stream.
3	getFD() This method returns the underlying file descriptor.
4	write(byte[] buffer, int byteOffset, int byteCount) This method Writes count bytes from the byte array buffer starting at position offset to this stream.

Apart from the methods of read and close, there are other methods provided by the **FileInputStream** class for better reading files. These methods are listed below:

Sr.No	Method & description
1	available() This method returns an estimated number of bytes that can be read or skipped without blocking for more input.
2	getChannel() This method returns a read-only FileChannel that shares its position

	with this stream.
3	getFD() This method returns the underlying file descriptor.
4	read(byte[] buffer, int byteOffset, int byteCount) This method reads at most length bytes from this stream and stores them in the byte array by starting at offset.

Example:

Here is an example demonstrating the use of internal storage to store and read files. It creates a basic storage application that allows you to read and write from internal storage.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Storage under a package com.example.storage. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.storage/MainActivity.java**.

```
package com.example.storage;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    private EditText et;
    private String data;
    private String file = "mydata";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et = (EditText)(findViewById(R.id.editText1));
    }

    public void save(View view){
        data = et.getText().toString();
        try {
            FileOutputStream fOut =
                openFileOutput(file,MODE_WORLD_READABLE);
            fOut.write(data.getBytes());
            fOut.close();
            Toast.makeText(getBaseContext(),"file saved",
```

```
        Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void read(View view){
    try{
        FileInputStream fin = openFileInput(file);
        int c;
        String temp="";
        while( (c = fin.read()) != -1){
            temp = temp + Character.toString((char)c);
        }
        et.setText(temp);
        Toast.makeText(getBaseContext(),"file read",
            Toast.LENGTH_SHORT).show();

    }catch(Exception e){

    }

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="182dp"
        android:onClick="save"
        android:text="@string/save" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_alignRight="@+id/button1"
        android:layout_below="@+id/button1"
        android:layout_marginTop="46dp"
        android:onClick="read"
        android:text="@string/read" />

    <EditText
```

```

        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_alignParentTop="true"
        android:layout_marginTop="23dp"
        android:ems="10"
        android:inputType="textMultiLine" >

        <requestFocus />
    </EditText>
</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">Storage</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="save">save to internal storage</string>
    <string name="read">load from internal storage</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.storage"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk


```

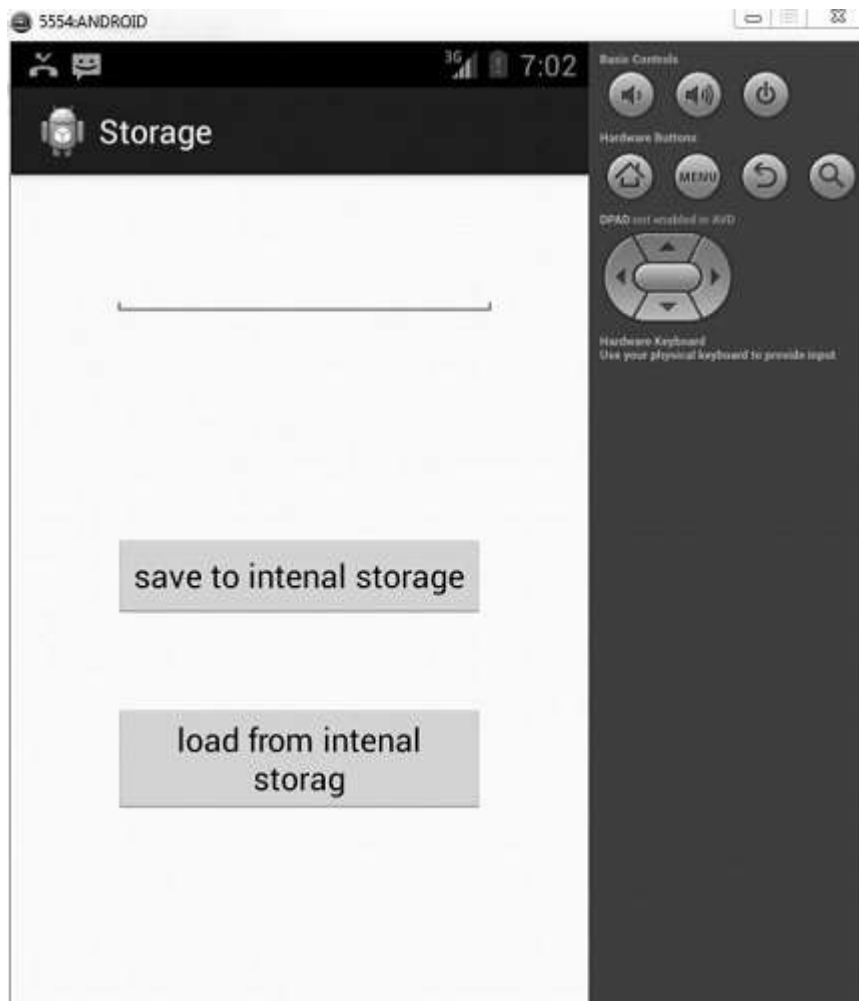
```
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.storage.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

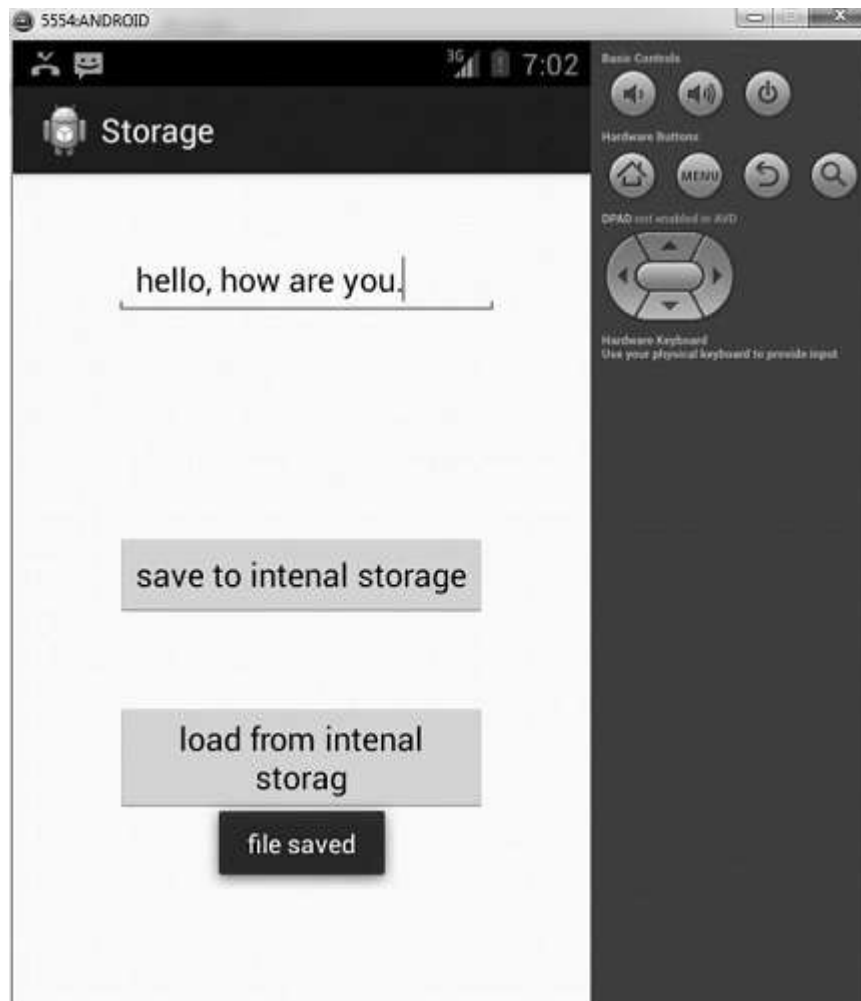
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

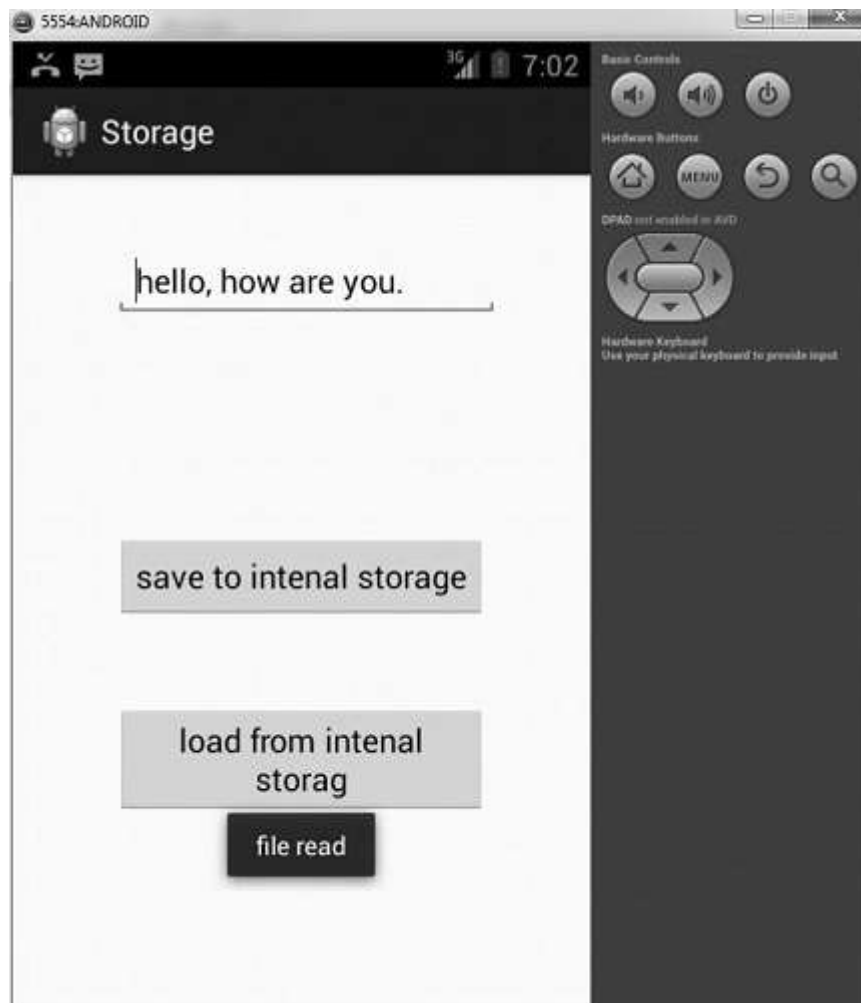
Let's try to run our Storage application we just modified. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything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



Now what you need to do is to enter any text in the field. For example, we have entered some text. Press the save button. The following notification would appear in you AVD:



Now when you press the load button, the application will read the file, and display the data. In our case, following data would be returned:



Note, you can actually view this file by switching to DDMS tab. In DDMS, select file explorer and navigate this path.

```
data>data>com.example.storage>files>mydata
```

This has also been shown in the image below.

com.example.storage	2013-10-29	07:02	drwxr-xr-x	
cache	2013-10-29	07:01	drwxrwx--x	
files	2013-10-29	07:02	drwxrwx--x	
mydata	19 2013-10-29	07:02	-rw-rw-r--	
lib	2013-10-29	07:01	lrwxrwxrwx	-> /data/a...

45. JETPLAYER

The Android platform includes a JET engine that lets you add interactive playback of JET audio content in your applications. Android provides JetPlayer class to handle this stuff.

In order to Jet Content, you need to use the JetCreator tool that comes with AndroidSDK. The usage of jetCreator has been discussed in the example. In order to play the content created by JetCreator, you need JetPlayer class supported by android.

In order to use JetPlayer, you need to instantiate an object of JetPlayer class. Its syntax is given below:

```
JetPlayer jetPlayer = JetPlayer.getJetPlayer();
```

The next thing you need to do is to call loadJetFile method and pass in the path of your Jet file. After that you have to add this into the Queue of JetPlayer. Its syntax is given below:

```
jetPlayer.loadJetFile("/sdcard/level1.jet");  
byte segmentId = 0;  
// queue segment 5, repeat once, use General MIDI, transpose by -1 octave  
jetPlayer.queueJetSegment(5, -1, 1, -1, 0, segmentId++);
```

The method queueJetSegment Queues the specified segment in the JET Queue. The last thing you need is to call the play method to start playing the music. Its syntax is given below:

```
jetPlayer.play();
```

Apart from these methods, there are other methods defined in the JetPlayer class. They are defined below:

Sr.No	Method & description
1	clearQueue() Empties the segment queue, and clears all clips that are scheduled for playback.
2	closeJetFile()

	Closes the resource containing the JET content.
3	getJetPlayer() Factory method for the JetPlayer class.
4	loadJetFile(String path) Loads a .jet file from a given path.
5	pause() Pauses the playback of the JET segment queue.
6	release() Stops the current JET playback, and releases all associated native resources.

Example:

The following example demonstrates the use of JetCreator tool to create Jet content. Once that content is created, you can play it through JetPlayer.

To experiment with this example, you need to run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as JetPlayer under a package com.example.jetplayer. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Install Python and WxPython on your computer from internet.
3	Run the jet creator from command prompt.
4	Create Jet content and then save it.
5	Run the application and verify the results.

Using JetCreator

Installing python

The first step that you need while using JetCreator is to install the python. The python can be installed from its official website <https://www.python.org/> or from anywhere else on the internet.

Please keep in mind the version number of the python should either be 2.6 or 2.7 because this example follows that.

Once you download python, install it. After installing you have to set path to the python. Open your command prompt and type the following command. It is shown in the image below:

```
set path=C:\Python27
```

Once path is set, you can verify it by typing python and hit enter. It is shown below:

```
Python 2.7 (r27:82525, Jul 4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Installing WxPython

The next thing you need to do is to install the wxPython. It can be downloaded [here](#). Once downloaded, you will install it. It will be automatically installed in the python directory.

Running JetCreator

The next thing you need is to move to the path where JetCreator is present. It is in the tools, SDK folder of the android. It is shown below:

```
Administrator: C:\Windows\system32\cmd.exe
D:\adt-bundle-windows-x86-20130219\adt-bundle-windows-x86-20130219\sdk\tools\Jet
Creator>
```

Once in the folder, type this command and hit enter.

```
python JetCreator.py
```

It is shown in the figure below:



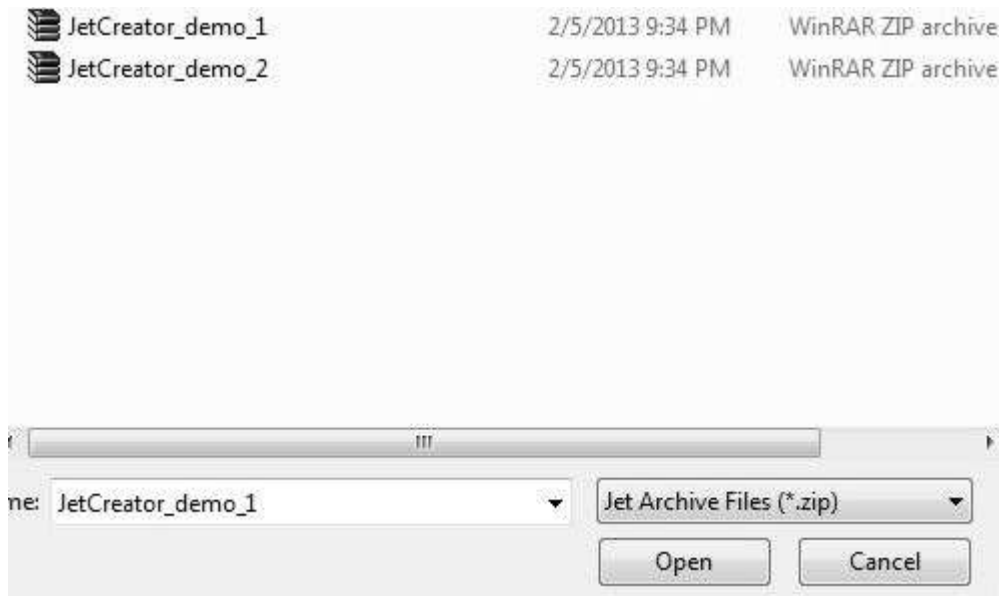
```
D:\adt-bundle-windows-x86-20130219\adt-bundle-windows-x86-20130219\jdk\tools\JetCreator>python JetCreator.py
```

As soon as you hit enter, Jet Creator window will open. It would be something like this.

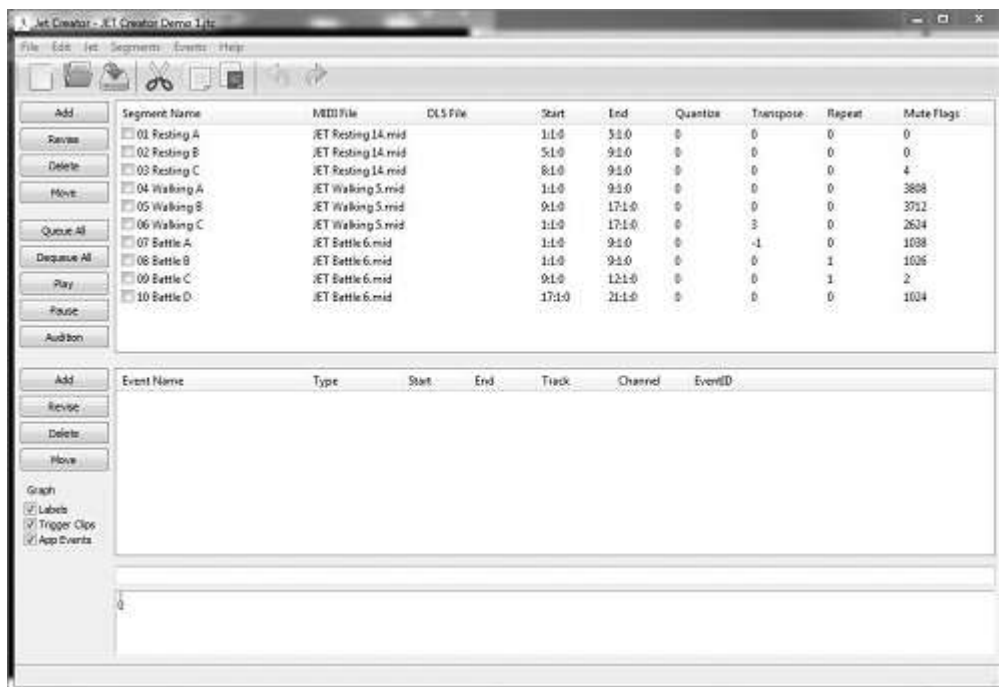


Creating JetContent

In the above Jet Window, click on the import button. And select JetCreator_demo_1 or 2 from the JetFolder from the demo content folder in the Jet folder. It is shown in the image below:



Once you import the content, you will see the content in the JetCreator window. It is shown below:



Now you can explore different options of JetCreator by visiting the JetCreator link http://developer.android.com/guide/topics/media/jet/jetcreator_manual.htm. Finally in order to create .jet file, you need to save the content from the file menu.

Verifying Results

Once you got the jet file, you can play it using jet player. The main code of playing it has been given below:

```
JetPlayer jetPlayer = JetPlayer.getJetPlayer();
jetPlayer.loadJetFile("/sdcard/level1.jet");
byte segmentId = 0;
// queue segment 5, repeat once, use General MIDI, transpose by -1 octave
jetPlayer.queueJetSegment(5, -1, 1, -1, 0, segmentId++);
jetPlayer.play();
```

46. JSON PARSER

JSON stands for JavaScript Object Notation. It is an independent data exchange format and is the best alternative for XML. This chapter explains how to parse the JSON file and extract necessary information from it.

Android provides four different classes to manipulate JSON data. These classes are **JSONArray**, **JSONObject**, **JSONStringer** and **JSONTokenizer**.

The first step is to identify the fields in the JSON data in which you are interested. For example, in the JSON given below we are interested in getting temperature only.

```
{
"sys":
  {
    "country":"GB",
    "sunrise":1381107633,
    "sunset":1381149604
  },
"weather":[
  {
    "id":711,
    "main":"Smoke",
    "description":"smoke",
    "icon":"50n"
  }
],
"main":
  {
    "temp":304.15,
    "pressure":1009,
  }
}
```


JSON - Elements

A JSON file consist of many components. Here is the table defining the compoents of a JSON file and their description:

Sr.No	Component & description
1	<p>Array([])</p> <p>In a JSON file, square bracket ([]) represents a JSON array.</p>
2	<p>Objects({})</p> <p>In a JSON file, curly bracket ({}) represents a JSON object.</p>
3	<p>Key</p> <p>A JSON object contains a key that is just a string. Pairs of key/value make up a JSON object.</p>
4	<p>Value</p> <p>Each key has a value that could be string, integer or double etc.</p>

JSON - Parsing

For parsing a JSON object, we will create an object of class JSONObject and specify a string containing JSON data to it. Its syntax is:

```
String in;
JSONObject reader = new JSONObject(in);
```

The last step is to parse the JSON. A JSON file consist of different object with different key/value pair etc. So JSONObject has a separate function for parsing each of the component of JSON file. Its syntax is given below:

```
JSONObject sys = reader.getJSONObject("sys");
country = sys.getString("country");

JSONObject main = reader.getJSONObject("main");
temperature = main.getString("temp");
```

The method **getJSONObject** returns the JSON object. The method **getString** returns the string value of the specified key.

Apart from these methods, there are other methods provided by this class for better parsing JSON files. These methods are listed below:

Sr.No	Method & description
1	<p>get(String name) This method just Returns the value but in the form of Object type.</p>
2	<p>getBoolean(String name) This method returns the boolean value specified by the key.</p>
3	<p>getDouble(String name) This method returns the double value specified by the key.</p>
4	<p>getInt(String name) This method returns the integer value specified by the key.</p>
5	<p>getLong(String name) This method returns the long value specified by the key.</p>
6	<p>length() This method returns the number of name/value mappings in this object.</p>
7	<p>names() This method returns an array containing the string names in this object.</p>

Example:

Here is an example demonstrating the use of JSONObject class. It creates a basic Weather application that allows you to parse JSON from google weather api and shows the result.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as JSONParser under a package com.example.jsonparser. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Create a new java file under src/HandleJSON.java to fetch and parse XML data.
6	Modify AndroidManifest.xml to add necessary internet permission.
7	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.jsonparser/MainActivity.java**.

```
package com.example.jsonparser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private String url1 =
        "http://api.openweathermap.org/data/2.5/weather?q=";
```

```
private EditText location, country, temperature, humidity, pressure;
private HandleJSON obj;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    location = (EditText)findViewById(R.id.editText1);
    country = (EditText)findViewById(R.id.editText2);
    temperature = (EditText)findViewById(R.id.editText3);
    humidity = (EditText)findViewById(R.id.editText4);
    pressure = (EditText)findViewById(R.id.editText5);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items
    //to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

public void open(View view){
    String url = location.getText().toString();
    String finalUrl = url1 + url;
    country.setText(finalUrl);
    obj = new HandleJSON(finalUrl);
    obj.fetchJSON();

    while(obj.parsingComplete);
    country.setText(obj.getCountry());
    temperature.setText(obj.getTemperature());
    humidity.setText(obj.getHumidity());
    pressure.setText(obj.getPressure());
}
```

```
}  
}
```

Following is the content of **src/com.example.jsonparser/HandleXML.java**.

```
package com.example.jsonparser;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.StringWriter;  
import java.io.UnsupportedEncodingException;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Map;  
  
import org.json.JSONObject;  
import org.xmlpull.v1.XmlPullParser;  
import org.xmlpull.v1.XmlPullParserFactory;  
  
import android.annotation.SuppressLint;  
  
public class HandleJSON {  
    private String country = "county";  
    private String temperature = "temperature";  
    private String humidity = "humidity";  
    private String pressure = "pressure";  
    private String urlString = null;  
  
    public volatile boolean parsingComplete = true;  
    public HandleJSON(String url){  
        this.urlString = url;  
    }  
}
```

```
public String getCountry(){
    return country;
}
public String getTemperature(){
    return temperature;
}
public String getHumidity(){
    return humidity;
}
public String getPressure(){
    return pressure;
}

@SuppressLint("NewApi")
public void readAndParseJSON(String in) {
    try {
        JSONObject reader = new JSONObject(in);

        JSONObject sys = reader.getJSONObject("sys");
        country = sys.getString("country");

        JSONObject main = reader.getJSONObject("main");
        temperature = main.getString("temp");

        pressure = main.getString("pressure");
        humidity = main.getString("humidity");

        parsingComplete = false;

    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

```

    }

}

public void fetchJSON(){
    Thread thread = new Thread(new Runnable(){
        @Override
        public void run() {
            try {
                URL url = new URL(urlString);
                HttpURLConnection conn = (HttpURLConnection)
                    url.openConnection();
                conn.setReadTimeout(10000 /* milliseconds */);
                conn.setConnectTimeout(15000 /* milliseconds */);
                conn.setRequestMethod("GET");
                conn.setDoInput(true);
                // Starts the query
                conn.connect();

                InputStream stream = conn.getInputStream();

                String data = convertStreamToString(stream);

                readAndParseJSON(data);
                stream.close();

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });

        thread.start();
    }

    static String convertStreamToString(java.io.InputStream is) {
        java.util.Scanner s = new

```

```

        java.util.Scanner(is).useDelimiter("\\A");
        return s.hasNext() ? s.next() : "";
    }
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="15dp"
        android:text="@string/location"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/textView1"
        android:layout_alignParentRight="true"
        android:ems="10" />

```



```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="68dp"
    android:text="@string/country"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="19dp"
    android:text="@string/temperature"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView3"
    android:layout_below="@+id/textView3"
    android:layout_marginTop="32dp"
    android:text="@string/humidity"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:layout_alignLeft="@+id/textView4"
    android:layout_below="@+id/textView4"
    android:layout_marginTop="21dp"
    android:text="@string/pressure"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView3"
    android:layout_toRightOf="@+id/textView3"
    android:ems="10" >

    <requestFocus />
</EditText>

<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView3"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignLeft="@+id/editText2"
    android:ems="10" />

<EditText
    android:id="@+id/editText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView5"
    android:layout_alignLeft="@+id/editText1"
    android:ems="10" />
```

```

<EditText
    android:id="@+id/editText5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView5"
    android:layout_alignBottom="@+id/textView5"
    android:layout_alignRight="@+id/editText4"
    android:ems="10" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_below="@+id/editText1"
    android:onClick="open"
    android:text="@string/weather" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">JSONParser</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="location">Location</string>
    <string name="country">Country:</string>
    <string name="temperature">Temperature:</string>
    <string name="humidity">Humidity:</string>
    <string name="pressure">Pressure:</string>
    <string name="weather">Weather</string>
</resources>

```


Following is the content of **AndroidManifest.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.jsonparser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.jsonparser.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Let's try to run our JSONParser 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 following Emulator window:



Now what you need to do is to enter any location in the location field. For example, we have entered newyork. Press the weather button, when you enter the location. The following screen would appear in you AVD:



Now when you press the weather button, the application will contact the Google Weather API and will request for your necessary JSON location file and will parse it. In case of newyork following file would be returned:

London Temperature from google weather api

Note that this temperature is in kelvin, so if you want to convert it into more understandable format, you have to convert it into Celsius.

47. LINKEDIN INTEGRATION

Android allows your application to connect to LinkedIn and share data or any kind of updates on LinkedIn. This chapter is about integrating LinkedIn into your application.

There are two ways through which you can integrate LinkedIn and share something from your application. These ways are listed below.

- LinkedIn SDK (Scribe)
- Intent Share

Integrating LinkedIn SDK

This is the first way of connecting with LinkedIn. You have to register your application and then receive some Application Id, and then you have to download the LinkedIn SDK and add it to your project. The steps are listed below.

Registering your application

Create a new LinkedIn application at <https://www.Linkedin.com/secure/developer>. Click on add new application. It is shown below:

List of Applications

You have not added any applications yet.

 Add New Application

Now fill in your application name, description and your website url. It is shown below:

Application Info

* Application Name:	<input type="text" value="sampletutorialspoint"/>
* Description:	<input type="text" value="login authentication application"/>
* Website URL:	<input type="text" value="tutorialspoint.com"/>

Where your people should go to learn about your application.

If everything works fine, you will receive an API key with the secret. Just copy the API key and save it somewhere. It is shown in the image below:

API Key:

025u0r3zz3uj

Secret Key:

ISGAOTNBscmTcFS4

Downloading SDK and integrating it

Download LinkedIn sdk here. Copy the scribe-1.3.0.jar jar into your project libs folder.

Posting updates on LinkedIn application

Once everything is complete, you can run the LinkedIn samples which can be found here.

Intent share

Intent share is used to share data between applications. In this strategy, we will not handle the SDK stuff, but let the LinkedIn application handle it. We will simply call the LinkedIn application and pass the data to share. This way, we can share something on LinkedIn.

Android provides intent library to share data between activities and applications. In order to use it as share intent, we have to specify the type of the share intent to **ACTION_SEND**. Its syntax is given below:

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
```

Next thing you need is to define the type of data to pass, and then pass the data. Its syntax is given below:

```
shareIntent.setType("text/plain");
shareIntent.putExtra(Intent.EXTRA_TEXT, "Hello, from tutorialspoint");
startActivity(Intent.createChooser(shareIntent, "Share your thoughts"));
```

Apart from these methods, there are other methods available that allows intent handling. They are listed below:

Sr.No	Method & description
1	addCategory(String category) This method adds a new category to the intent.
2	createChooser(Intent target, CharSequence title) Convenience function for creating a ACTION_CHOOSER Intent.
3	getAction() This method retrieve the general action to be performed, such as ACTION_VIEW.
4	getCategories() This method returns the set of all categories in the intent.nt and the current scaling event.
5	putExtra(String name, int value) This method adds extended data to the intent.
6	toString() This method returns a string containing a concise, human-readable description of this object.

Example:

Here is an example demonstrating the use of IntentShare to share data on LinkedIn. It creates a basic application that allows you to share some text on LinkedIn.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as IntentShare under a package com.example.intentshare. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.

2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.intentshare/MainActivity.java**.

```
package com.example.intentshare;

import java.io.File;
import java.io.FileOutputStream;

import com.example.intentshare.R;

import android.app.Activity;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.view.Menu;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private ImageView img;
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    img = (ImageView) findViewById(R.id.imageView1);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

public void open(View view){
    Intent shareIntent = new Intent();
    shareIntent.setAction(Intent.ACTION_SEND);
    shareIntent.setType("text/plain");
    shareIntent.putExtra(Intent.EXTRA_TEXT, "Hello, from
    tutorialspoint");
    startActivity(Intent.createChooser(shareIntent, "Share your
    thoughts"));
}
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"

```

```

        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".MainActivity" >

        <ImageView
            android:id="@+id/imageView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_marginLeft="98dp"
            android:layout_marginTop="139dp"
            android:onClick="open"
            android:src="@drawable/tp" />

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="48dp"
            android:text="@string/tap"
            android:textAppearance="?android:attr/textAppearanceLarge" />

    </RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">IntentShare</string>

```

```

<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="tap">Tap the button to share something</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.intentshare"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

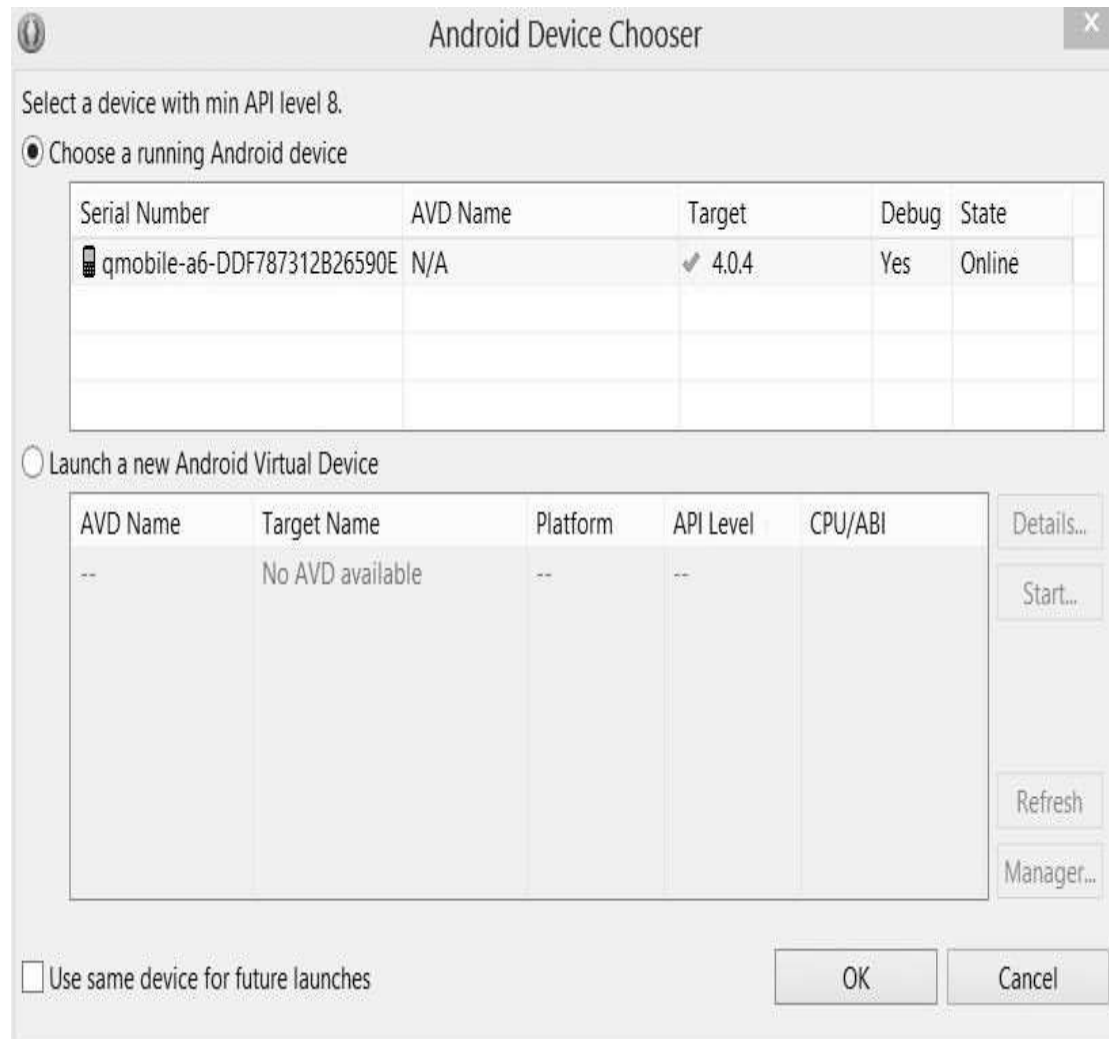
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.intentshare.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

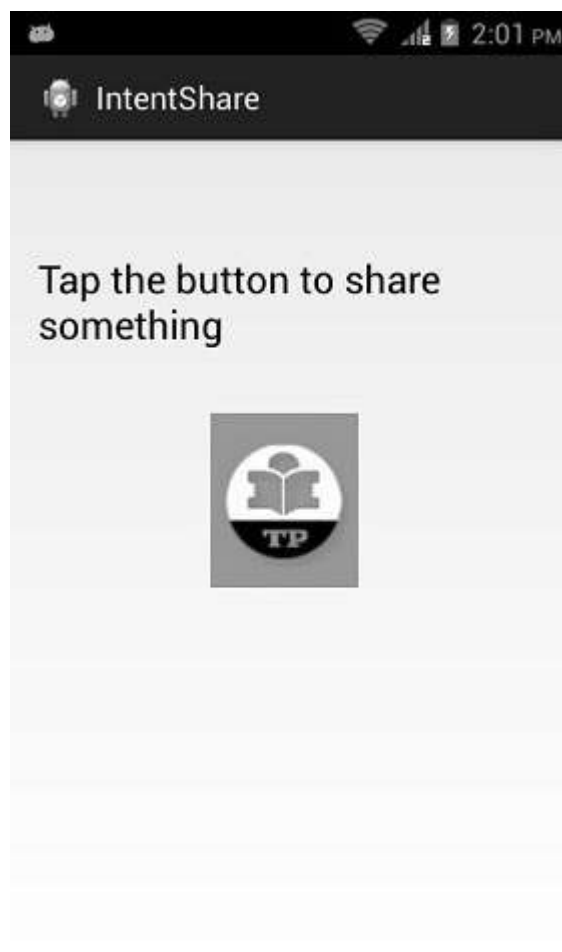
</manifest>

```

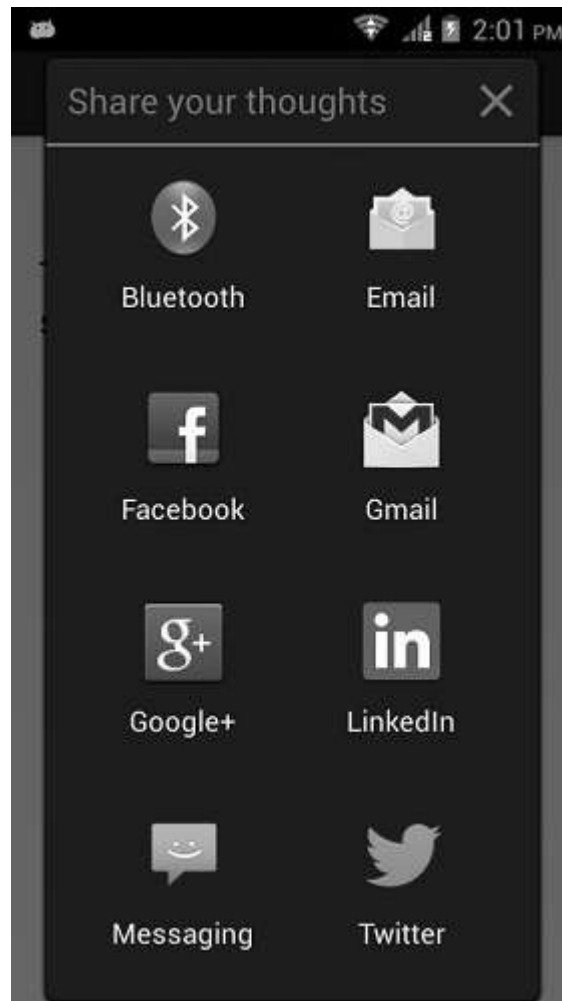
Let's try to run your IntentShare application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



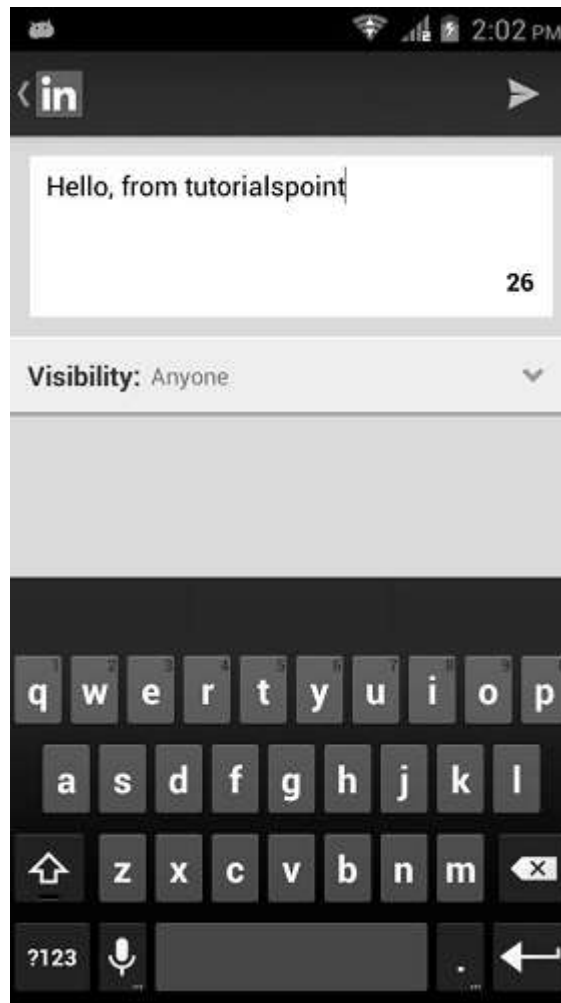
Select your mobile device as an option and then check your mobile device which will display your default screen:



Now just tap on the image logo and you will see a list of share providers.



Now just select LinkedIn from that list and then write any message. It is shown in the image below:



Now, select the arrow button and then it would be posted on your LinkedIn page. It is shown below:

tutorialspoint
Hello, from tutorialspoint
Like • Share • 47 seconds ago

48. LOADING SPINNER

You can show progress of a task in android through loading progress bar. The progress bar comes in two shapes. Loading bar and Loading Spinner. In this chapter we will discuss spinner.

Spinner is used to display progress of those tasks whose total time of completion is unknown. In order to use that, you just need to define it in the xml like this.

```
<ProgressBar
    android:id="@+id/progressBar1"
    style="?android:attr/progressBarStyleLarge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true" />
```

After defining it in xml, you have to get its reference in java file through ProgressBar class. Its syntax is given below:

```
private ProgressBar spinner;
spinner = (ProgressBar)findViewById(R.id.progressBar1);
```

After that you can make it disappear, and bring it back when needed through setVisible Method. Its syntax is given below:

```
spinner.setVisibility(View.GONE);
spinner.setVisibility(View.VISIBLE);
```

Apart from these Methods, there are other methods defined in the ProgressBar class, that you can use to handle spinner more effectively.

Sr.No	Method & description
1	isIndeterminate() Indicate whether this progress bar is in indeterminate mode.
2	postInvalidate() Cause an invalidate to happen on a subsequent cycle through the event loop.

3	setIndeterminate(boolean indeterminate) Change the indeterminate mode for this progress bar.
4	invalidateDrawable(Drawable dr) Invalidates the specified Drawable.
5	incrementSecondaryProgressBy(int diff) Increase the progress bar's secondary progress by the specified amount.
6	getProgressDrawable() Get the drawable used, to draw the progress bar in progress mode.

Example:

Here is an example demonstrating the use of ProgressBar to handle spinner. It creates a basic application that allows you to turn on the spinner on clicking the button.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Spinner under a package com.example.spinner. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.spinner/MainActivity.java**.

```
package com.example.spinner;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.ProgressBar;

public class MainActivity extends Activity {

    private ProgressBar spinner;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        spinner = (ProgressBar)findViewById(R.id.progressBar1);
        spinner.setVisibility(View.GONE);
    }
    public void load(View view){
        spinner.setVisibility(View.VISIBLE);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="103dp"
        android:onClick="load"
        android:text="@string/hello_world" />

    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button1"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

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

    <string name="app_name">Spinner</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">load spinner</string>

</resources>
```

Following is the content of **AndroidManifest.xml** file.


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.spinner"
    android:versionCode="1"
    android:versionName="1.0" >

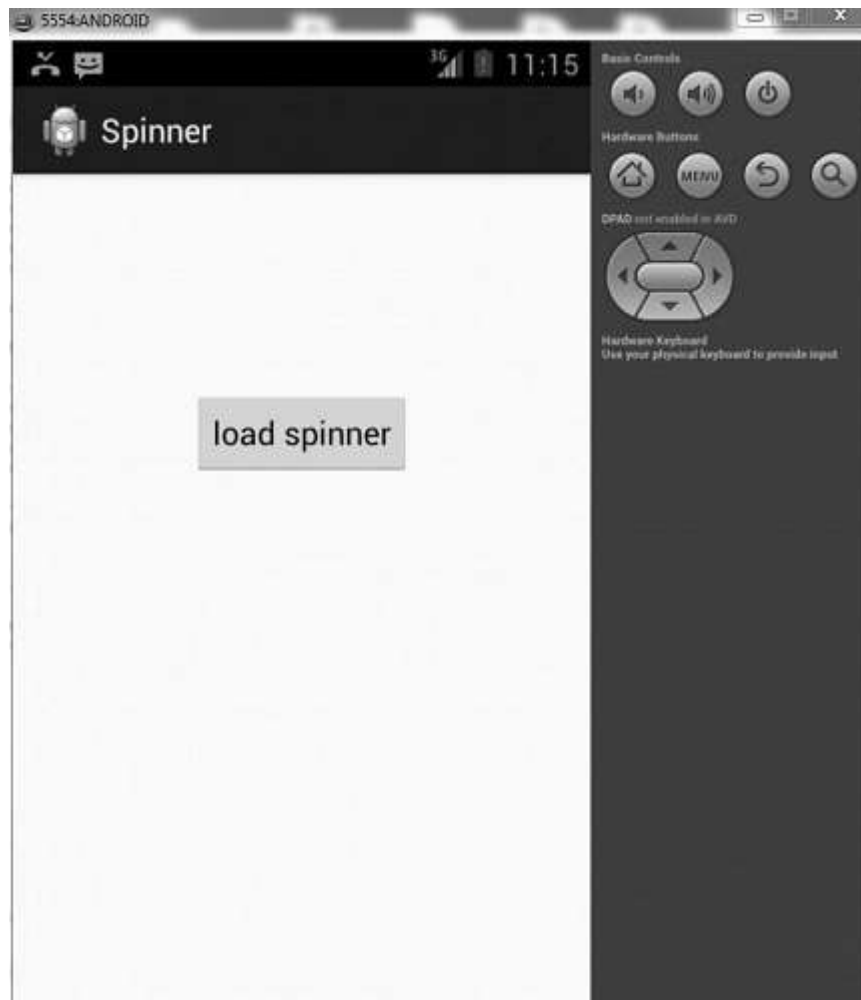
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.spinner.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

```
</activity>
</application>
</manifest>
```

Let's try to run our Loading Spinner 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 following Emulator window:



Now click on the load spinner button to turn on the loading spinner. It is shown in the image below:



49. LOCALIZATION

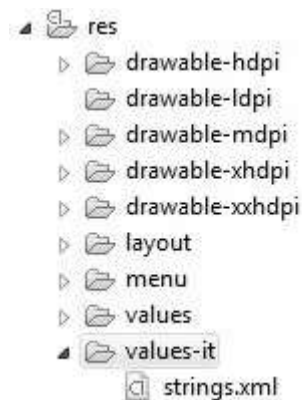
An android application can run on many devices in many different regions. In order to make your application more interactive, your application should handle text, numbers, files etc. in ways appropriate to the locales where your application will be used.

Here we will explain, how you can localize your application according to different regions etc. We will localize the strings used in the application, and in the same way other things can be localized.

Localizing Strings

In order to localize the strings used in your application, make a new folder under **res** with name of **values-local** where local would be the replaced with the region.

For example, in the case of italy, the **values-it** folder would be made under res. It is shown in the image below:



Once that folder is made, copy the **strings.xml** from default folder to the folder you have created and change its contents. For example, we have changed the value of hello_world string.

Italy, res/values-it/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello_world">Ciao mondo!</string>
</resources>
```

Spanish, res/values-it/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello_world">Hola Mundo!</string>
</resources>
```

French, res/values-it/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello_world">Bonjour le monde !</string>
</resources>
```

Apart from these languages, the region code of other languages have been given in the table below:

Sr.No	Language & code
1	Afrikans Code: af. Folder name: values-af
2	Arabic Code: ar. Folder name: values-ar
3	Bengali Code: bn. Folder name: values-bn
4	Czech Code: cs. Folder name: values-cs
5	Chinese Code: zh. Folder name: values-zh
6	German Code: de. Folder name: values-de

7	<p>French</p> <p>Code: fr. Folder name: values-fr</p>
8	<p>Japanese</p> <p>Code: ja. Folder name: values-ja</p>

Example:

Here is an example demonstrating the use of localization of strings. It creates a basic application that allows you to customize your application according to US and Italy region.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Locals under a package com.example.locals. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Create the res/values-it/string.xml to add necessary string components.
6	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.locals/MainActivity.java**.

```
package com.example.locals;
```

```

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

```

```

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="174dp"
    android:text="@string/hello_world"
    android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">Locals</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>

</resources>

```

Following is the content of the **res/values-it/string.xml**.

```

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

    <string name="app_name">Locals</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Ciao mondo!</string>

</resources>

```


Following is the content of **AndroidManifest.xml** file.

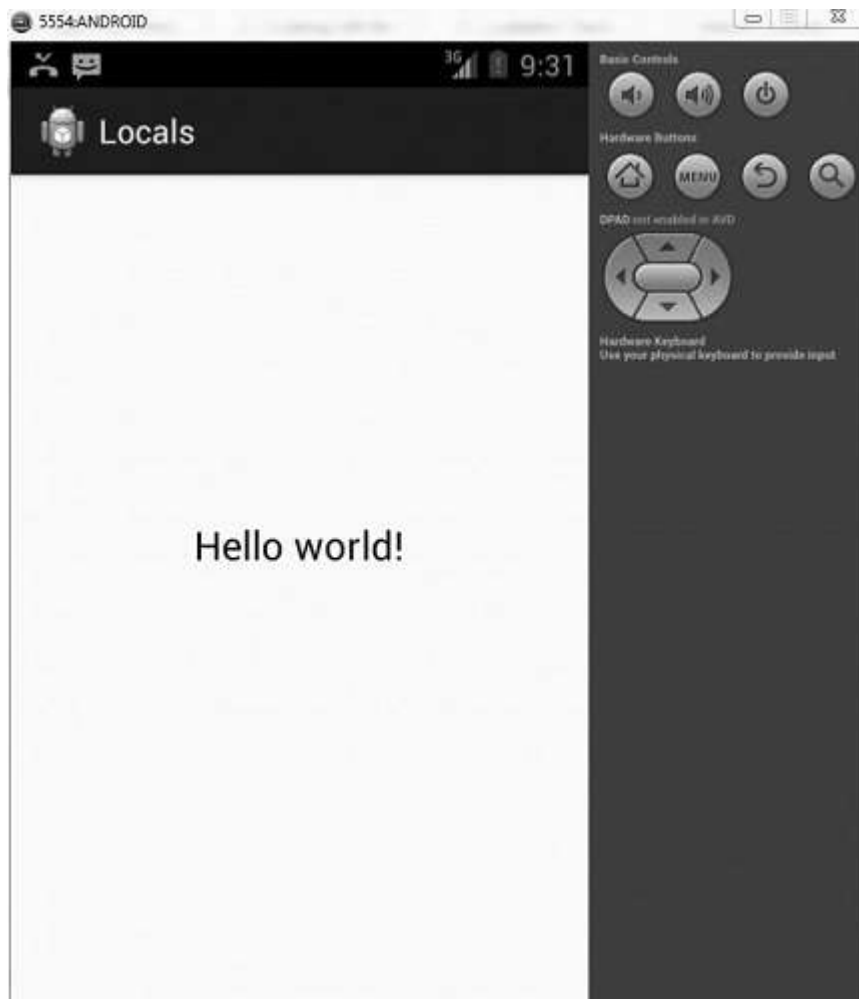
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.locals"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.locals.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Let's try to run our Localization 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 following Emulator window:



Now change your device language setting from menu/system-settings/language to italian.

Now open the application again and this time you will see hello world in italian language. It has been shown below::



50. LOGIN SCREEN

A login application is the screen asking your credentials to login to some particular application. You might have seen it when logging into Facebook, twitter etc.

This chapter explains, how to create a login screen and how to manage security when false attempts are made.

First you have to define two TextView asking username and password of the user. The password TextView must have **inputType** set to password. Its syntax is given below:

```
<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPassword" />
<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

Define a button with login text and set its **onClick** Property. After that define the function mentioned in the onClick property in the java file.

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="login"
    android:text="@string/Login"
/>
```

In the java file, inside the method of onClick get the username and passwords text using **getText()** and **toString()** method and match it with the text using **equals()** function.

```
EditText username = (EditText)findViewById(R.id.editText1);
```

```

EditText password = (EditText)findViewById(R.id.editText2);
public void login(View view){
if(username.getText().toString().equals("admin") &&
password.getText().toString().equals("admin")){
//correcct password
}else{
//wrong password
}
}

```

The last thing you need to do is to provide a security mechanism, so that unwanted attempts should be avoided. For this initialize a variable and on each false attempt, decrement it. And when it reaches to 0, disable the login button.

```

int counter = 3;
counter--;
if(counter==0){
//disble the button, close the application etc.
}

```

Example:

Here is an example demonstrating a login application. It creates a basic application that gives you only three attempts to login to an application.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as LoginScreen under a package com.example.loginscreen. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
3	Modify src/MainActivity.java file to add necessary code.
4	Modify the res/layout/activity_main to add respective XML components.

5	Modify the res/values/string.xml to add necessary string components.
6	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.loginscreen/MainActivity.java**.

```
package com.example.loginscreen;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private EditText username=null;
    private EditText password=null;
    private TextView attempts;
    private Button login;
    int counter = 3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        username = (EditText)findViewById(R.id.editText1);
        password = (EditText)findViewById(R.id.editText2);
        attempts = (TextView)findViewById(R.id.textView5);
        attempts.setText(Integer.toString(counter));
        login = (Button)findViewById(R.id.button1);
```

```
}

public void login(View view){
    if(username.getText().toString().equals("admin") &&
        password.getText().toString().equals("admin")){
        Toast.makeText(getApplicationContext(), "Redirecting...",
            Toast.LENGTH_SHORT).show();
    }
    else{
        Toast.makeText(getApplicationContext(), "Wrong Credentials",
            Toast.LENGTH_SHORT).show();
        attempts.setBackgroundColor(Color.RED);
        counter--;
        attempts.setText(Integer.toString(counter));
        if(counter==0){
            login.setEnabled(false);
        }
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="18dp"
        android:text="@string/hello_world"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="50dp"
        android:text="@string/username"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editText1"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/textView2"
        android:layout_marginLeft="32dp"
        android:layout_toRightOf="@+id/textView2"
        android:ems="10" >

        <requestFocus />
</EditText>

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="38dp"
    android:text="@string/password"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignLeft="@+id/editText1"
    android:ems="10"
    android:inputType="textPassword" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/editText2"
```

```

        android:layout_centerHorizontal="true"
        android:layout_marginTop="94dp"
        android:onClick="login"
        android:text="@string/Login" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView3"
    android:layout_below="@+id/textView3"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="48dp"
    android:text="@string/attempts"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignTop="@+id/textView4"
    android:text="TextView" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">LoginScreen</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Login Screen</string>
    <string name="username">Username:</string>

```

```

<string name="password">Password:</string>
<string name="Login">Login:</string>
<string name="attempts">Attempts Left:</string>

</resources>
Following is the content of AndroidManifest.xml file.
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.loginscreen"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.loginscreen.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```


Let's try to run our Login 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 following Emulator window:



Type anything in the username and password field except admin, and then press the login button. We put admin in the username field and nimda in the password field. We got failed attempt. This is shown below:



Do this two more time, and you will see that you have 0 login attempts left and your login button is disabled.



Now open the application again, and this time enter correct username as admin and password as admin and click on login. You will be successfully logged in.



51. MEDIA PLAYER

Android provides many ways to control playback of audio/video files and streams. One of this way is through a class called **MediaPlayer**.

Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio, video etc. In order to use MediaPlayer, we have to call a static Method **create()** of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows:

```
MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);
```

The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name **raw** and place the music file into it.

Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below.

```
mediaPlayer.start();  
mediaPlayer.pause();
```

On call to **start()** method, the music will start playing from the beginning. If this method is called again after the **pause()** method, the music would start playing from where it is left and not from the beginning.

In order to start music from the beginning, you have to call **reset()** method. Its syntax is given below.

```
mediaPlayer.reset();
```

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below:

Sr.No	Method & description
1	isPlaying() This method just returns true/false indicating the song is playing or not.
2	seekTo(positon) This method takes an integer, and move song to that particular

	second.
3	getCurrentDuration() This method returns the current position of song in milliseconds.
4	getDuration() This method returns the total time duration of song in milliseconds.
5	reset() This method resets the media player.
6	release() This method releases any resource attached with MediaPlayer object.
7	setVolume(float leftVolume, float rightVolume) This method sets the up down volume for this player.
8	setDataSource(FileDescriptor fd) This method sets the data source of audio/video file.
9	selectTrack(int index) This method takes an integer, and select the track from the list on that particular index.
10	getTrackInfo() This method returns an array of track information.

Example:

Here is an example demonstrating the use of MediaPlayer class. It creates a basic media player that allows you to forward, backward, play and pause a song.

To experiment with this example, you need to run this on an actual device to hear the audio sound.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as MediaPlayer under a package com.example.mediaoplayer. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add MediaPlayer code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Create a new folder under MediaPlayer with name as raw and place an mp3 music file in it with name as song.mp3.
6	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.mediaoplayer/MainActivity.java**.

```
package com.example.mediaoplayer;

import java.util.concurrent.TimeUnit;

import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.SeekBar;
```

```
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

    public TextView songName,startTimeField,endTimeField;
    private MediaPlayer mediaPlayer;
    private double startTime = 0;
    private double finalTime = 0;
    private Handler myHandler = new Handler();;
    private int forwardTime = 5000;
    private int backwardTime = 5000;
    private SeekBar seekbar;
    private ImageButton playButton,pauseButton;
    public static int oneTimeOnly = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        songName = (TextView)findViewById(R.id.textView4);
        startTimeField =(TextView)findViewById(R.id.textView1);
        endTimeField =(TextView)findViewById(R.id.textView2);
        seekbar = (SeekBar)findViewById(R.id.seekBar1);
        playButton = (ImageButton)findViewById(R.id.imageButton1);
        pauseButton = (ImageButton)findViewById(R.id.imageButton2);
        songName.setText("song.mp3");
        mediaPlayer = MediaPlayer.create(this, R.raw.song);
        seekbar.setClickable(false);
        pauseButton.setEnabled(false);

    }

    public void play(View view){
        Toast.makeText(getApplicationContext(), "Playing sound",
```



```

Toast.LENGTH_SHORT).show();
    mediaPlayer.start();
    finalTime = mediaPlayer.getDuration();
    startTime = mediaPlayer.getCurrentPosition();
    if(oneTimeOnly == 0){
        seekbar.setMax((int) finalTime);
        oneTimeOnly = 1;
    }

    endTimeField.setText(String.format("%d min, %d sec",
        TimeUnit.MILLISECONDS.toMinutes((long) finalTime),
        TimeUnit.MILLISECONDS.toSeconds((long) finalTime) -
        TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.
            toMinutes((long) finalTime)))
    );
    startTimeField.setText(String.format("%d min, %d sec",
        TimeUnit.MILLISECONDS.toMinutes((long) startTime),
        TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
        TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.
            toMinutes((long) startTime)))
    );
    seekbar.setProgress((int)startTime);
    myHandler.postDelayed(UpdateSongTime,100);
    pauseButton.setEnabled(true);
    playButton.setEnabled(false);
}

private Runnable UpdateSongTime = new Runnable() {
    public void run() {
        startTime = mediaPlayer.getCurrentPosition();
        startTimeField.setText(String.format("%d min, %d sec",
            TimeUnit.MILLISECONDS.toMinutes((long) startTime),
            TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
            TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.

```

```

        toMinutes((long) startTime))
    );
    seekbar.setProgress((int)startTime);
    myHandler.postDelayed(this, 100);
}
};
public void pause(View view){
    Toast.makeText(getApplicationContext(), "Pausing sound",
        Toast.LENGTH_SHORT).show();

    mediaPlayer.pause();
    pauseButton.setEnabled(false);
    playButton.setEnabled(true);
}
public void forward(View view){
    int temp = (int)startTime;
    if((temp+forwardTime)<=finalTime){
        startTime = startTime + forwardTime;
        mediaPlayer.seekTo((int) startTime);
    }
    else{
        Toast.makeText(getApplicationContext(),
            "Cannot jump forward 5 seconds",
            Toast.LENGTH_SHORT).show();
    }
}
public void rewind(View view){
    int temp = (int)startTime;
    if((temp-backwardTime)>0){
        startTime = startTime - backwardTime;
        mediaPlayer.seekTo((int) startTime);
    }
    else{

```

```

        Toast.makeText(getApplicationContext(),
            "Cannot jump backward 5 seconds",
            Toast.LENGTH_SHORT).show();
    }

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageButton
        android:id="@+id/imageButton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"

```

```
        android:layout_alignParentLeft="true"
        android:layout_marginBottom="14dp"
        android:onClick="forward"
        android:src="@android:drawable/ic_media_ff" />

<ImageButton
    android:id="@+id/imageButton4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignTop="@+id/imageButton2"
    android:layout_marginLeft="22dp"
    android:layout_toRightOf="@+id/imageButton2"
    android:onClick="rewind"
    android:src="@android:drawable/ic_media_rew" />

<ImageButton
    android:id="@+id/imageButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/imageButton1"
    android:layout_marginLeft="14dp"
    android:layout_toRightOf="@+id/imageButton1"
    android:onClick="pause"
    android:src="@android:drawable/ic_media_pause" />

<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/imageButton3"
    android:layout_marginLeft="24dp"
    android:layout_toRightOf="@+id/imageButton3"
    android:onClick="play"
```

```
android:src="@android:drawable/ic_media_play" />
```

```
<SeekBar
```

```
    android:id="@+id/seekBar1"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/imageButton3"  
    android:layout_toLeftOf="@+id/textView2"  
    android:layout_toRightOf="@+id/textView1" />
```

```
<TextView
```

```
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignRight="@+id/imageButton3"  
    android:layout_alignTop="@+id/seekBar1"  
    android:text="@string/initial_Time"  
    android:textAppearance="?android:attr/textAppearanceSmall" />
```

```
<TextView
```

```
    android:id="@+id/textView2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/imageButton4"  
    android:layout_alignTop="@+id/seekBar1"  
    android:text="@string/initial_Time"  
    android:textAppearance="?android:attr/textAppearanceSmall" />
```

```
<TextView
```

```
    android:id="@+id/textView3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/imageButton3"  
    android:text="@string/hello_world"
```

```

        android:textAppearance="?android:attr/textAppearanceMedium" />

<ImageView
    android:id="@+id/imageView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView3"
    android:src="@drawable/ic_launcher" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView3"
    android:layout_alignBottom="@+id/textView3"
    android:layout_toRightOf="@+id/imageButton1"
    android:text="TextView" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">MediaPlayer</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Now Playing:</string>
    <string name="inital_Time">0 min, 0 sec</string>

</resources>

```


Following is the content of **AndroidManifest.xml** file.

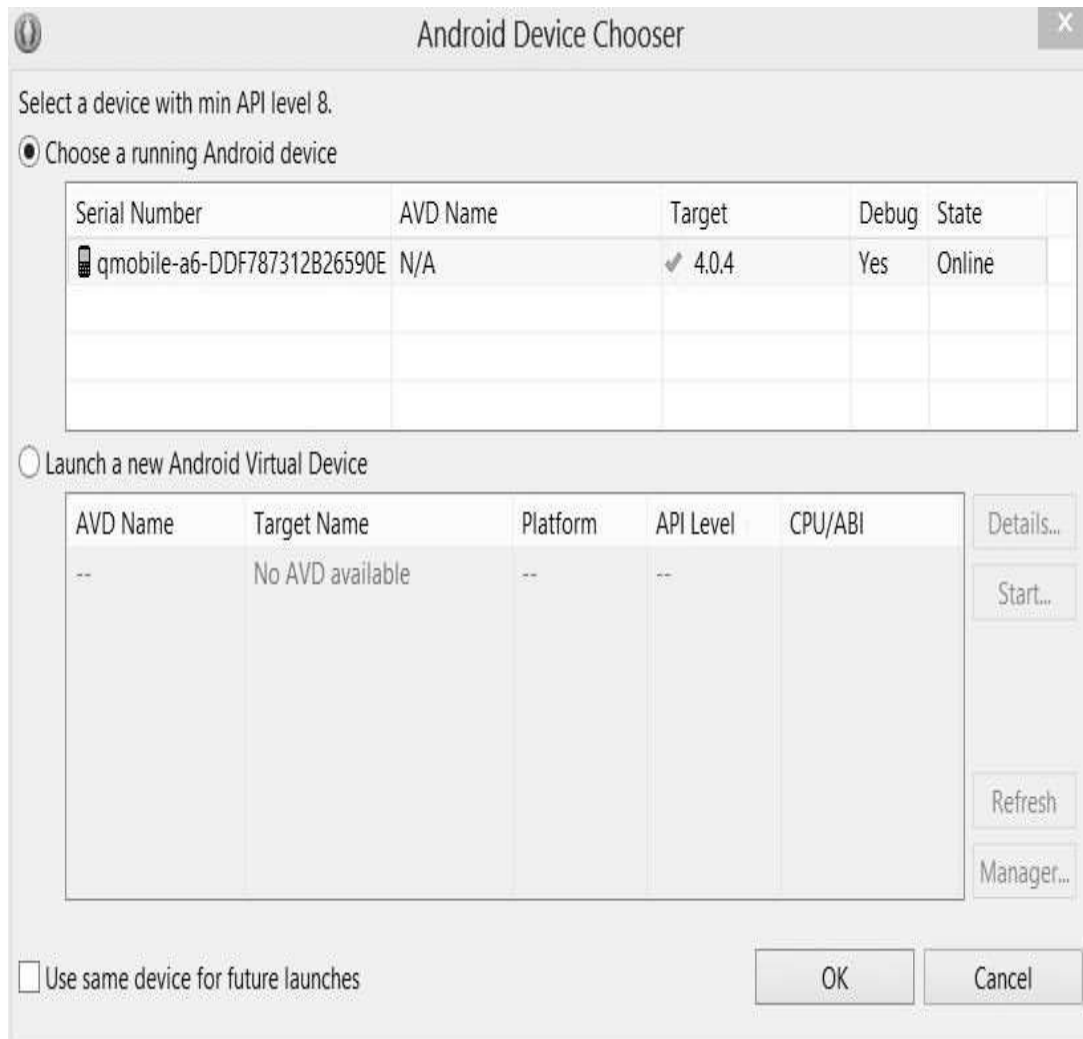
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mediaoplayer"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

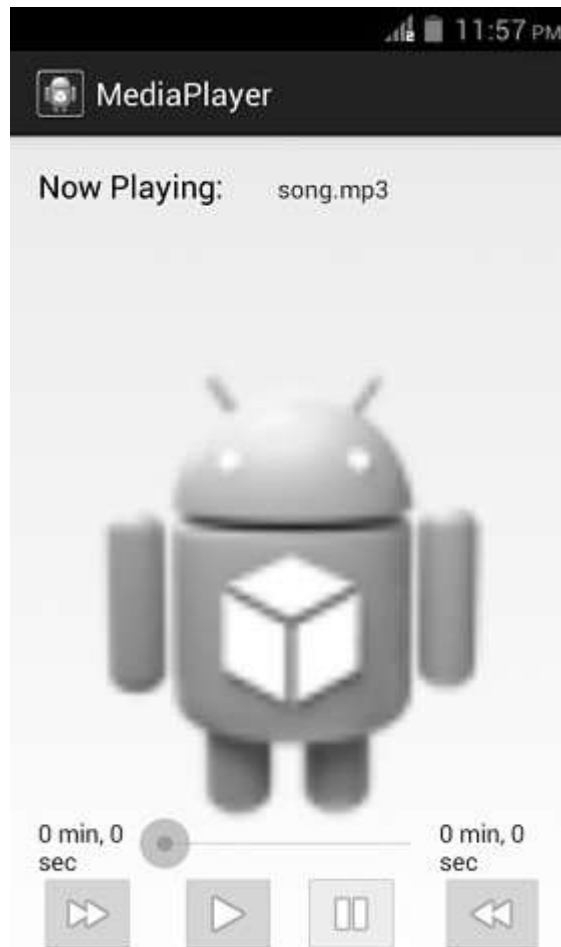
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.mediaoplayer.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

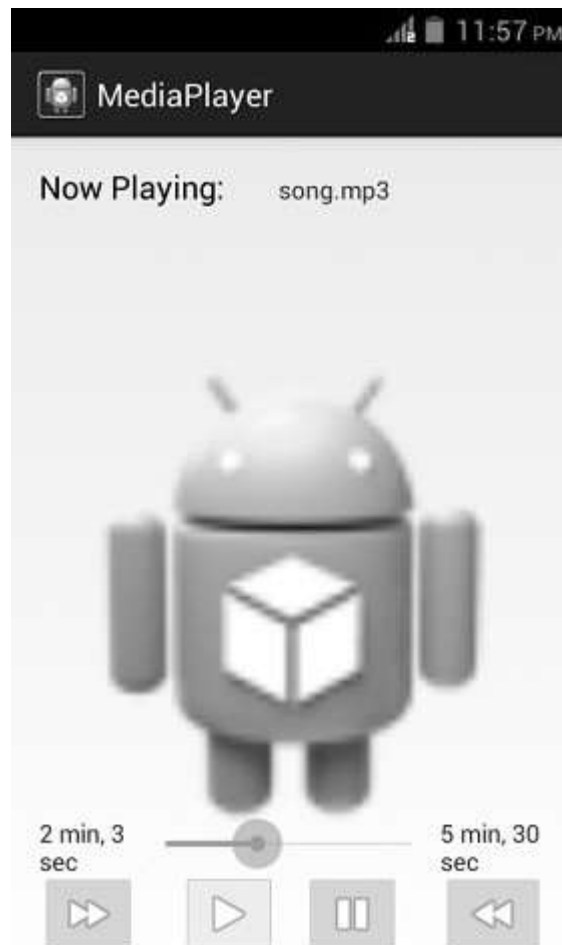
Let's try to run your MediaPlayer application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



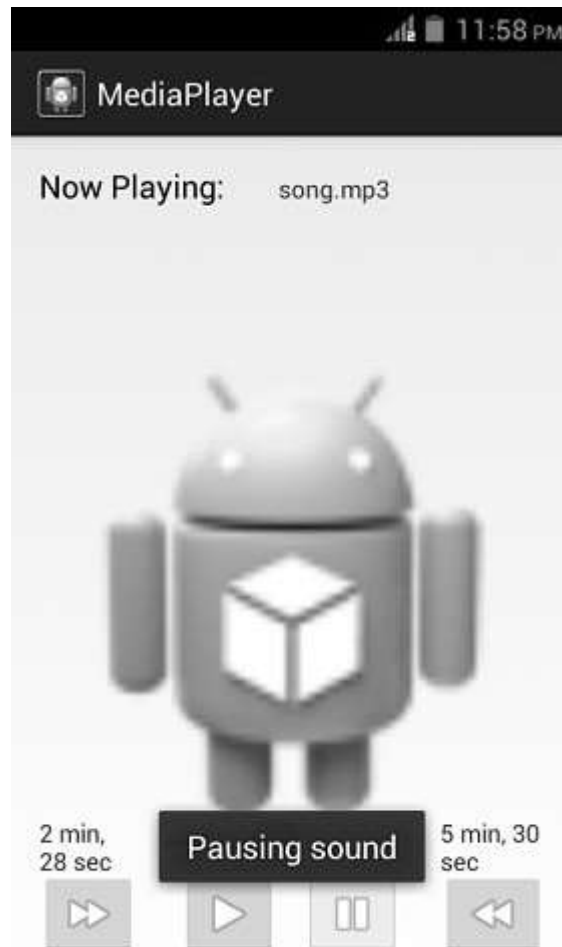
Select your mobile device as an option and then check your mobile device which will display your default screen:



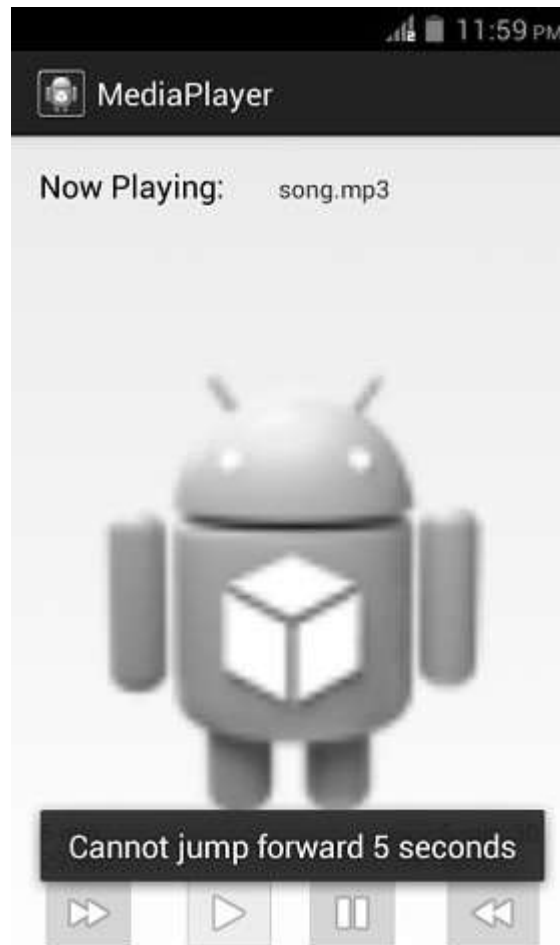
By default you would see the pause button disabled. Now press play button to disable it, and it would enable the pause button. It is shown in the picture below:



Uptill now, the music has been playing. Now press the pause button and see the pause notification. This is shown below:



Now when you press the play button again, the song will not play from the beginning but from where it was paused. Now press the fast forward or backward button to jump the song forward or backward to 5 seconds. A time would come when the song cannot be jumped forward. At this point, the notification would appear which would be something like this:



Your music would remain playing in the background while you are doing other tasks in your mobile. In order to stop it, you have to exit this application from background activities.

52. MULTITOUCH

Multi-touch gesture happens when more than one finger touches the screen at the same time. Android allows us to detect these gestures.

Android system generates the following touch events whenever multiple fingers touches the screen at the same time.

Sr.No	Event & description
1	ACTION_DOWN For the first pointer that touches the screen. This starts the gesture.
2	ACTION_POINTER_DOWN For extra pointers that enter the screen beyond the first.
3	ACTION_MOVE A change has happened during a press gesture.
4	ACTION_POINTER_UP Sent when a non-primary pointer goes up.
5	ACTION_UP Sent when the last pointer leaves the screen.

So in order to detect any of the above mentioned event, you need to override **onTouchEvent()** method and check these events manually. Its syntax is given below:

```
public boolean onTouchEvent(MotionEvent ev){
    final int actionPeformed = ev.getAction();
    switch(actionPeformed){
        case MotionEvent.ACTION_DOWN:{
            break;
        }
    }
}
```

```

    case MotionEvent.ACTION_MOVE:{
        break;
    }
    return true;
}

```

In these cases, you can perform any calculation you like. For example zooming, shrinking etc. In order to get the co-ordinates of the X and Y axis, you can call **getX()** and **getY()** method. Its syntax is given below:

```

final float x = ev.getX();
final float y = ev.getY();

```

Apart from these methods, there are other methods provided by this MotionEvent class for better dealing with multitouch. These methods are listed below:

Sr.No	Method & description
1	getAction() This method returns the kind of action being performed.
2	getPressure() This method returns the current pressure of this event for the first index.
3	getRawX() This method returns the original raw X coordinate of this event.
4	getRawY() This method returns the original raw Y coordinate of this event.
5	getSize() This method returns the size for the first pointer index.
6	getSource() This method gets the source of the event.

7	getXPrecision() This method returns the precision of the X coordinates being reported.
8	getYPrecision() This method returns the precision of the Y coordinates being reported.

Example:

Here is an example demonstrating the use of Multitouch. It creates a basic Multitouch gesture application that allows you to view the co-ordinates when multitouch is performed.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Multitouch under a package com.example.multitouch. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add multitouch code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.multitouch/MainActivity.java**.

```
package com.example.multitouch;

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

```
import android.view.Menu;
import android.view.MotionEvent;

import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    float xAxis = 0f;
    float yAxis = 0f;
    float lastXAxis = 0f;
    float lastYAxis = 0f;
    private EditText xText,yText,moveX,moveY;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        xText = (EditText)findViewById(R.id.editText2);
        yText = (EditText)findViewById(R.id.editText3);
        moveX = (EditText)findViewById(R.id.editText1);
        moveY = (EditText)findViewById(R.id.editText4);
    }

    @Override
    public boolean onTouchEvent(MotionEvent ev){
        final int actionPeformed = ev.getAction();
        switch(actionPeformed){
            case MotionEvent.ACTION_DOWN:{
                final float x = ev.getX();
                final float y = ev.getY();
                lastXAxis = x;
                lastYAxis = y;
                xText.setText(Float.toString(lastXAxis));
                yText.setText(Float.toString(lastYAxis));
            }
            break;
        }
    }
}
```



```

    }
    case MotionEvent.ACTION_MOVE:{
        final float x = ev.getX();
        final float y = ev.getY();
        final float dx = x - lastXAxis;
        final float dy = y - lastYAxis;
        xAxis += dx;
        yAxis += dy;
        moveX.setText(Float.toString(xAxis));
        moveY.setText(Float.toString(yAxis));
        break;
    }
}
return true;

}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"

```

```
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity" >

<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_below="@+id/editText2"
    android:ems="10" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_marginTop="25dp"
    android:ems="10" >
    <requestFocus />
</EditText>

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText3"
    android:layout_below="@+id/editText3"
    android:ems="10" >
</EditText>

<EditText
    android:id="@+id/editText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
android:layout_alignLeft="@+id/editText1"  
android:layout_below="@+id/editText1"  
android:ems="10" />
```

```
<TextView  
    android:id="@+id/textView2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/editText3"  
    android:layout_alignParentLeft="true"  
    android:text="@string/xaxis"  
    android:textAppearance="?android:attr/textAppearanceSmall" />
```

```
<TextView  
    android:id="@+id/textView3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editText3"  
    android:layout_alignRight="@+id/textView2"  
    android:text="@string/yaxis"  
    android:textAppearance="?android:attr/textAppearanceSmall" />
```

```
<TextView  
    android:id="@+id/textView4"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/editText4"  
    android:layout_alignLeft="@+id/textView3"  
    android:text="@string/MoveX"  
    android:textAppearance="?android:attr/textAppearanceSmall" />
```

```
<TextView  
    android:id="@+id/textView5"  
    android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/editText4"
        android:layout_alignBottom="@+id/editText4"
        android:layout_alignRight="@+id/textView4"
        android:text="@string/MoveY"
        android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="109dp"
    android:text="@string/perform"
    android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Gestures</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Pinch to zoom in or out!</string>
    <string name="xaxis">X-Axis</string>
    <string name="yaxis">Y-Axis</string>
    <string name="MoveX">Move X</string>
    <string name="MoveY">Move Y</string>
    <string name="perform">Touch here</string>
</resources>

```

Following is the content of **AndroidManifest.xml** file.

```


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

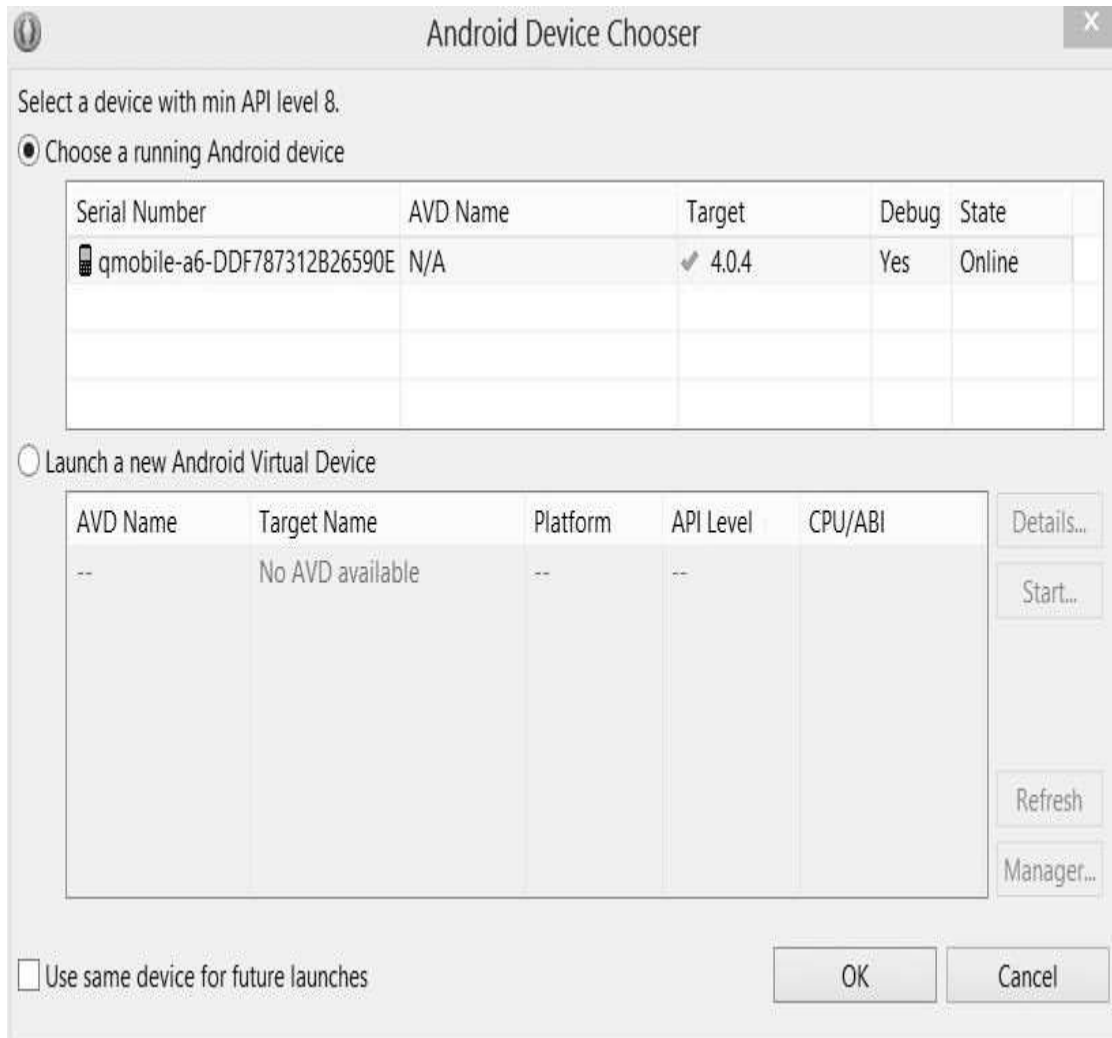
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.multitouch"
    android:versionCode="1"
    android:versionName="1.0" >

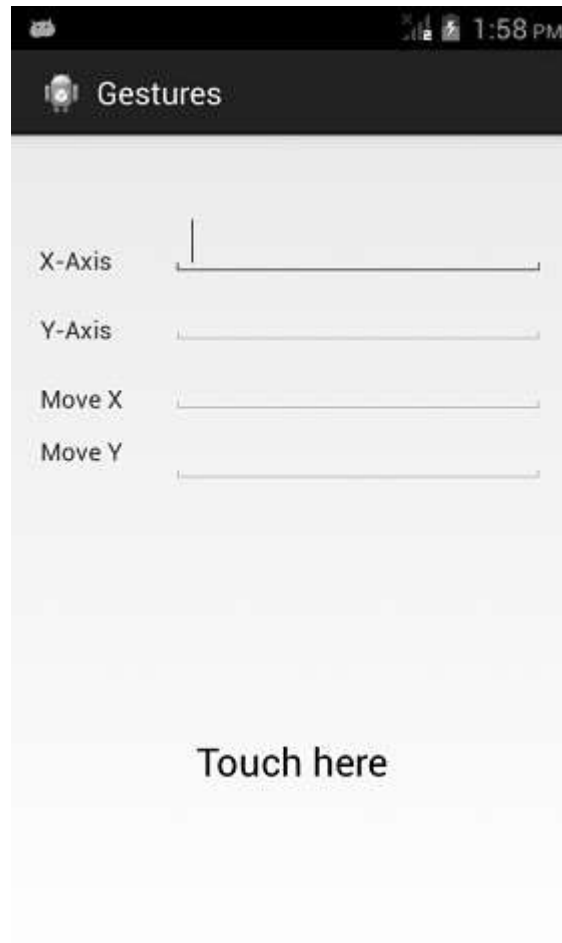
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.multitouch.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

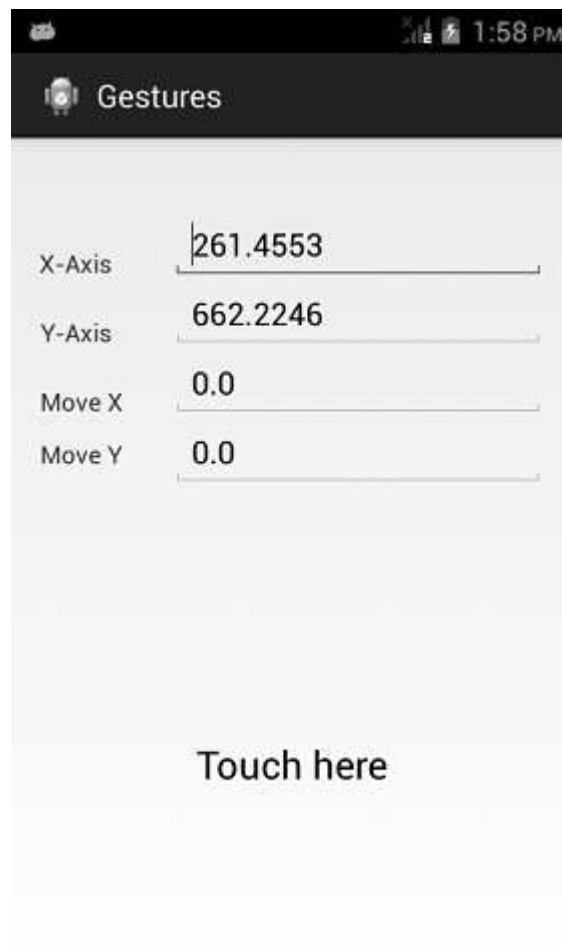
Let's try to run your Multitouch application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



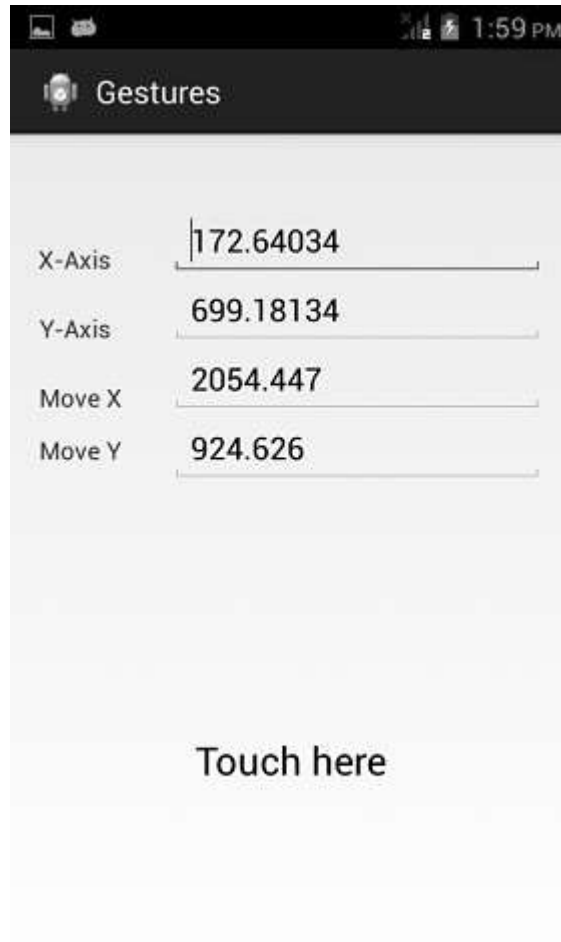
Select your mobile device as an option and then check your mobile device which will display your default screen:



By default you will see nothing in any field. Now just tap on the Touch here area and see some data in the fields. It is shown below:



You will see that the data in the Move field is 0, because only a single touch gesture has been performed. Now tap on the screen and start dragging your finger. You will see the change in the data of the move field. It is shown below:



53. NAVIGATION

We will see how you can provide navigation forward and backward between an application in this chapter. We will first look at how to provide up navigation in an application.

Providing Up Navigation

The up navigation will allow our application to move to previous activity from the next activity. It can be done like this.

To implement Up navigation, the first step is to declare which activity is the appropriate parent for each activity. You can do it by specifying **parentActivityName** attribute in an activity. Its syntax is given below:

```
android:parentActivityName="com.example.test.MainActivity"
```

After that you need to call **setDisplayHomeAsUpEnabled()** method of **getActionBar()** in the onCreate method of the activity. This will enable the back button in the top action bar.

```
getActionBar().setDisplayHomeAsUpEnabled(true);
```

The last thing you need to do is to override **onOptionsItemSelected** method. When the user presses it, your activity receives a call to **onOptionsItemSelected()**. The ID for the action is **android.R.id.home**. Its syntax is given below:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(this);
            return true;
    }
}
```

Handling device back button

Since you have enabled your back button to navigate within your application, you might want to put the application close function in the device back button.

It can be done by overriding **onBackPressed** and then calling **moveTaskToBack** and **finish** method. Its syntax is given below:

```
@Override
public void onBackPressed() {
    moveTaskToBack(true);
    MainActivity2.this.finish();
}
```

Apart from this `setDisplayHomeAsUpEnabled` method, there are other methods available in `ActionBar` API class. They are listed below:

Sr.No	Method & description
1	addTab(ActionBar.Tab tab, boolean setSelected) This method adds a tab for use in tabbed navigation mode.
2	getSelectedTab() This method returns the currently selected tab if in tabbed navigation mode and at least one tab is present there. If there is none than it returns null.
3	hide() This method hides the ActionBar if it is currently showing.
4	removeAllTabs() This method remove all tabs from the action bar and deselects the current tab.
5	selectTab(ActionBar.Tab tab) This method selects the specified tab.

Example:

The below example demonstrates the use of Navigation. It creates a basic application that allows you to navigate within your application.

To experiment with this example, you need to run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as test under a package com.example.test. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Activity code.
3	Create a new activity with the name of MainActivity2 and edit it to add activity code.
4	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
5	Modify layout XML file res/layout/activity_main_activity2.xml add any GUI component if required.
6	Modify res/values/string.xml file and add necessary string components.
7	Modify AndroidManifest.xml to add necessary code.
8	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.test/MainActivity.java**.

```
package com.example.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

public void activity2(View view){
    Intent intent = new
    Intent(this,com.example.test.MainActivity2.class);
    startActivity(intent);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
```

Here is the content of **src/com.example.test/MainActivity2.java**.

```
package com.example.test;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
```

```
public class MainActivity2 extends Activity {

    @SuppressWarnings("NewApi")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_activity2);
        getActionBar().setDisplayHomeAsUpEnabled(true);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main_activity2, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            // Respond to the action bar's Up/Home button
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onBackPressed() {
        moveTaskToBack(true);
        MainActivity2.this.finish();
    }
}
```

Here is the content of **activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="87dp"
        android:text="@string/test1"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:onClick="activity2"
        android:text="@string/go2" />

</RelativeLayout>

```

Here is the content of **activity_main_activity2.xml**.

```

<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"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity2" >

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="125dp"
    android:text="@string/test2"
    android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">test</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="test1">This is activity 1</string>
    <string name="test2">This is activity 2</string>
    <string name="go1">Go to activity 1</string>
    <string name="go2">Go to activity 2</string>
    <string name="title_activity_main_activity2">MainActivity2</string>

```



```
</resources>
```

Here is the content of **AndroidManifest.xml**.


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.test"
    android:versionCode="1"
    android:versionName="1.0" >

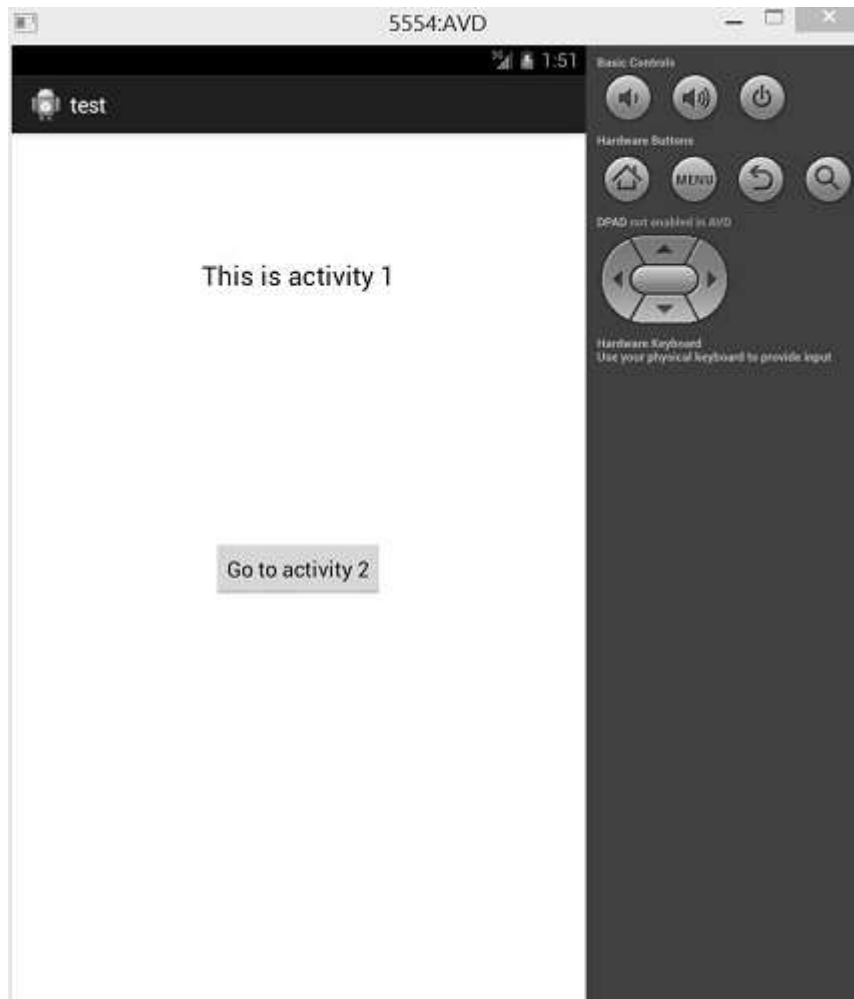
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="14" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.test.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.example.test.MainActivity2"
            android:label="@string/title_activity_main_activity2"
            android:parentActivityName="com.example.test.MainActivity" >
        </activity>
    </application>
```

```
</manifest>
```

Let's try to run your Navigation application. 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:



Now just press the go to activity2 button and the following screen will be shown to you.



Now at the top right corner, you will see the back button. Just press the back button and you will be brought to the first activity.



Now again go to activity2 and this time press the device exit button. You will see your application will be closed. It is shown in the image below:



54. NETWORK CONNECTION

Android lets your application connect to the internet or any other local network and allows you to perform network operations.

A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection.

Checking Network Connection

Before you perform any network operations, you must first check if you are connected to that network or internet etc. For this, android provides **ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method. Its syntax is given below:

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

Once you instantiate the object of **ConnectivityManager** class, you can use **getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

The last thing you need to do is to check **Connected State** of the network. Its syntax is given below:

```
for (int i = 0; i<info.length; i++){
    if (info[i].getState() == NetworkInfo.State.CONNECTED){
        Toast.makeText(context, "Internet is connected
        Toast.LENGTH_SHORT).show();
    }
}
```

Apart from this connected states, there are other states a network can achieve. They are listed below:

Sr.No	State
1	Connecting

2	Disconnected
3	Disconnecting
4	Suspended
5	Unknown

Performing Network Operations

After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.

Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website. Its syntax is as follows:

```
String link = "http://www.google.com";
URL url = new URL(link);
```

After that you need to call **openConnection** method of url class and receive it in an HttpURLConnection object. After that you need to call the **connect** method of HttpURLConnection class.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();
```

And the last thing you need to do is to fetch the HTML from the website. For this you will use **InputStream** and **BufferedReader** class. Its syntax is given below:

```
InputStream is = conn.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
String webPage = "", data="";
while ((data = reader.readLine()) != null){
    webPage += data + "\n";
}
```

Apart from this connect method, there are other methods available in HttpURLConnection class. They are listed below:

Sr.No	Method & description
1	<p>disconnect()</p> <p>This method releases this connection so that its resources may be either reused or closed.</p>
2	<p>getRequestMethod()</p> <p>This method returns the request method which will be used to make the request to the remote HTTP server.</p>
3	<p>getResponseCode()</p> <p>This method returns response code returned by the remote HTTP server.</p>
4	<p>setRequestMethod(String method)</p> <p>This method Sets the request command which will be sent to the remote HTTP server.</p>
5	<p>usingProxy()</p> <p>This method returns whether this connection uses a proxy server or not.</p>

Example:

The below example demonstrates the use of HttpURLConnection class. It creates a basic application that allows you to download HTML from a given webpage.

To experiment with this example, you need to run this on an actual device on which wifi internet is connected.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as NetworkConnection under a package com.example.networkconnection. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Activity code.

3	Create src/DownloadWebPage.java file to add NetworkConnection code.
4	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
5	Modify res/values/string.xml file and add necessary string components.
6	Modify AndroidManifest.xml to add necessary permissions.
7	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.networkconnection/MainActivity.java**.

```
package com.example.networkconnection;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

    private EditText urlField;
    private TextView data;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        urlField = (EditText)findViewById(R.id.editText1);
        data = (TextView)findViewById(R.id.textView2);
    }
}
```

```

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void download(View view){

        String url = urlField.getText().toString();
        new DownloadWebPage(this,data).execute(url);
    }
}

```

Here is the content of **src/com.example.networkconnection/DownloadWebPage.java**.

```

package com.example.networkconnection;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.widget.EditText;
import android.widget.TextView;

```

```
import android.widget.Toast;

public class DownloadWebPage extends AsyncTask{

    private TextView dataField;
    private Context context;
    public DownloadWebPage(Context context,TextView dataField) {
        this.context = context;
        this.dataField = dataField;
    }

    //check Internet connection.
    private void checkInternetConenction(){
        ConnectivityManager check = (ConnectivityManager) this.context.
        getSystemService(Context.CONNECTIVITY_SERVICE);
        if (check != null)
        {
            NetworkInfo[] info = check.getAllNetworkInfo();
            if (info != null)
                for (int i = 0; i <info.length; i++)
                    if (info[i].getState() == NetworkInfo.State.CONNECTED)
                    {
                        Toast.makeText(context, "Internet is connected",
                        Toast.LENGTH_SHORT).show();
                    }
        }
        else{
            Toast.makeText(context, "not conencted to internet",
            Toast.LENGTH_SHORT).show();
        }
    }
    protected void onPreExecute(){
```

```

        checkInternetConenction();
    }
    @Override
    protected String doInBackground(String... arg0) {
        try{
            String link = (String)arg0[0];
            URL url = new URL(link);
            HttpURLConnection conn = (HttpURLConnection)
            url.openConnection();
            conn.setReadTimeout(10000);
            conn.setConnectTimeout(15000);
            conn.setRequestMethod("GET");
            conn.setDoInput(true);
            conn.connect();
            InputStream is = conn.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader
            (is, "UTF-8" ) );
            String data = null;
            String webPage = "";
            while ((data = reader.readLine()) != null){
                webPage += data + "\n";
            }
            return webPage;
        }catch(Exception e){
            return new String("Exception: " + e.getMessage());
        }
    }
    @Override
    protected void onPostExecute(String result){
        this.dataField.setText(result);
    }
}

```

Here is the content of **activity_main.xml**.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="30dp"
        android:text="@string/url"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="18dp"
        android:ems="10" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="69dp"
        android:onClick="download"
        android:text="@string/click" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/button1"
    android:layout_marginTop="56dp"
    android:text="@string/google"
    android:textAppearance="?android:attr/textAppearanceSmall" />

</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">NetworkConnection</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="url">URL here</string>
    <string name="click">Download WebPage</string>
    <string name="google">http://www.tutorialspoint.com</string>

</resources>

```

Here is the content of **AndroidManifest.xml**


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.networkconnection"
    android:versionCode="1"
    android:versionName="1.0" >

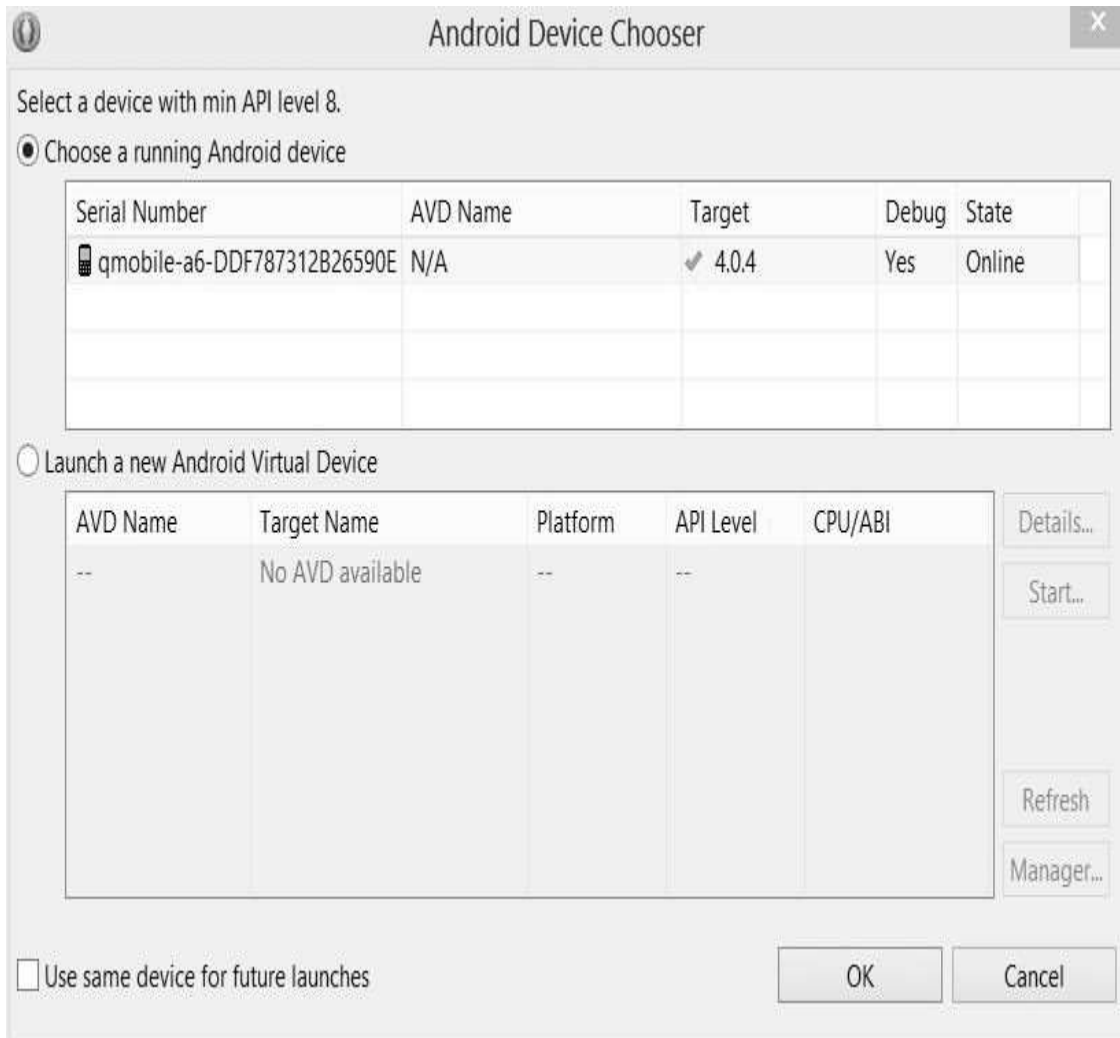
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.networkconnection.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Let's try to run your NetworkConnection application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



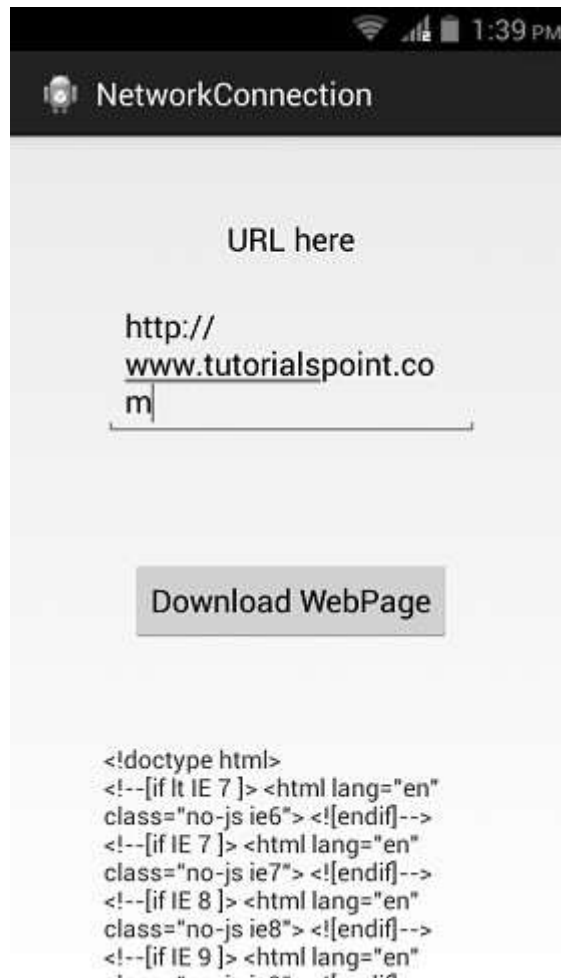
Select your mobile device as an option and then check your mobile device which will display following screen:



Now just type in your website whose HTML you want to fetch. In our case we are typing tutorialspoint.com. It is shown in the figure:



Now press the Download WebPage button and wait for a few seconds and HTML will be downloaded and will be shown to you. It is shown in the figure below:



55. NFC GUIDE

NFC stands for **Near Field Communication**, and as the name implies it provides a wireless communication mechanism between two compatible devices. NFC is a short range wireless technology having a range of 4cm or less for two devices to share data.

How It Works:

Like Bluetooth and WiFi, and all manner of other wireless signals, NFC works on the principle of sending information over radio waves. NFC data is sent through electromagnetic induction between two devices.

NFC works on the basis of tags, it allows you to share some amount of data between an NFC tag and an android powered device or between two android powered devices. Tags have various set of complexities. The Data stored in the tag can be written in a variety of formats, but android APIs are based around a NFC standard called as **NFC Data Exchange Format (NDEF)**.

The transmission frequency for data across NFC is 13.56 megahertz, and data can be sent at either 106, 212 or 424 kilobytes per second, which is quick enough for a range of data transfers from contact details to swapping pictures, songs and videos.

Android powered devices with NFC supports following three main modes of operations:

Three Modes of Operation

- **Reader/Writer Mode:**

It allows the NFC device to read or write passive NFC tags.

- **P2P mode:**

This mode allows NFC device to exchange data with other NFC peers.

- **Card emulation mode:**

It allows the NFC device itself to act as an NFC card, so it can be accessed by an external NFC reader.

How it works with Android:

To get the permission to access NFC Hardware, add the following permission in your Android.Manifest file.

```
<uses-sdk android:minSdkVersion="10"/>
```

First thing to note is that not all android powered devices provide NFC technology. So to make sure that your application shows up in google play for only those devices that have NFC Hardware, add the following line in your **Android.Manifest** file.

```
<uses-feature android:name="android.hardware.nfc"
android:required="true"/>
```

Android provides an android.nfc package for communicating with another device. This package contains following classes:

Sr.No	Classes
1	NdefMessage It represents an immutable NDEF Message. .
2	NdefRecord It represents an immutable NDEF Record.
3	NfcAdapter It represents the local NFC adapter.
4	NfcEvent It wraps information associated with any NFC event.
5	NfcManager It is a high level manager used to obtain an instance of an NfcAdapter.
6	Tag It represents an NFC tag that has been discovered.

NFC tags system works in android with the help of some intent filters that are listed below:

Sr.No	Filters & Features
1	<p>ACTION_NDEF_DISCOVERED</p> <p>This intent is used to start an Activity when a tag contains an NDEF payload.</p>
2	<p>ACTION_TECH_DISCOVERED</p> <p>This intent is used to start an activity if the tag does not contain NDEF data, but is of known technology.</p>
3	<p>ACTION_TAG_DISCOVERED</p> <p>This intent is started if no activities handle the ACTION_NDEF_DISCOVERED or ACTION_TECH_DISCOVERED intents.</p>

To code an application that uses NFC technology is complex so don't use it in your app unless necessary. The use of NFC is not common in devices but it is getting popular. Let's see what is the future of this technology:

Future Applications

With this technology growing day by day and due to introduction of contact-less payment systems this technology is getting a boom. A service known as **Google Wallet** is already introduced in the US whose purpose is to make our smartphones a viable alternative to credit and transport cards.

56. PHP/MYSQL

Here, in this chapter, we are going to explain, how you can integrate PHP and MYSQL with your android application. This is very useful in case you have a webserver, and you want to access its data on your android application.

MYSQL is used as a database at the webserver and PHP is used to fetch data from the database. Our application will communicate with the PHP page with necessary parameters and PHP will contact MYSQL database and will fetch the result and return the results to us.

PHP - MYSQL

Creating Database

MYSQL database can be created easily using this simple script. The **CREATE DATABASE** statement creates the database.

```
<?php
$con=mysqli_connect("example.com","username","password");
$sql="CREATE DATABASE my_db";
if (mysqli_query($con,$sql))
{
    echo "Database my_db created successfully";
}
?>
```

Creating Tables

Once database is created, it is time to create some tables in the database. The **CREATE TABLE** statement creates the database.

```
<?php
$con=mysqli_connect("example.com","username","password","my_db");
$sql="CREATE TABLE table1(Username CHAR(30),Password CHAR(30),Role
CHAR(30))";
if (mysqli_query($con,$sql))
{
    echo "Table have been created successfully";
}
```

```
}
?>
```

Inserting Values in tables

When the database and tables are created, it is time to insert some data into the tables. The **Insert Into** statement creates the database.

```
<?php
$con=mysqli_connect("example.com","username","password","my_db");
$sql="INSERT INTO table1 (FirstName, LastName, Age) VALUES ('admin',
'admin','adminstrator')";
if (mysqli_query($con,$sql))
{
    echo "Values have been inserted successfully";
}
?>
```

PHP - GET and POST methods

PHP is also used to fetch the record from the mysql database once it is created. In order to fetch record some information must be passed to PHP page regarding what record to be fetched.

The first method to pass information is through GET method in which **\$_GET** command is used. The variables are passed in the url and the record is fetched. Its syntax is given below:

```
<?php
$con=mysqli_connect("example.com","username","password","database name");
if (mysqli_connect_errno($con))
{
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
$username = $_GET['username'];
$password = $_GET['password'];
$result = mysqli_query($con,"SELECT Role FROM table1 where
Username='$username' and Password='$password'");
$row = mysqli_fetch_array($result);
```



```

$data = $row[0];
if($data){
echo $data;
}
mysqli_close($con);
?>

```

The second method is to use POST method. The only change in the above script is to replace `$_GET` with **`$_POST`**. In Post method, the variables are not passed through URL.

Android - Connecting MYSQL

Connecting Via Get Method

There are two ways to connect to MYSQL via PHP page. The first one is called **Get method**. We will use **HttpGet** and **HttpClient** class to connect. Their syntax is given below:

```

URL url = new URL(link);
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet();
request.setURI(new URI(link));

```

After that you need to call **execute** method of HttpClient class and receive it in a HttpResponse object. After that you need to open streams to receive the data.

```

HttpResponse response = client.execute(request);
BufferedReader in = new BufferedReader
(new InputStreamReader(response.getEntity().getContent()));

```

Connecting Via Post Method

In the Post method, the **URLEncoder**, **URLConnection** class will be used. The urlencoder will encode the information of the passing variables. It's syntax is given below:

```

URL url = new URL(link);
String data = URLEncoder.encode("username", "UTF-8")
+ "=" + URLEncoder.encode(username, "UTF-8");
data += "&" + URLEncoder.encode("password", "UTF-8")

```

```
+ "=" + URLEncoder.encode(password, "UTF-8");
URLConnection conn = url.openConnection();
```

The last thing you need to do is to write this data to the link. After writing, you need to open stream to receive the responded data.

```
OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
wr.write( data );
BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
```

Example:

The below example is a complete example of connecting your android application with MYSQL database via PHP page. It creates a basic application that allows you to login using GET and POST method.

PHP - MYSQL part

In this example a database with the name of temp has been created at 000webhost.com. In that database, a table has been created with the name of table1. This table has three fields. (Username, Password, Role). The table has only one record which is ("admin","admin","administrator").

The php page has been given below which takes parameters by post method.

```
<?php
$con=mysqli_connect("mysql10.000webhost.com","username","password","db_name");
if (mysqli_connect_errno($con))
{
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
$username = $_POST['username'];
$password = $_POST['password'];
$result = mysqli_query($con,"SELECT Role FROM table1 where
Username='$username' and Password='$password'");
$row = mysqli_fetch_array($result);
$data = $row[0];
if($data){
```

```

echo $data;
}
mysqli_close($con);
?>

```

Android Part

To experiment with this example, you need to run this on an actual device on which wifi internet is connected.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as PHPMYSQL under a package com.example.phpmysql. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Activity code.
3	Create src/SigninActivity.java file to add PHPMYSQL code.
4	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
5	Modify res/values/string.xml file and add necessary string components.
6	Modify AndroidManifest.xml to add necessary permissions.
7	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of src/com.example.phpmysql/MainActivity.java.

```

package com.example.phpmysql;

import android.app.Activity;
import android.os.Bundle;

```

```
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

    private EditText usernameField,passwordField;
    private TextView status,role,method;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        usernameField = (EditText)findViewById(R.id.editText1);
        passwordField = (EditText)findViewById(R.id.editText2);
        status = (TextView)findViewById(R.id.textView6);
        role = (TextView)findViewById(R.id.textView7);
        method = (TextView)findViewById(R.id.textView9);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    public void login(View view){
        String username = usernameField.getText().toString();
        String password = passwordField.getText().toString();
        method.setText("Get Method");
        new SigninActivity(this,status,role,0).execute(username,password);
    }
}
```

```

public void loginPost(View view){
    String username = usernameField.getText().toString();
    String password = passwordField.getText().toString();
    method.setText("Post Method");
    new SigninActivity(this,status,role,1).execute(username,password);

}

}

```

Here is the content of src/com.example.phpmysql/SigninActivity.java.

```

package com.example.phpmysql;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.URI;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import android.content.Context;
import android.os.AsyncTask;
import android.widget.TextView;

public class SigninActivity extends AsyncTask<String,Void,String>{

    private TextView statusField,roleField;
    private Context context;

```

```

private int byGetOrPost = 0;
//flag 0 means get and 1 means post.(By default it is get.)
public SigninActivity(Context context,TextView statusField,
TextView roleField,int flag) {
    this.context = context;
    this.statusField = statusField;
    this.roleField = roleField;
    byGetOrPost = flag;
}

protected void onPreExecute(){

}
@Override
protected String doInBackground(String... arg0) {
    if(byGetOrPost == 0){ //means by Get Method
        try{
            String username = (String)arg0[0];
            String password = (String)arg0[1];
            String link =
                "http://myphpmysqlweb.hostei.com/login.php?username="
                +username+"&password="+password;
            URL url = new URL(link);
            HttpClient client = new DefaultHttpClient();
            HttpGet request = new HttpGet();
            request.setURI(new URI(link));
            HttpResponse response = client.execute(request);
            BufferedReader in = new BufferedReader
                (new InputStreamReader(response.getEntity().getContent()));

            StringBuffer sb = new StringBuffer("");
            String line="";
            while ((line = in.readLine()) != null) {
                sb.append(line);
            }
        }
    }
}

```

```

        break;
    }
    in.close();
    return sb.toString();
}catch(Exception e){
    return new String("Exception: " + e.getMessage());
}
}
else{
    try{
        String username = (String)arg0[0];
        String password = (String)arg0[1];
        String link="http://myphpmysqlweb.hostei.com/loginpost.php";
        String data = URLEncoder.encode("username", "UTF-8")
+ "=" + URLEncoder.encode(username, "UTF-8");
        data += "&" + URLEncoder.encode("password", "UTF-8")
+ "=" + URLEncoder.encode(password, "UTF-8");
        URL url = new URL(link);
        URLConnection conn = url.openConnection();
        conn.setDoOutput(true);
        OutputStreamWriter wr = new OutputStreamWriter
(conn.getOutputStream());
        wr.write( data );
        wr.flush();
        BufferedReader reader = new BufferedReader
(new InputStreamReader(conn.getInputStream()));
        StringBuilder sb = new StringBuilder();
        String line = null;
        // Read Server Response
        while((line = reader.readLine()) != null)
        {
            sb.append(line);
            break;
        }
    }
}

```

```

        return sb.toString();
    }catch(Exception e){
        return new String("Exception: " + e.getMessage());
    }
}
}
@Override
protected void onPostExecute(String result){
    this.statusField.setText("Login Successful");
    this.roleField.setText(result);
}
}
}

```

Here is the content of **activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="25dp"
        android:ems="10"
        android:inputType="textPassword" >

```



```
</EditText>

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="44dp"
    android:ems="10" >

<requestFocus android:layout_width="wrap_content" />

</EditText>

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editText1"
    android:layout_alignParentLeft="true"
    android:text="@string/Username" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:text="@string/App"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView
    android:id="@+id/textView7"
```

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_alignBottom="@+id/textView5"  
android:layout_alignLeft="@+id/textView6"  
android:text="@string/Role"  
android:textAppearance="?android:attr/textAppearanceMedium"  
android:textSize="10sp" />
```

```
<TextView
```

```
    android:id="@+id/textView5"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/textView6"  
    android:layout_marginTop="27dp"  
    android:layout_toLeftOf="@+id/editText1"  
    android:text="@string/LoginRole" />
```

```
<TextView
```

```
    android:id="@+id/textView8"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/textView6"  
    android:layout_alignLeft="@+id/textView5"  
    android:layout_marginBottom="27dp"  
    android:text="@string/method" />
```

```
<TextView
```

```
    android:id="@+id/textView4"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/textView8"  
    android:layout_below="@+id/button1"  
    android:layout_marginTop="86dp"  
    android:text="@string/LoginStatus" />
```

```
<TextView
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/textView4"
    android:layout_centerHorizontal="true"
    android:text="@string/Status"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textSize="10sp" />
```

```
<TextView
    android:id="@+id/textView9"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/textView8"
    android:layout_alignLeft="@+id/textView6"
    android:text="@string/Choose"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textSize="10sp" />
```

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:layout_toRightOf="@+id/textView6"
    android:onClick="loginPost"
    android:text="@string/LoginPost" />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

        android:layout_alignBaseline="@+id/button2"
        android:layout_alignBottom="@+id/button2"
        android:layout_alignLeft="@+id/textView2"
        android:onClick="login"
        android:text="@string/LoginGet" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/editText2"
    android:layout_alignBottom="@+id/editText2"
    android:layout_alignParentLeft="true"
    android:text="@string/Password" />

</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">PHPMYSQL</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Username">Username</string>
    <string name="Password">Password</string>
    <string name="LoginGet">Login - Get</string>
    <string name="LoginPost">Login - Post</string>
    <string name="App">Login Application</string>
    <string name="LoginStatus">Login Status</string>
    <string name="LoginRole">Login Role</string>
    <string name="Status">Not login</string>
    <string name="Role">Not assigned</string>
    <string name="method">Login Method</string>

```

```

    <string name="Choose">Choose Method</string>

</resources>

```

Here is the content of **AndroidManifest.xml**.


```

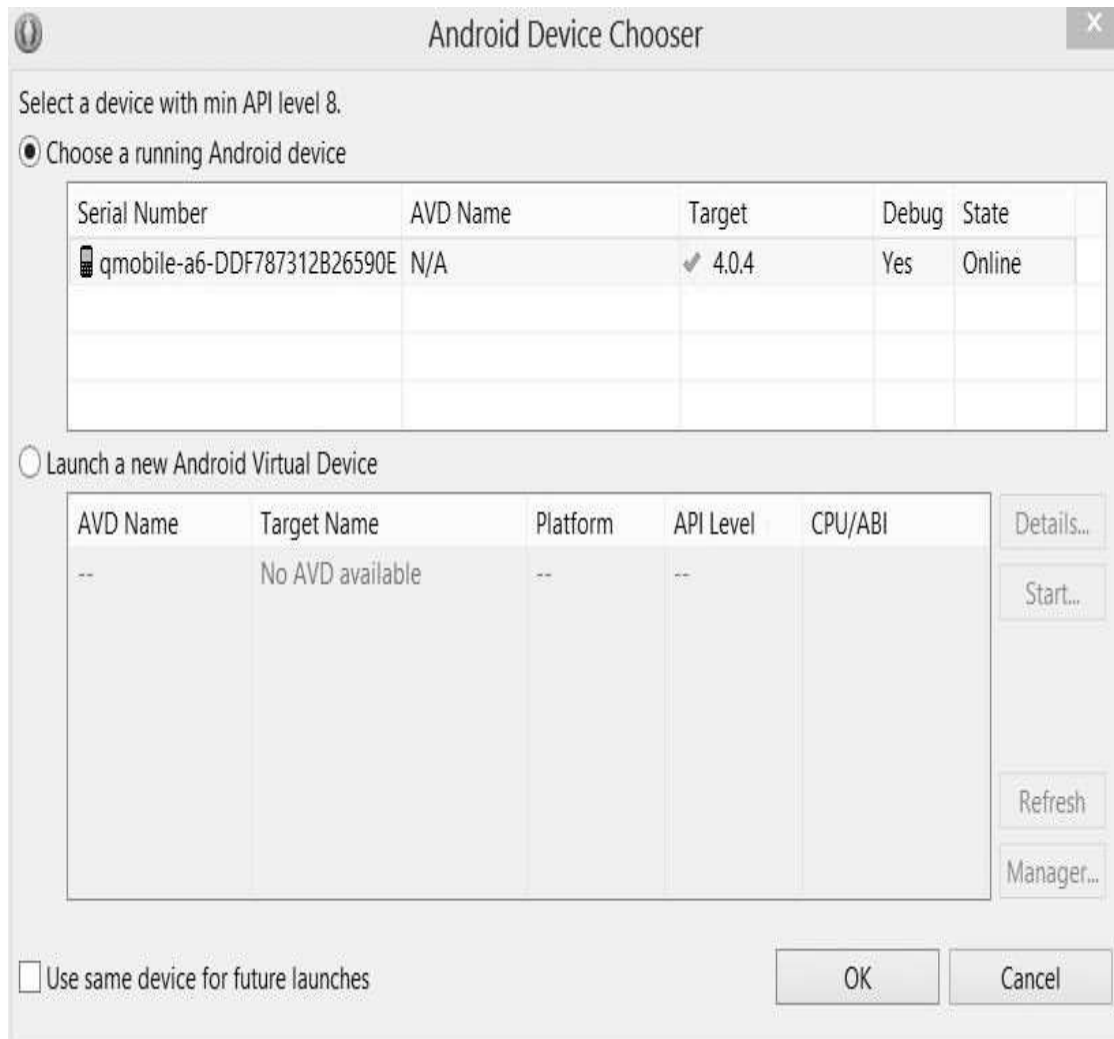
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.phpmysql"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.phpmysql.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

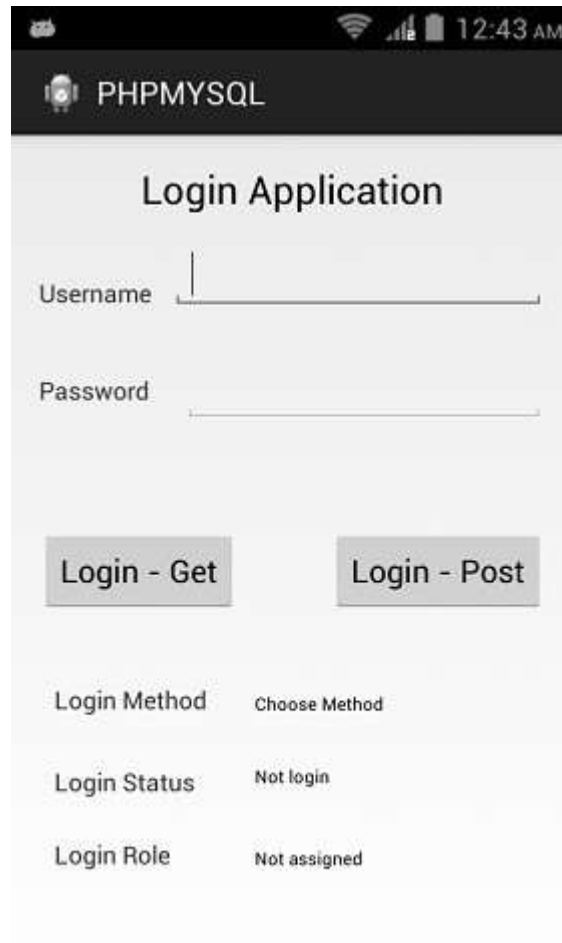
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

Let's try to run your PHPMYSQL application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Now just type in your username and password. In our case we are typing admin as username and password. It is shown in the figure:

PHPMYSQL

Login Application

Username

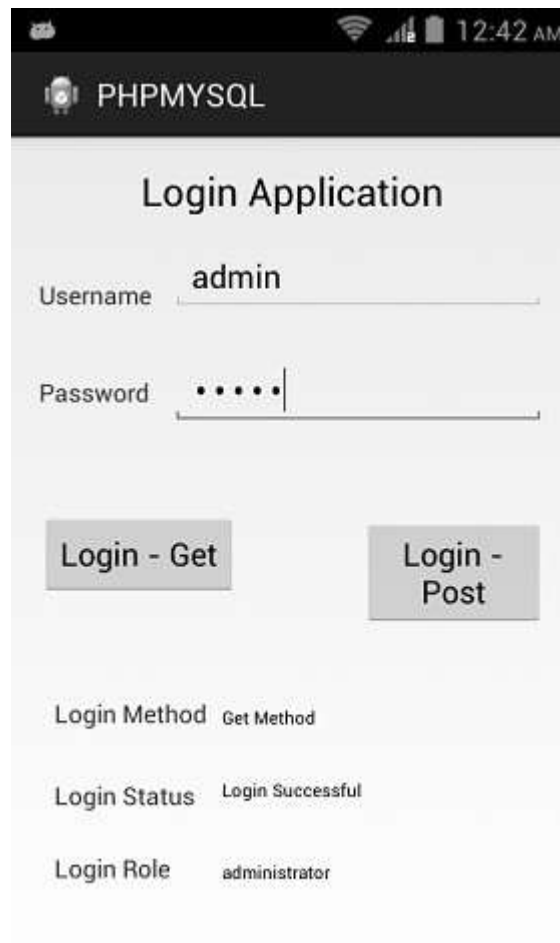
Password

Login Method: Choose Method

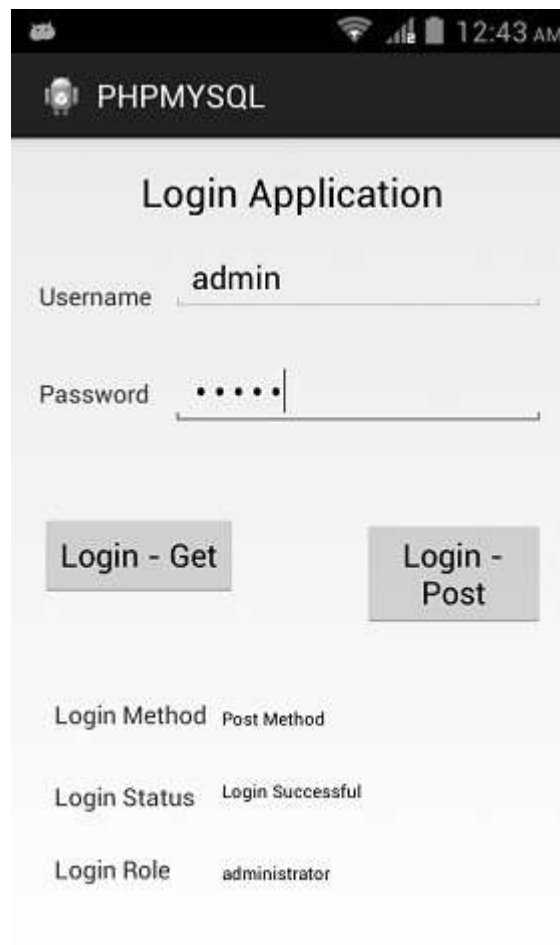
Login Status: Not login

Login Role: Not assigned

Now press the Get button, wait a few seconds and response will be downloaded and will be shown to you. In this case, the response is the ROLE that is fetched in case of admin as username and password. It is shown in the figure below:



Now again press the POST button and same result would appear. It is shown in the figure below:



57. PROGRESS CIRCLE

The easiest way to make a progress circle is using a class called ProgressDialog. The loading bar can also be made through that class. The only logical difference between bar and circle is, that the former is used when you know the total time for waiting for a particular task whereas the latter is used when you do not know the waiting time.

In order to this, you need to instantiate an object of this class. Its syntax is.

```
ProgressDialog progress = new ProgressDialog(this);
```

Now you can set some properties of this dialog. Such as, its style, its text etc.

```
progress.setMessage("Downloading Music :) ");  
progress.setProgressStyle(ProgressDialog.STYLE_SPINNER);  
progress.setIndeterminate(true);
```

Apart from these methods, there are other methods that are provided by the ProgressDialog class.

Sr. No	Style and description
1	getMax() This method returns the maximum value of the progress.
2	incrementProgressBy(int diff) This method increment the progress bar by the difference of value passed as a parameter.
3	setIndeterminate(boolean indeterminate) This method sets the progress indicator as determinate or indeterminate.
4	setMax(int max) This method sets the maximum value of the progress dialog.
5	setProgress(int value) This method is used to update the progress dialog with some specific

	value.
6	<p>show(Context context, CharSequence title, CharSequence message)</p> <p>This is a static method, used to display progress dialog.</p>

Example:

This example demonstrates the spinning use of the progress dialog. It display a spinning progress dialog on pressing the button.

To experiment with this example, you need to run this on an actual device on after developing the application according to the steps below.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as ProgressDialog under a package com.example.progressdialog. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add progress code to display the spinning progress dialog.
3	Modify res/layout/activity_main.xml file to add respective XML code.
4	Modify res/values/string.xml file to add a message as a string constant.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.progressdialog/MainActivity.java**.

```
package com.example.progressdialog;

import com.example.progressdialog.R;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {

    private ProgressDialog progress;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        progress = new ProgressDialog(this);
    }

    public void open(View view){
        progress.setMessage("Downloading Music :) ");
        progress.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progress.setIndeterminate(true);
        progress.show();

        final int totalProgressTime = 100;

        final Thread t = new Thread(){

            @Override
            public void run(){

                int jumpTime = 0;
                while(jumpTime < totalProgressTime){
                    try {
                        sleep(200);
```

```

        jumpTime += 5;
        progress.setProgress(jumpTime);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    }

}

};

t.start();

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
}

```

Modify the content of **res/layout/activity_main.xml** to the following:

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

```

```

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="150dp"
    android:onClick="open"
    android:text="@string/download_button" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="19dp"
    android:text="@string/download_text"
    android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Modify the **res/values/string.xml** to the following:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ProgressDialog</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="download_button">Download</string>
    <string name="download_text">Press the button to download
music</string>
</resources>

```

This is the default **AndroidManifest.xml**:


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.progressdialog"
    android:versionCode="1"
    android:versionName="1.0" >

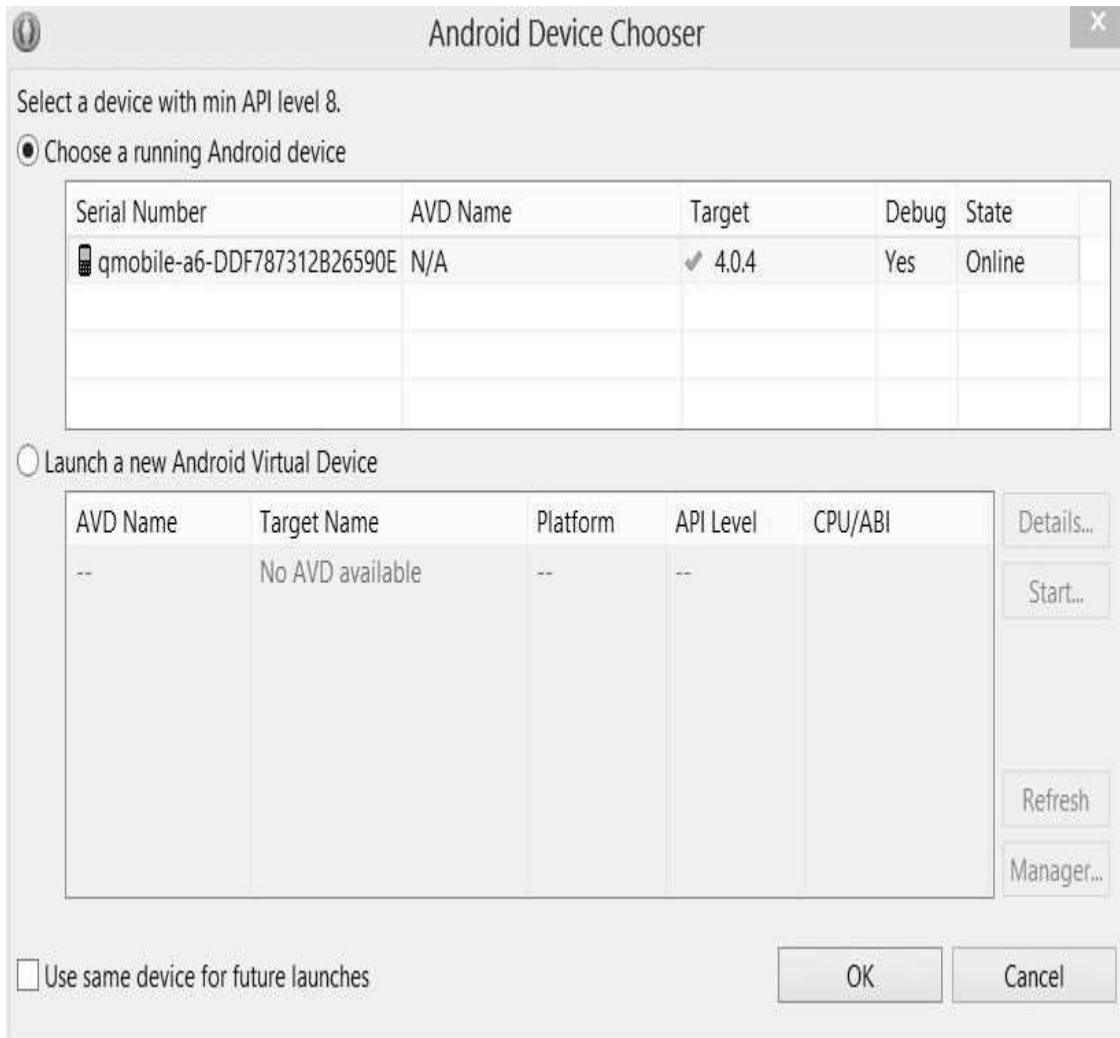
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.progressdialog.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

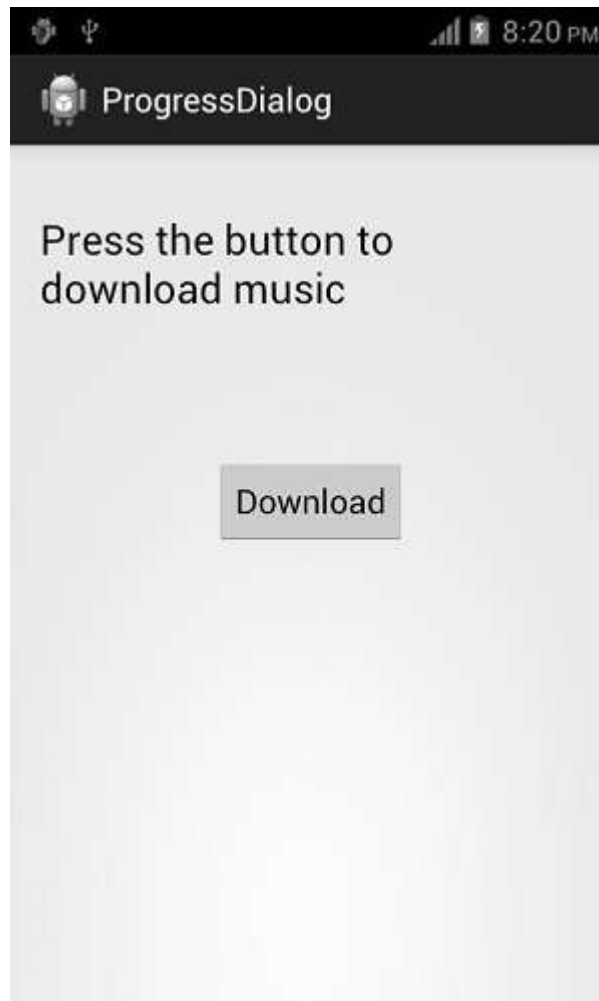
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

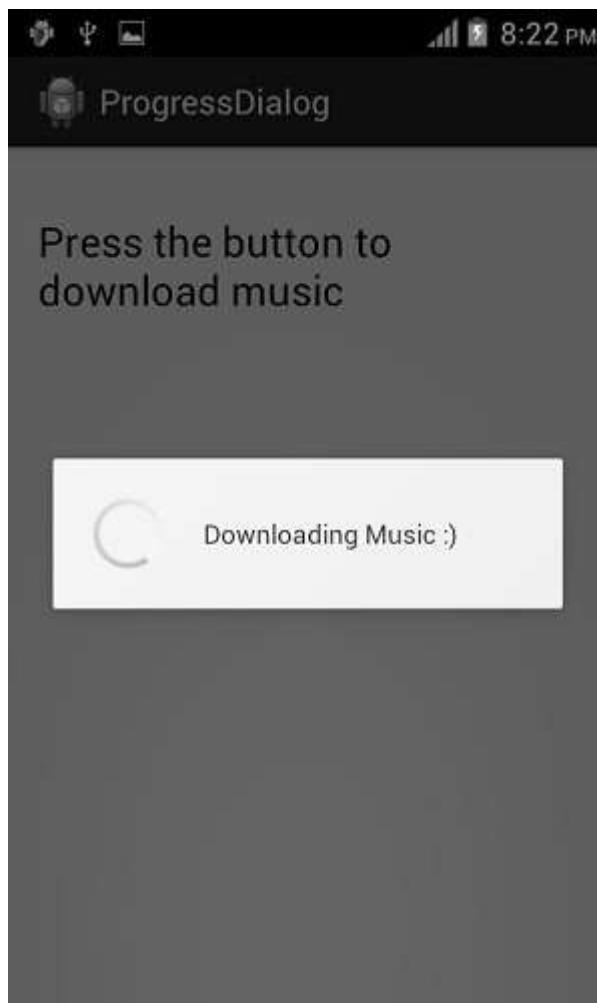

Let's try to run your ProgressDialog application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Just press the button to start the Progress Dialog. After pressing, following screen would appear:



58. PROGRESS BAR USING PROGRESS DIALOG

Progress bars are used to show progress of a task. For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.

In android there is a class called ProgressDialog that allows you to create progress bar. In order to do this, you need to instantiate an object of this class. Its syntax is.

```
ProgressDialog progress = new ProgressDialog(this);
```

Now you can set some properties of this dialog. Such as, its style, its text etc.

```
progress.setMessage("Downloading Music :) ");  
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);  
progress.setIndeterminate(true);
```

Apart from these methods, there are other methods that are provided by the ProgressDialog class

Sr. NO	Title and description
1	getMax() This method returns the maximum value of the progress.
2	incrementProgressBy(int diff) This method increments the progress bar by the difference of value passed as a parameter.
3	setIndeterminate(boolean indeterminate) This method sets the progress indicator as determinate or indeterminate.
4	setMax(int max) This method sets the maximum value of the progress dialog.
5	setProgress(int value)

	This method is used to update the progress dialog with some specific value.
6	show(Context context, CharSequence title, CharSequence message) This is a static method, used to display progress dialog.

Example:

This example demonstrates the horizontal use of the progress dialog which is in fact a progress bar. It display a progress bar on pressing the button.

To experiment with this example, you need to run this on an actual device after developing the application according to the steps below.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as ProgressDialog under a package com.example.progressdialog. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add progress code to display the progress dialog.
3	Modify res/layout/activity_main.xml file to add respective XML code.
4	Modify res/values/string.xml file to add a message as a string constant.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.progressdialog/MainActivity.java**.

```
package com.example.progressdialog;

import com.example.progressdialog.R;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.app.ProgressDialog;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {

    private ProgressDialog progress;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        progress = new ProgressDialog(this);
    }

    public void open(View view){
        progress.setMessage("Downloading Music :) ");
        progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        progress.setIndeterminate(true);
        progress.show();

        final int totalProgressTime = 100;

        final Thread t = new Thread(){

            @Override
            public void run(){

                int jumpTime = 0;
                while(jumpTime < totalProgressTime){
                    try {
```

```

        sleep(200);
        jumpTime += 5;
        progress.setProgress(jumpTime);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    }

}
};
t.start();

}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
}

```

Modify the content of **res/layout/activity_main.xml** to the following:

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

```

```

tools:context=".MainActivity" >

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="150dp"
    android:onClick="open"
    android:text="@string/download_button" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="19dp"
    android:text="@string/download_text"
    android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

Modify the **res/values/string.xml** to the following:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ProgressDialog</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="download_button">Download</string>
    <string name="download_text">Press the button to download
music</string>
</resources>

```


This is the default **AndroidManifest.xml**:


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.progressdialog"
    android:versionCode="1"
    android:versionName="1.0" >

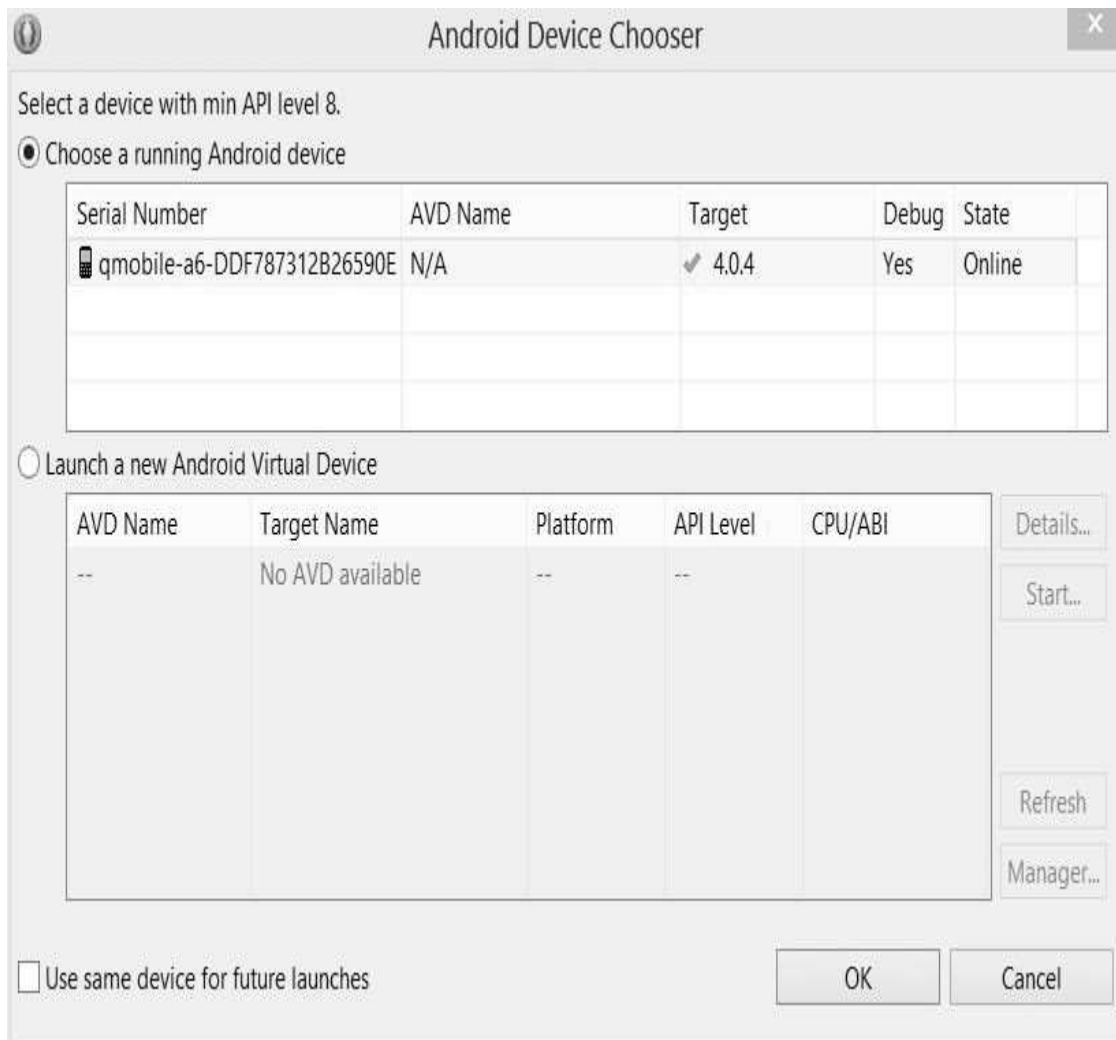
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.progressdialog.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

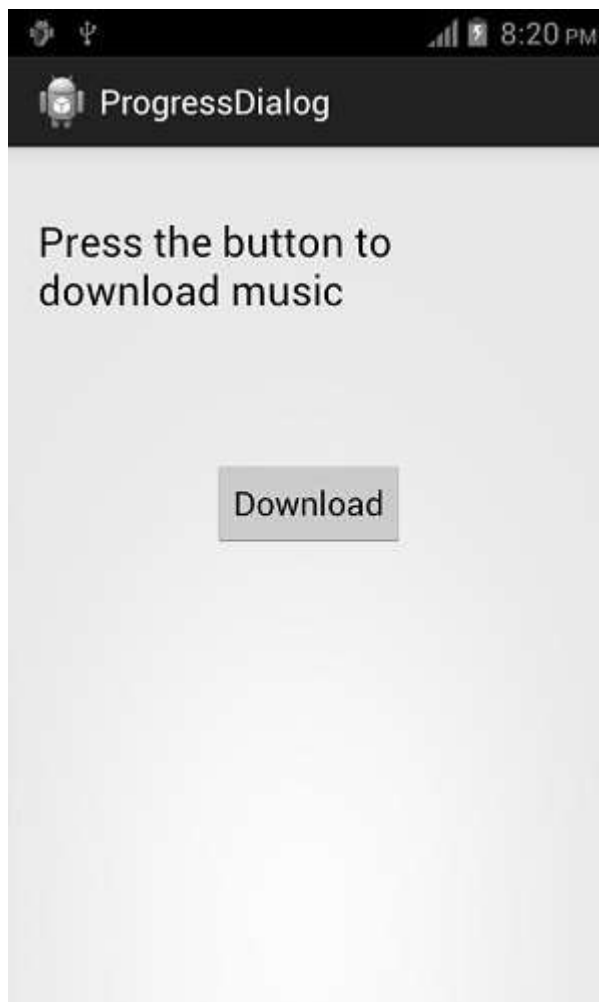
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

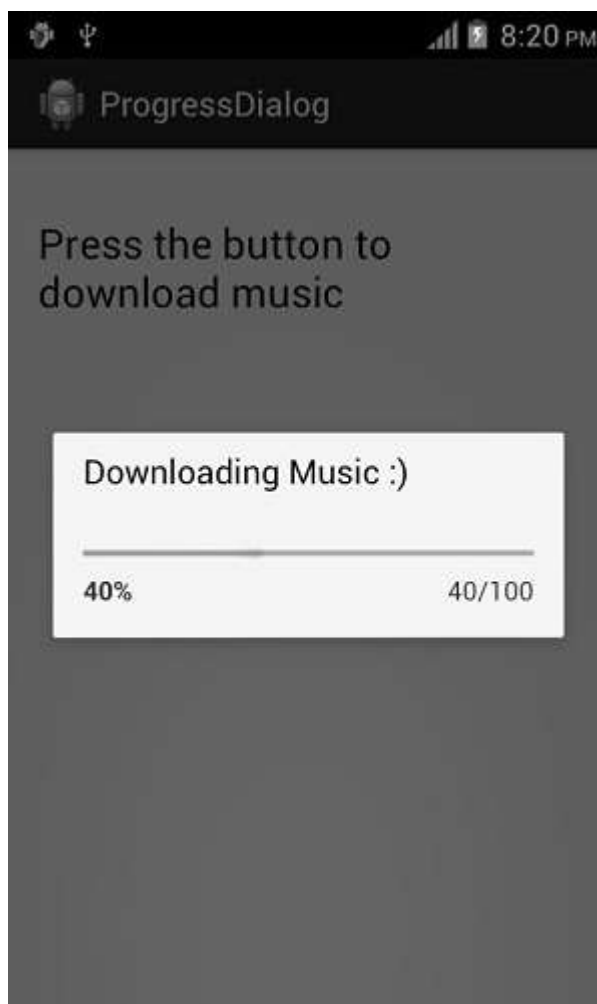
Let's try to run your ProgressDialog application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



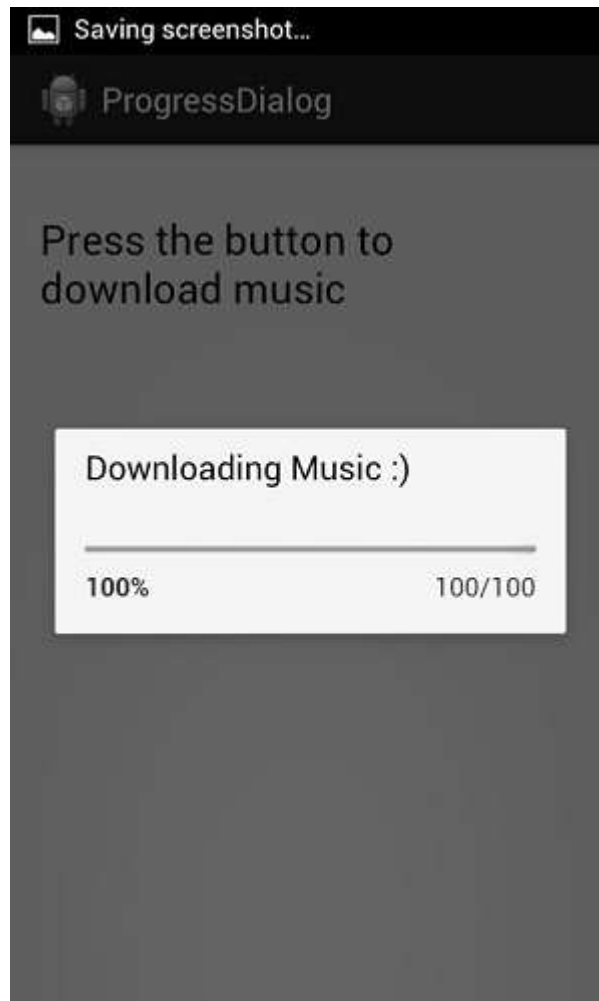
Select your mobile device as an option and then check your mobile device which will display following screen:



Just press the button to start the Progress bar. After pressing, following screen would appear:



It will continuously update itself, and after few seconds, it would appear something like this.



59. PUSH NOTIFICATION

A notification is a message you can display to the user outside of your application's normal UI. You can create your own notifications in android very easily.

Android provides **NotificationManager** class for this purpose. In order to use this class, you need to instantiate an object of this class by requesting the android system through **getSystemService()** method. Its syntax is given below:

```
NotificationManager NM;  
NM=(NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
```

After that you will create Notification through **Notification** class and specify its attributes such as icon, title and time etc. Its syntax is given below:

```
Notification notify=new  
Notification(android.R.drawable.stat_notify_more,title,System.currentTimeMillis());
```

The next thing you need to do is to create a **PendingIntent** by passing context and intent as a parameter. By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself.

```
PendingIntent pending=PendingIntent.getActivity(getApplicationContext(),  
0, new Intent(),0);
```

The last thing you need to do is to call **setLatestEventInfo** method of the Notification class and pass the pending intent along with notification subject and body details. Its syntax is given below. And then finally call the notify method of the NotificationManager class.

```
notify.setLatestEventInfo(getApplicationContext(), subject,  
body,pending);  
NM.notify(0, notify);
```

Apart from the notify method, there are other methods available in the NotificationManager class. They are listed below:

Sr.No	Method & description
1	cancel(int id) This method cancels a previously shown notification.
2	cancel(String tag, int id) This method also cancels a previously shown notification.
3	cancelAll() This method cancels all previously shown notifications.
4	notify(int id, Notification notification) This method posts a notification to be shown in the status bar.
5	notify(String tag, int id, Notification notification) This method also Post a notification to be shown in the status bar.

Example:

The below example demonstrates the use of NotificationManager class. It creates a basic application that allows you to create a notification.

To experiment with this example, you need to run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Status under a package com.example.status. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Notification code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string

	components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.status/MainActivity.java**.

```
package com.example.status;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    NotificationManager NM;
    EditText one,two,three;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        one = (EditText)findViewById(R.id.editText1);
        two = (EditText)findViewById(R.id.editText2);
        three = (EditText)findViewById(R.id.editText3);
    }

    @Override
```



```

public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@SuppressWarnings("deprecation")
public void notify(View vobj){
    String title = one.getText().toString();
    String subject = two.getText().toString();
    String body = three.getText().toString();

    NM=(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    Notification notify=new Notification(android.R.drawable.
    stat_notify_more,title,System.currentTimeMillis());
    PendingIntent pending=PendingIntent.getActivity(
    getApplicationContext(),0, new Intent(),0);

    notify.setLatestEventInfo(getApplicationContext(),subject,body,pending);
    NM.notify(0, notify);
}
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

```

```
android:paddingRight="@dimen/activity_horizontal_margin"  
android:paddingTop="@dimen/activity_vertical_margin"  
tools:context=".MainActivity" >
```

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentBottom="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginBottom="86dp"  
    android:onClick="notify"  
    android:text="@string/notification" />
```

```
<EditText  
    android:id="@+id/editText1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentRight="true"  
    android:layout_alignParentTop="true"  
    android:layout_marginTop="53dp"  
    android:ems="10" />
```

```
<EditText  
    android:id="@+id/editText2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/editText1"  
    android:layout_below="@+id/editText1"  
    android:layout_marginTop="28dp"  
    android:ems="10" />
```

```
<EditText  
    android:id="@+id/editText3"
```

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_alignLeft="@+id/editText2"  
android:layout_below="@+id/editText2"  
android:layout_marginTop="23dp"  
android:ems="10" />
```

```
<TextView
```

```
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editText1"  
    android:layout_marginRight="14dp"  
    android:layout_toLeftOf="@+id/editText1"  
    android:text="@string/title" />
```

```
<TextView
```

```
    android:id="@+id/textView2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/editText3"  
    android:layout_alignRight="@+id/textView1"  
    android:text="@string/heading" />
```

```
<TextView
```

```
    android:id="@+id/textView3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editText3"  
    android:layout_alignLeft="@+id/textView2"  
    android:text="@string/body" />
```

```
<TextView
```

```
    android:id="@+id/textView4"
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/textView2"
        android:text="@string/create"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">Status</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="notification">Notify</string>
    <string name="title">Title</string>
    <string name="heading">Subject</string>
    <string name="body">Body</string>
    <string name="create">Create Notification</string>

</resources>

```

Here is the content of **AndroidManifest.xml**.

```


<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.status"
    android:versionCode="1"
    android:versionName="1.0" >

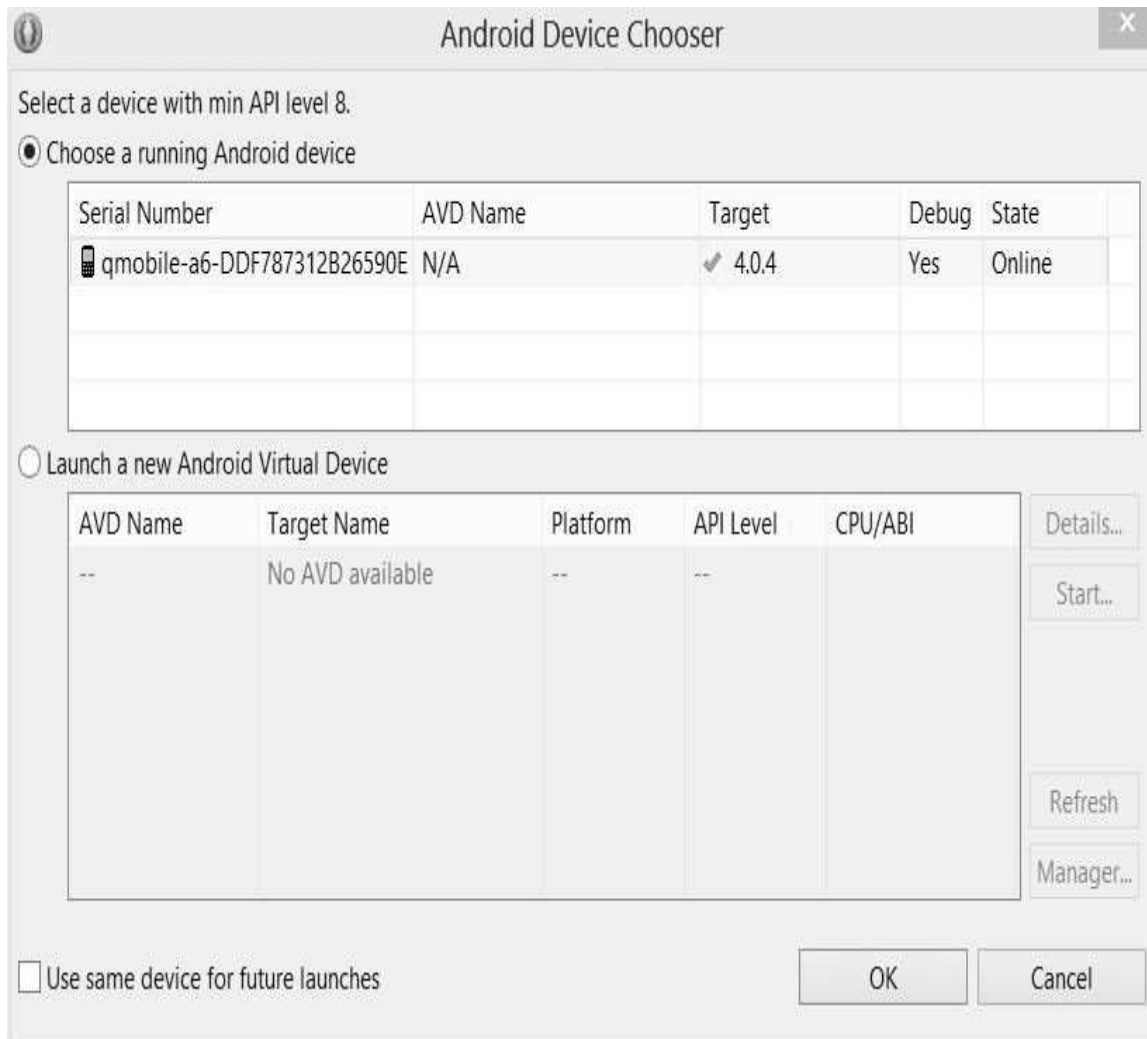
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

```

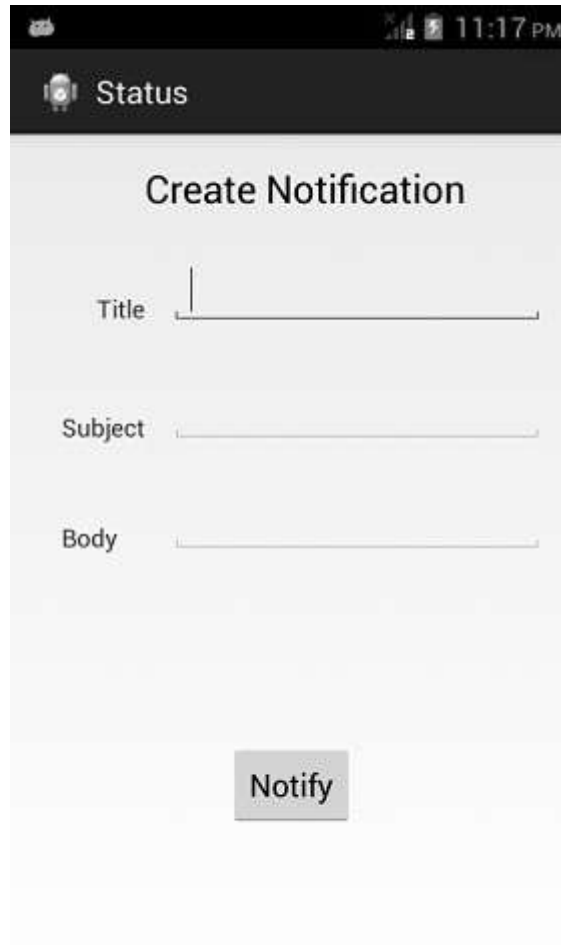
```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <activity
    android:name="com.example.status.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />

      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
</application>
</manifest>
```

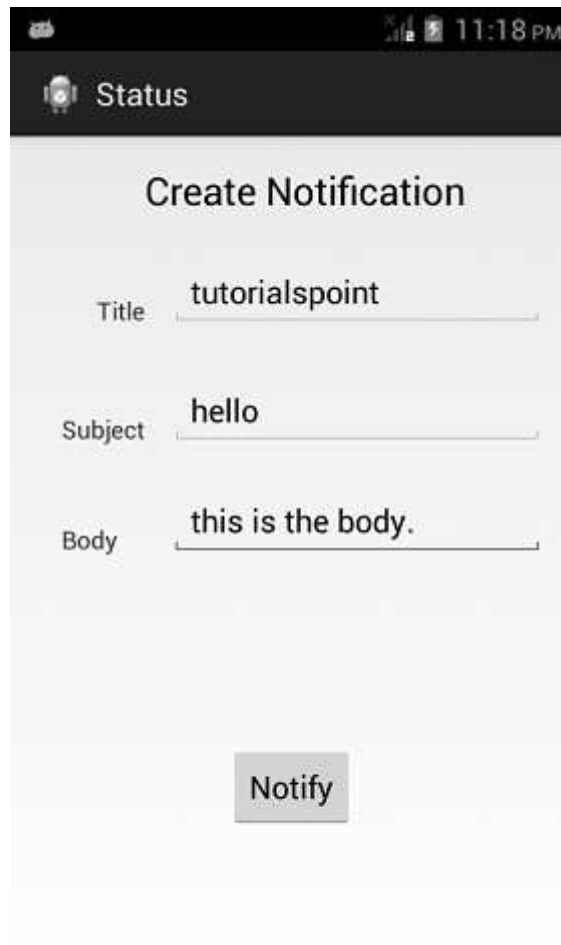
Let's try to run your TextToSpeech application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:

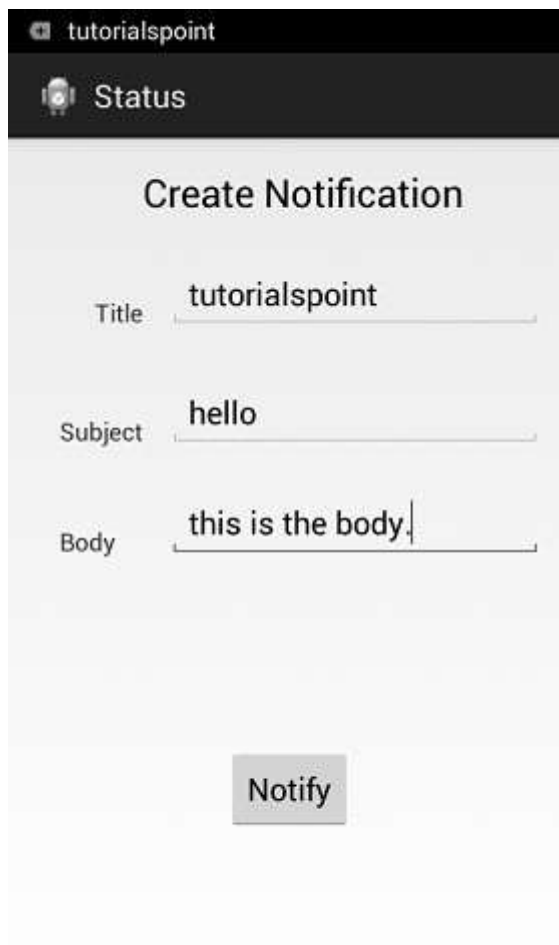


Now fill in the field with the title, subject and the body. This has been shown below in the figure:



The screenshot shows an Android application interface. At the top, there is a dark header bar with the word "Status" and an Android robot icon. Below the header, the main content area is titled "Create Notification". There are three text input fields arranged vertically. The first field is labeled "Title" and contains the text "tutorialspoint". The second field is labeled "Subject" and contains the text "hello". The third field is labeled "Body" and contains the text "this is the body.". At the bottom of the form, there is a rectangular button labeled "Notify". The status bar at the very top of the screen shows the time as 11:18 PM and some system icons.

Now click on the notify button and you will see a notification in the top notification bar. It has been shown below:



tutorialspoint

Status

Create Notification

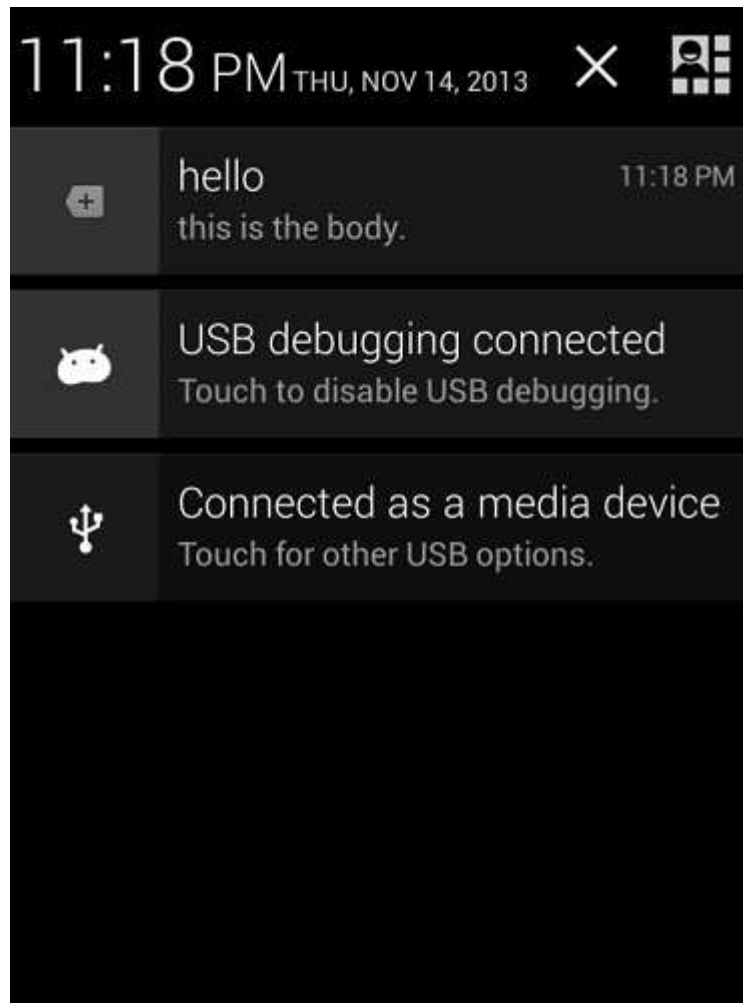
Title

Subject

Body

Notify

Now scroll down the notification bar and see the notification. This has been shown below in the figure:



60. RENDERSCRIPT

This chapter teaches you about Android RenderScript. Usually the apps on android are designed as to consume as minimum resources as possible. But some applications like some 3D games need high level processing on android.

To provide these applications high performance android introduced the RenderScript. It is android based framework which is used for running applications that perform very highly computational tasks. The development on this framework is done in Native Development Kit (NDK) provided by android. RenderScript is extremely useful for applications which performs following types of actions:

- 3D Rendering
- Image Processing
- Computational Photography
- Computer Vision

How RenderScript Works:

RenderScript framework is basically based on data parallel computation. It distributes your application workload on all the processors available on your device like multi-core CPUs or GPUs.

This parallel distribution of workload frees the programmer from the tension of load balancing and work scheduling. You can write more detailed and complex algorithms for your app without the worry of computational power.

How to Begin:

To use the RenderScript Framework you must have following two things:

- A RenderScript Kernel
- RenderScript APIs

A RenderScript Kernel

A kernel is a program which manages data processing instructions and manage workload on Central Processing Units. A kernel is a fundamental part of the operating system.

Similarly to run the RenderScript framework we need a written script named as Kernel to manage all the data processing requests from our app and utilize more features of the android OS provided by the NDK and as mentioned earlier that

the development of RenderScript is done in the Native Development Kit of Android.

The Kernel Script is written in C-99 standard of C-language. This Standard was before the development of C++. A RenderScript kernel script file usually placed in **.rs** file. Each file is called as a script. A RenderScript Kernel script can contain following elements:

Sr.No	Elements
1	<p>A Language declaration</p> <p>It declares the version of RenderScript Kernel language used in this script.</p>
2	<p>A package declaration</p> <p>This declaration names the package name of the Java class which will be affected by this Kernel Code.</p>
3	<p>Invokable functions</p> <p>You can call these invokable functions from your JAVA code with arbitrary arguments.</p>
4	<p>Script Globals Variables</p> <p>These are just like the variables defined in C and C++ programming language. You can access these variables from your JAVA code.</p>

Following is the Sample Code of a Kernel:

```

uchar4 __convert__((kernel)) invert(uchar4 in, uint32_t x, uint32_t y) {
    uchar4 out = in;
    out.r = 255 - in.r;
    out.g = 255 - in.g;
    return out;
}
    
```

RenderScript APIs

If you want to use RenderScript in your API, you can do it in following two ways:

Sr.No	APIs
1	android.renderscript This API is available on devices running Android 3.0 and higher.
2	android.support.v8.renderscript This API is available on devices running Android 2.2 and higher.

To android support library following tools are required:

- Android SDK Tools version 22.2
- Android SDK Build-tools version 18.1.0

How to use RenderScript Support Library

First Open the **project.properties** file in your project and add following lines in the file:

```
renderscript.target=18
renderscript.support.mode=true
sdk.buildtools=18.1.0
```

Now open your main class which use RenderScript and add an import for the Support Library classes as following:

```
import android.support.v8.renderscript.*;
```

Following are the purposes of above mentioned properties that we add in the **project.properties** file.

Sr.No	Project properties
1	renderscript.target It specifies the bytecode version to be generated.
2	renderscript.support.mode It specifies a compatible version for the generated bytecode to fall back.

3**sdk.buildtools**

It specifies the versions of Android SDK build tools to use.

Now call your RenderScript Kernel functions and compute complex algorithms in your app.

61. RSS READER

RSS stands for Really Simple Syndication. RSS is an easy way to share your website updates and content with your users so that users might not have to visit your site daily for any kind of updates.

RSS Example

RSS is a document that is created by the website with .xml extension. You can easily parse this document and show it to the user in your application. An RSS document looks like this.

```
<rss version="2.0">
<channel>
  <title>Sample RSS</title>
  <link>http://www.google.com</link>
  <description>World's best search engine</description>
</channel>
</rss>
```

RSS Elements

An RSS document such as above has the following elements.

Sr.No	Component & description
1	channel This element is used to describe the RSS feed.
2	title Defines the title of the channel.
3	link Defines the hyperlink to the channel.
4	description

	Describes the channel.
--	------------------------

Parsing RSS

Parsing an RSS document is more like parsing XML. So now lets see how to parse an XML document.

For this, we will create XmlPullParser object, but in order to create that we will first create XmlPullParserFactory object and then call its newInstance() method to create XmlPullParser. Its syntax is given below:

```
private XmlPullParserFactory xmlFactoryObject =
XmlPullParserFactory.newInstance();

private XmlPullParser myparser = xmlFactoryObject.newPullParser();
```

The next step involves specifying the file for XmlPullParser that contains XML. It could be a file or could be a Stream. In our case it is a stream. Its syntax is given below:

```
mysparser.setInput(stream, null);
```

The last step is to parse the XML. An XML file consist of events, Name, Text, AttributesValue etc. So XmlPullParser has a separate function for parsing each of the component of XML file. Its syntax is given below:

```
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)
{
    String name=myParser.getName();
    switch (event){
        case XmlPullParser.START_TAG:
            break;
        case XmlPullParser.END_TAG:
            if(name.equals("temperature")){
                temperature = myParser.getAttributeValue(null,"value");
            }
            break;
    }
    event = myParser.next();
}
```


The method **getEventType** returns the type of event that happens. e.g.: Document start, tag start etc. The method **getName** returns the name of the tag. Since we are only interested in temperature, we just check in conditional statement to get a temperature tag, we call the method **getAttributeValue** to return us the value of temperature tag.

Apart from these methods, there are other methods provided by this class for better parsing XML files. These methods are listed below:

Sr.No	Method & description
1	<p>getAttributeCount()</p> <p>This method just Returns the number of attributes of the current start tag.</p>
2	<p>getAttributeName(int index)</p> <p>This method returns the name of the attribute specified by the index value.</p>
3	<p>getColumnNumber()</p> <p>This method returns the current column number, starting from 0.</p>
4	<p>getDepth()</p> <p>This method returns the current depth of the element.</p>
5	<p>getLineNumber()</p> <p>Returns the current line number, starting from 1.</p>
6	<p>getNamespace()</p> <p>This method returns the namespace URI of the current element.</p>
7	<p>getPrefix()</p> <p>This method returns the prefix of the current element.</p>
8	<p>getName()</p> <p>This method returns the name of the tag.</p>
9	<p>getText()</p>

	This method returns the text for that particular element.
10	<p>isWhitespace()</p> <p>This method checks whether the current TEXT event contains only whitespace characters.</p>

Example:

Here is an example demonstrating the use of XMLPullParser class. It creates a basic Parsing application that allows you to parse an RSS document present here at <http://tutorialspoint.com/android/sampleXML.xml> and then shows the result.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as RSSReader under a package com.example.rssreader. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Create a new java file under src/HandleXML.java to fetch and parse XML data.
6	Modify AndroidManifest.xml to add necessary internet permission.
7	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.rssreader/MainActivity.java**.

```
package com.example.rssreader;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private String
        finalUrl="http://tutorialspoint.com/android/sampleXML.xml";
    private HandleXML obj;
    private EditText title,link,description;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        title = (EditText)findViewById(R.id.editText1);
        link = (EditText)findViewById(R.id.editText2);
        description = (EditText)findViewById(R.id.editText3);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

```

public void fetch(View view){
    obj = new HandleXML(finalUrl);
    obj.fetchXML();
    while(obj.parsingComplete);
        title.setText(obj.getTitle());
        link.setText(obj.getLink());
        description.setText(obj.getDescription());
    }
}

```

Following is the content of the java file **src/com.example.rssreader/HandleXML.java**.

```

package com.example.rssreader;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;

import android.util.Log;

public class HandleXML {

    private String title = "title";
    private String link = "link";
    private String description = "description";

    private String urlString = null;
    private XmlPullParserFactory xmlFactoryObject;
    public volatile boolean parsingComplete = true;
    public HandleXML(String url){
        this.urlString = url;
    }
}

```

```
}
public String getTitle(){
    return title;
}
public String getLink(){
    return link;
}
public String getDescription(){
    return description;
}
public void parseXMLAndStoreIt(XmlPullParser myParser) {
    int event;
    String text=null;
    try {
        event = myParser.getEventType();
        while (event != XmlPullParser.END_DOCUMENT) {
            String name=myParser.getName();
            switch (event){
                case XmlPullParser.START_TAG:
                    break;
                case XmlPullParser.TEXT:
                    text = myParser.getText();
                    break;
                case XmlPullParser.END_TAG:
                    if(name.equals("title")){
                        title = text;
                    }
                    else if(name.equals("link")){
                        link = text;
                    }
                    else if(name.equals("description")){
                        description = text;
                    }
                    else{
```

```

        }
        break;
    }
    event = myParser.next();
}
parsingComplete = false;
} catch (Exception e) {
    e.printStackTrace();
}
}
public void fetchXML(){
    Thread thread = new Thread(new Runnable(){
    @Override
    public void run() {
        try {
            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection)
            url.openConnection();
            conn.setReadTimeout(10000 /* milliseconds */);
            conn.setConnectTimeout(15000 /* milliseconds */);
            conn.setRequestMethod("GET");
            conn.setDoInput(true);
            // Starts the query
            conn.connect();
            InputStream stream = conn.getInputStream();
            xmlFactoryObject = XmlPullParserFactory.newInstance();
            XmlPullParser myparser = xmlFactoryObject.newPullParser();
            myparser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,
            false);
            myparser.setInput(stream, null);
            parseXMLAndStoreIt(myparser);
            stream.close();
        } catch (Exception e) {
        }
    }
}

```

```

    }
    });
    thread.start();
}
}

```

Modify the content of **res/layout/activity_main.xml** to the following:

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="26dp"
        android:text="@string/hello_world"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="48dp"

```

```
android:layout_toLeftOf="@+id/textView1"  
android:text="@string/title" />
```

```
<TextView  
    android:id="@+id/textView3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/textView2"  
    android:layout_below="@+id/textView2"  
    android:layout_marginTop="27dp"  
    android:text="@string/link" />  
<EditText  
    android:id="@+id/editText1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBaseline="@+id/textView2"  
    android:layout_alignBottom="@+id/textView2"  
    android:layout_alignParentRight="true"  
    android:ems="10" >  
<requestFocus />  
</EditText>  
  
<EditText  
    android:id="@+id/editText2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBaseline="@+id/textView3"  
    android:layout_alignBottom="@+id/textView3"  
    android:layout_alignLeft="@+id/editText1"  
    android:ems="10" >  
</EditText>  
  
<EditText  
    android:id="@+id/editText3"
```



```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/textView4"
        android:layout_alignBottom="@+id/textView4"
        android:layout_alignLeft="@+id/editText2"
        android:ems="10" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/editText3"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="37dp"
    android:onClick="fetch"
    android:text="@string/fetch" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"
    android:text="@string/description" />
</RelativeLayout>

```

Modify the **res/values/string.xml** to the following:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">RSSReader</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Sample RSS Reader</string>
    <string name="title">title</string>
    <string name="link">link</string>

```

```

<string name="description">Description</string>
<string name="fetch">Fetch Feed</string>
</resources>

```

This is the default **AndroidManifest.xml**.

```


<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.rssreader"
    android:versionCode="1"
    android:versionName="1.0" >

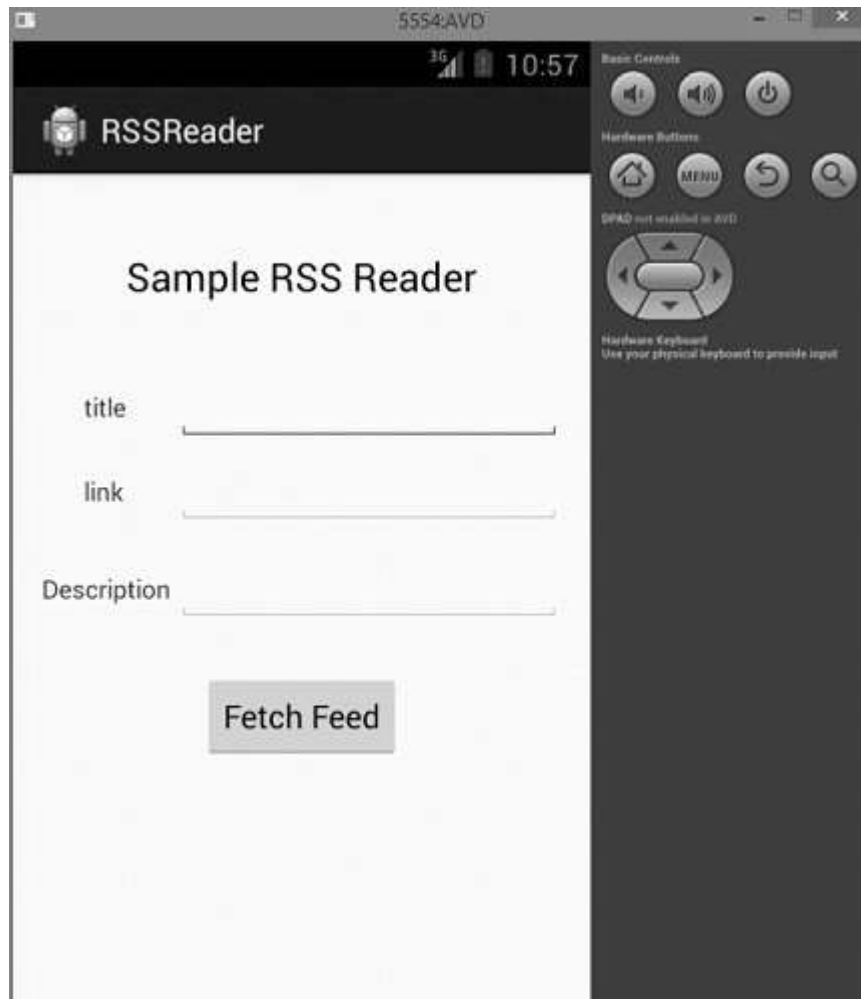
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.rssreader.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

Let's try to run your RSSReader application. 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:



Just press the Fetch Feed button to fetch RSS feed. After pressing, following screen would appear which would show the RSS data.



62. SCREEN CAST

Android Screen cast is a desktop application to control an android device remotely. If your phone is not rooted, you can only view your mobile activity in PC.

But if your phone is rooted, you can communicate both ways. You can also control your device remotely using keyboard and mouse if your phone is rooted.

Screen Cast Steps

The steps of using screen cast has been mentioned below:

Steps	Description
1	You should have latest android SDK installed on your PC.
2	Turn on USB debugging feature on your phone.
3	Connect your pc with phone via data cable.
4	Make sure you have Java Run Time 5 or later installed.
5	Download and open the androidscreeencast application.

Step 1

You can download the latest android SDK from <http://developer.android.com/sdk/index.html> .

Step 2

Turn on USB debugging feature on your device. It is usually found under settings and developer options.

Step 3

Just connect your PC with your phone via the USB data cable.

Step 4

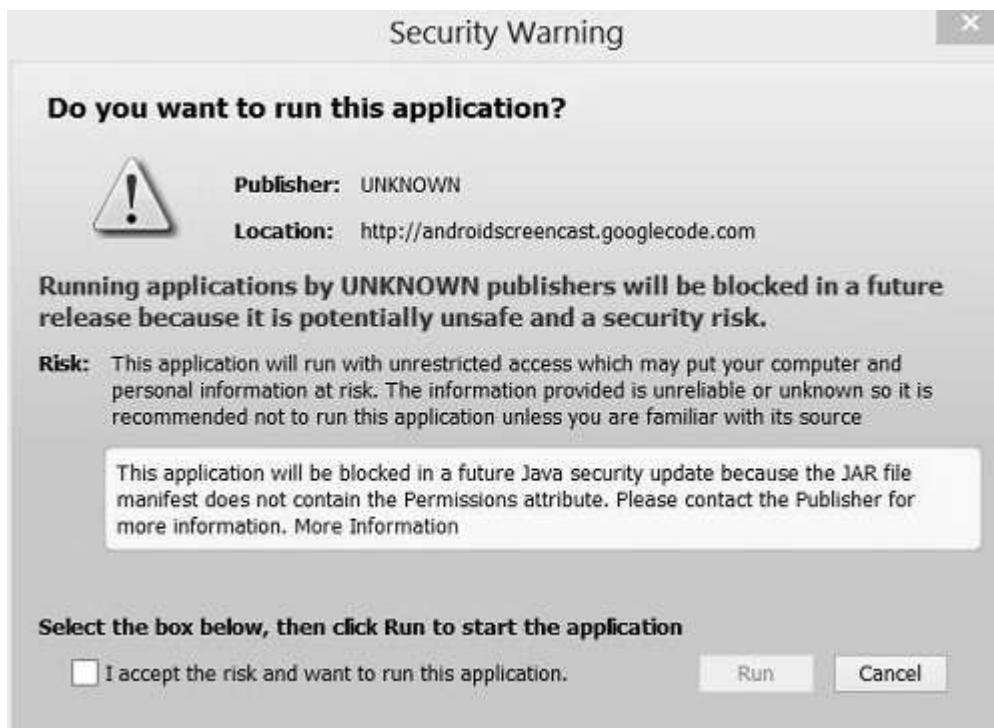
Install Java run time 5 or later, if you have not installed already. You can install it from <http://www.oracle.com/technetwork/java/javase/downloads/index.html> .

Step 5

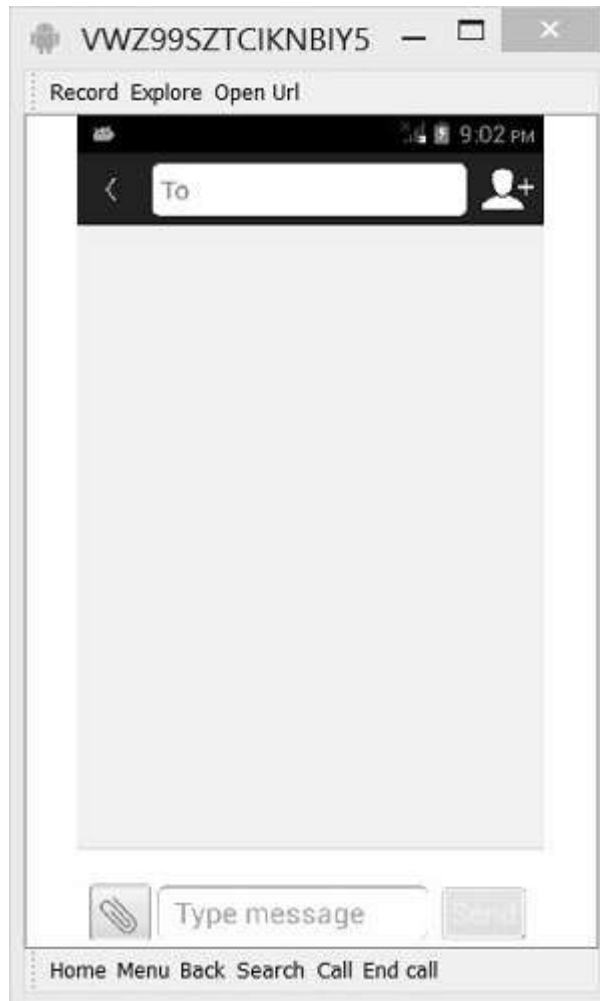
Finally install the androidScreenCast application. You can download it from <https://code.google.com/p/androidscreeencast/> . Once you download it, click to open. It is shown below:



Just wait for a few seconds for the application to load and following pop-up will appear asking your permission to launch this application. Click on accept check box and click on run. It is shown below:



If everything work fine, you will now see your phone on your pc. Navigate through your phone and you will see your mobile working on your pc. It is shown below:



You can see the message application in the above picture, that's because we have opened the messaging application in our mobile. Now type something from your mobile.



As you can see, we have written some text in the sms from our mobile and it appears on PC. So this way you can use this ScreenCast application.

63. SDK MANAGER

To download and install latest android APIs and development tools from the internet, android provide us with android SDK manager. Android SDK Manager separates the APIs, tools and different platforms into different packages which you can download.

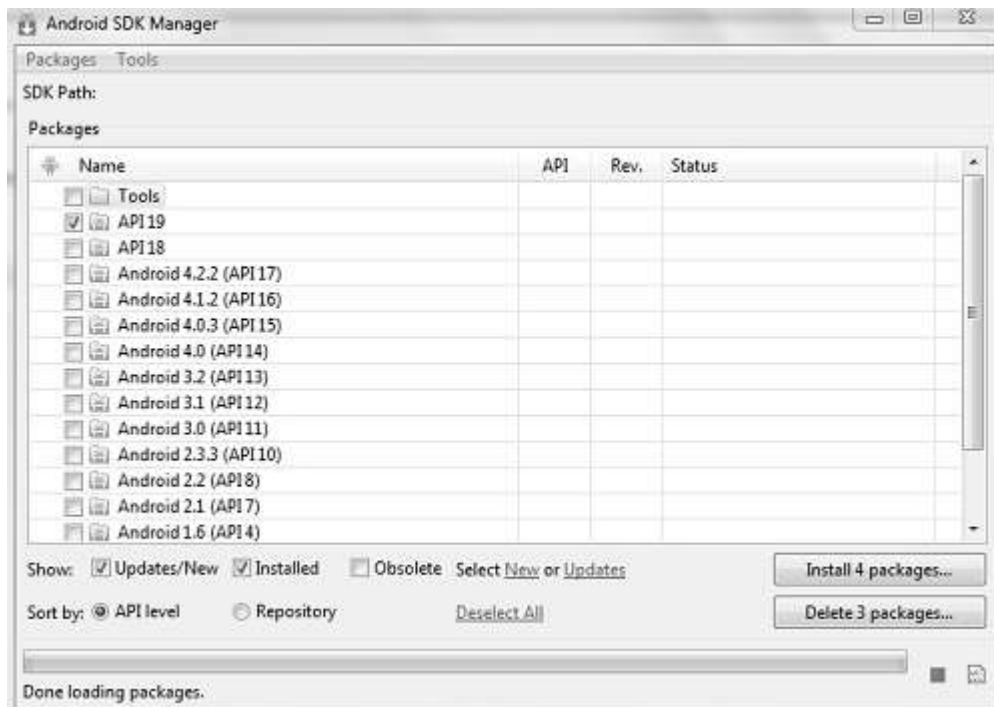
Android SDK manager comes with the Android SDK bundle. You can't download it separately. You can download the android sdk from <http://developer.android.com/sdk/index.html> .

Running Android SDK Manager:

Once downloaded, you can launch Android SDK Manager in one of the following ways:

- Click Window->Android SDK Manager option in Eclipse.
- Double Click on the SDK Manager.exe file in the Android SDK folder.

When it runs you will see the following screen:



You can select which package you want to download by selecting the checkboxes and then click **Install** to install those packages. By default SDK Manager keeps it up to date with latest APIs and other packages.

Once you download the SDK, following packages are available, but first three are necessary to run your SDK and others are recommended.

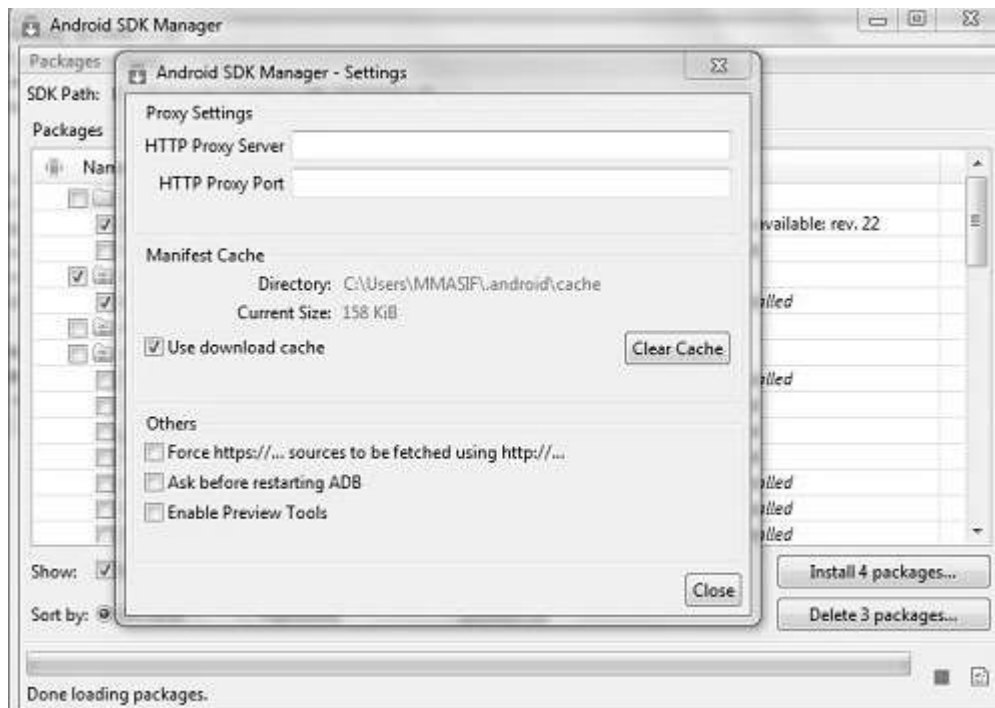
Recommended Packages

Sr.No	Package
1	<p>SDK Tools</p> <p>This is necessary package to run your SDK.</p>
2	<p>SDK Platform-tools</p> <p>This package will be installed once when you first run the SDK manager.</p>
3	<p>SDK Platform</p> <p>Atleast one platform must be installed in your environment to run your application.</p>
4	<p>System Image</p> <p>It's a good practice to download system images for all of the android versions so you can test your app on them with the Android Emulator.</p>
5	<p>SDK Samples</p> <p>This will give you some sample codes to learn about android.</p>

Enabling Proxy in Android SDK Manager

When you run the Android SDK Manager, by default it will check from the Android Repository and Third Party Add-ons and display the available packages to you.

If you want to use proxy, you can do it by clicking on the **Tools-->Options** in the menu. Once you click it, you will see the following screen:



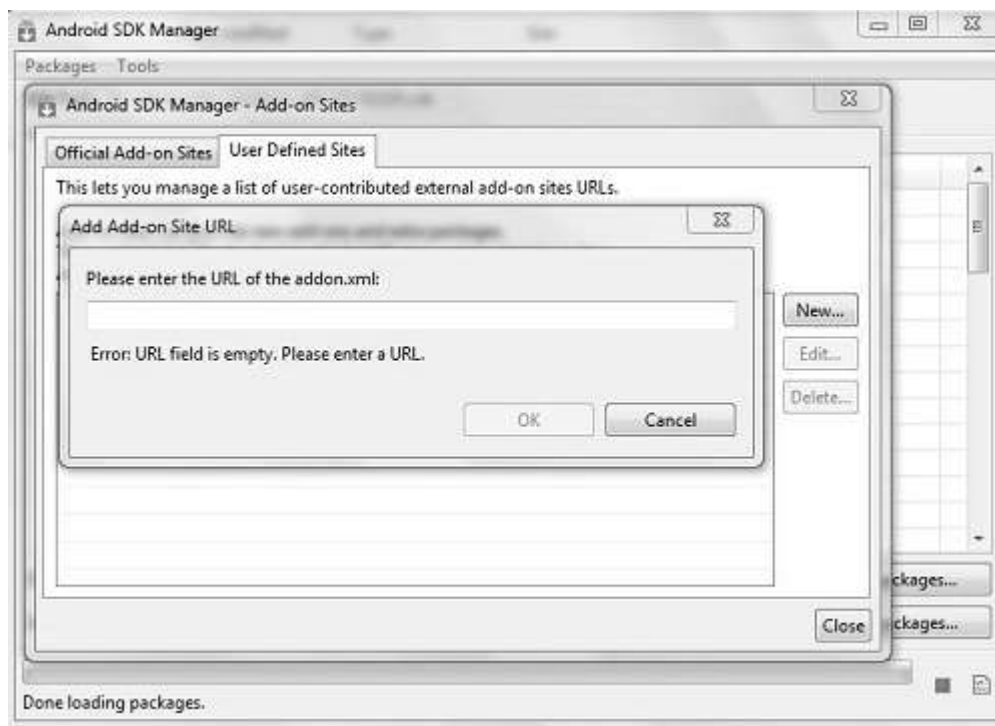
Just Enter the proxy and run your SDK Manager.

Adding New Third Party Sites

If you want to download some Third Party made Android add-ons, you can do it in the SDK manager by following steps:

- Click on the **Tools** option in the menu.
- Click on the **Manage Add-On Sites** option in the sub menu.
- Select the **User Defined Sites** tab.
- Click the **New** button.

Following screen will be displayed:



Just add the URL of Add-on.xml file and click **Ok**. Now you can download the Third Party Add-on in your development environment and use it.

64. SENSORS

Most of the android devices have built-in sensors that measure motion, orientation, and various environmental condition. The android platform supports three broad categories of sensors.

- Motion Sensors
- Environmental sensors
- Position sensors

Some of the sensors are hardware based and some are software based sensors. Whatever the sensor is, android allows us to get the raw data from these sensors and use it in our application. For this, android provides us with some classes.

Android provides `SensorManager` and `Sensor` classes to use the sensors in our application. In order to use sensors, first thing you need to do is to instantiate the object of `SensorManager` class. It can be achieved as follows.

```
SensorManager sMgr;  
sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
```

The next thing you need to do is to instantiate the object of `Sensor` class by calling the `getDefaultSensor()` method of the `SensorManager` class. Its syntax is given below:

```
Sensor light;  
light = sMgr.getDefaultSensor(Sensor.TYPE_LIGHT);
```

Once that sensor is declared, you need to register its listener and override two methods which are `onAccuracyChanged` and `onSensorChanged`. Its syntax is as follows:

```
sMgr.registerListener(this, light, SensorManager.SENSOR_DELAY_NORMAL);  
public void onAccuracyChanged(Sensor sensor, int accuracy) {  
}  
public void onSensorChanged(SensorEvent event) {  
}
```

Getting list of sensors supported.

You can get a list of sensors supported by your device by calling the `getSensorList` method, which will return a list of sensors containing their name and version number and much more information. You can then iterate the list to get the information. Its syntax is given below:

```
sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
List<Sensor> list = sMgr.getSensorList(Sensor.TYPE_ALL);
for(Sensor sensor: list){
}
```

Apart from these methods, there are other methods provided by the `SensorManager` class for managing sensors framework. These methods are listed below:

Sr.No	Method & description
1	getDefaultSensor(int type) This method gets the default sensor for a given type.
2	getOrientation(float[] R, float[] values) This method returns a description of the current primary clip on the clipboard but not a copy of its data.
3	getInclination(float[] I) This method computes the geomagnetic inclination angle in radians from the inclination matrix.
4	registerListener(SensorListener listener, int sensors, int rate) This method registers a listener for the sensor
5	unregisterListener(SensorEventListener listener, Sensor sensor) This method unregisters a listener for the sensors with which it is registered.
6	getOrientation(float[] R, float[] values) This method computes the device's orientation based on the rotation

	matrix.
7	<p>getAltitude(float p0, float p)</p> <p>This method computes the Altitude in meters from the atmospheric pressure and the pressure at sea level.</p>

Example:

Here is an example demonstrating the use of SensorManager class. It creates a basic application that allows you to view the list of sensors on your device.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Sensors under a package com.example.sensors. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.sensors/MainActivity.java**.

```
package com.example.sensors;

import java.util.List;

import android.app.Activity;
```

```
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {

    private SensorManager sMgr;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView sensorsData = (TextView)findViewById(R.id.textView1);

        sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
        List list = sMgr.getSensorList(Sensor.TYPE_ALL);

        StringBuilder data = new StringBuilder();
        for(Sensor sensor: list){
            data.append(sensor.getName() + "\n");
            data.append(sensor.getVendor() + "\n");
            data.append(sensor.getVersion() + "\n");
        }
        sensorsData.setText(data);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```



```

    }
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ScrollView
        android:id="@+id/scrollView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical" >

            <TextView
                android:id="@+id/textView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"

```

```

        android:text="Medium Text"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    </LinearLayout>
</ScrollView>

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">Sensors</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="list">List of sensors supported</string>
</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sensors"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

```

```
<activity
    android:name="com.example.sensors.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>

</manifest>
```

Let's try to run our Sensor 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 following Emulator window:



Now if you will look at your device screen, you will see the list of sensors supported by your device along with their name and version and other information.

If you would run this application on different devices, the output would be different because the output depends upon the number of sensors supported by your device.

65. SESSION MANAGEMENT

Session help you when want to store user data outside your application, so that when the next time user use your application, you can easily get back his details and perform accordingly.

This can be done in many ways. But the most easiest and nicest way of doing this is through **Shared Preferences**.

Shared Preferences

Shared Preferences allows you to save and retrieve data in the form of key value pair. In order to use shared preferences, you have to call a method `getSharedPreferences()` that returns a `SharedPreferences` instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedPreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
```

You can save something in the shared preferences by using `SharedPreferences.Editor` class. You will call the `edit` method of `SharedPreferences` instance and will receive it in an editor object. Its syntax is:

```
Editor editor = sharedPreferences.edit();
editor.putString("key", "value");
editor.commit();
```

Apart from the `putString` method, there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows:

Sr. No.	Mode and description
1	apply() It is an abstract method. It will commit your changes back from editor to the <code>SharedPreferences</code> object you are calling.
2	clear() It will remove all values from the editor.

3	remove(String key) It will remove the value whose key has been passed as a parameter.
4	putLong(String key, long value) It will save a long value in a preference editor.
5	putInt(String key, int value) It will save an integer value in a preference editor.
6	putFloat(String key, float value) It will save a float value in a preference editor.

Session Management through Shared Preferences

To perform session management from shared preferences, we need to check the values or data stored in shared preferences in the **onResume** method. If we do not have the data, we will start the application from the beginning as it is newly installed. But if we have the data, we will start from where the user left it. It is demonstrated in the example below:

Example

The below example demonstrates the use of Session Management. It creates a basic application that allows you to login for the first time. And then when you exit the application without logging out, you will be brought back to the same place if you start the application again. But if you logout from the application, you will be brought back to the main login screen.

To experiment with this example, you need to run this on an actual device or in an emulator .

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as <code>SessionManagement</code> under a package <code>com.example.sessionmanagement</code> . While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify <code>src/MainActivity.java</code> file to add progress code to add session code.

3	Create New Activity and it name as Welcome.java.Edit this file to add progress code to add session code.
4	Modify res/layout/activity_main.xml file to add respective XML code.
5	Modify res/layout/activity_welcome.xml file to add respective XML code.
6	Modify res/values/string.xml file to add a message as a string constant.
7	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content
of **src/com.example.sessionmanagement/MainActivity.java**.

```
package com.example.sessionmanagement;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private EditText username,password;
    public static final String MyPREFERENCES = "MyPrefs" ;
    public static final String name = "nameKey";
    public static final String pass = "passwordKey";
    SharedPreferences sharedPreferences;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    username = (EditText)findViewById(R.id.editText1);
    password = (EditText)findViewById(R.id.editText2);
}

@Override
protected void onResume() {
    sharedPreferences=getSharedPreferences(MyPREFERENCES,
    Context.MODE_PRIVATE);
    if (sharedPreferences.contains(name))
    {
        if(sharedPreferences.contains(pass)){
            Intent i = new Intent(this,com.example.sessionmanagement.
            Welcome.class);
            startActivity(i);
        }
    }
    super.onResume();
}

public void login(View view){
    Editor editor = sharedPreferences.edit();
    String u = username.getText().toString();
    String p = password.getText().toString();
    editor.putString(name, u);
    editor.putString(pass, p);
    editor.commit();
    Intent i = new Intent(this,com.example.
    sessionmanagement.Welcome.class);
    startActivity(i);
}

@Override
```



```

    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

```

Here is the content
of **src/com.example.sessionmanagement/Welcome.java**.

```

package com.example.sessionmanagement;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;

public class Welcome extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_welcome);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.welcome, menu);
    }
}

```

```

        return true;
    }
    public void logout(View view){
        SharedPreferences sharedPreferences = getSharedPreferences
        (MainActivity.MyPREFERENCES, Context.MODE_PRIVATE);
        Editor editor = sharedPreferences.edit();
        editor.clear();
        editor.commit();
        moveTaskToBack(true);
        Welcome.this.finish();
    }
    public void exit(View view){
        moveTaskToBack(true);
        Welcome.this.finish();
    }
}

```

Here is the content of **activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/editText1"

```

```
        android:layout_below="@+id/textView2"
        android:ems="10"
        android:inputType="textPassword" >
</EditText>

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_marginTop="52dp"
    android:text="@string/Username"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignTop="@+id/textView1"
    android:layout_marginRight="16dp"
    android:layout_marginTop="27dp"
    android:ems="10" >
    <requestFocus />
</EditText>

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/textView1"
    android:layout_below="@+id/editText1"
    android:text="@string/Password"
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/editText1"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="22dp"
    android:text="@string/Signin"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/editText2"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="45dp"
    android:onClick="login"
    android:text="@string/Login" />

</RelativeLayout>

```

Here is the content of **activity_welcome.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".Welcome" >

```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="146dp"
    android:onClick="logout"
    android:text="@string/logout" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button1"
    android:layout_alignParentTop="true"
    android:layout_marginTop="64dp"
    android:text="@string/title_activity_welcome"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="43dp"
    android:onClick="exit"
    android:text="@string/exit" />

</RelativeLayout>
```

Here is the content of **Strings.xml**.

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

    <string name="app_name">SessionManagement</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Username">Username</string>
    <string name="Password">Password</string>
    <string name="Signin">Sign In</string>
    <string name="Login">Login</string>
    <string name="logout">Logout</string>
    <string name="title_activity_welcome">Welcome</string>
    <string name="exit">Exit without logout</string>

</resources>
```

Here is the content of **AndroidManifest.xml**.


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sessionmanagement"
    android:versionCode="1"
    android:versionName="1.0" >

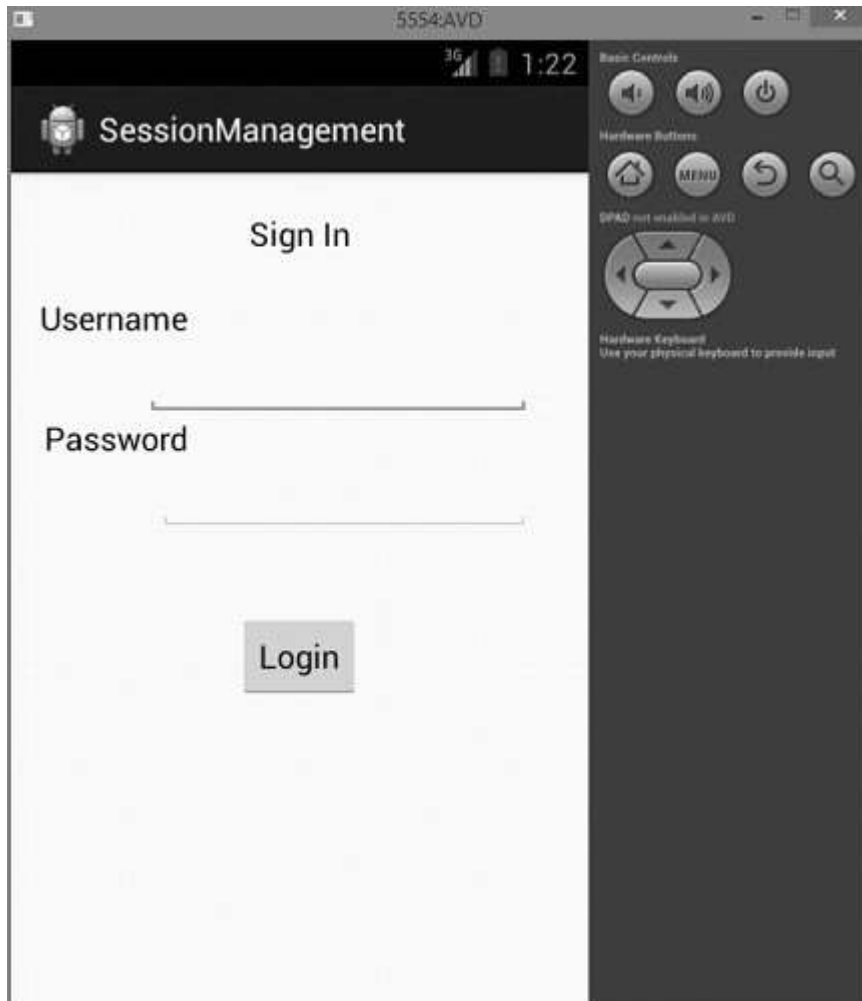
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
```

```
<activity
    android:name="com.example.sessionmanagement.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name="com.example.sessionmanagement.Welcome"
    android:label="@string/title_activity_welcome" >
</activity>
</application>
</manifest>
```

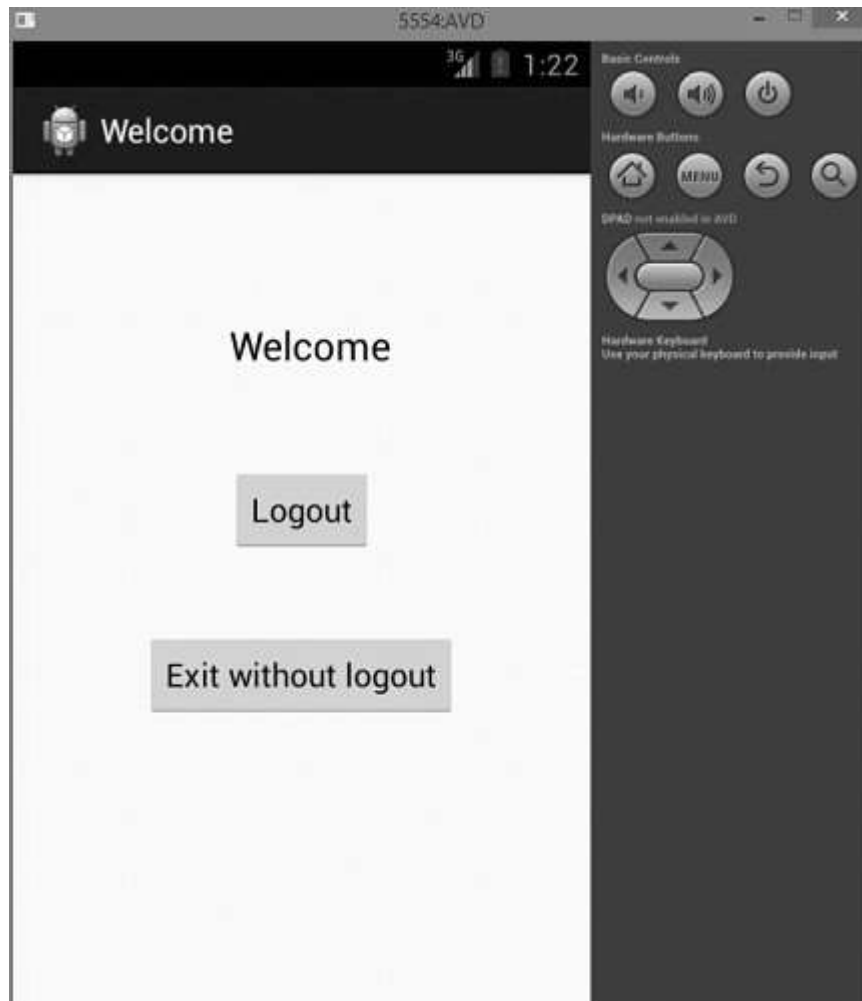
Let's try to run your Session Management application. We assume, you had created your AVD while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is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



Type in your username and password (**type anything you like, but remember what you type**), and click on login button. It is shown in the image below:



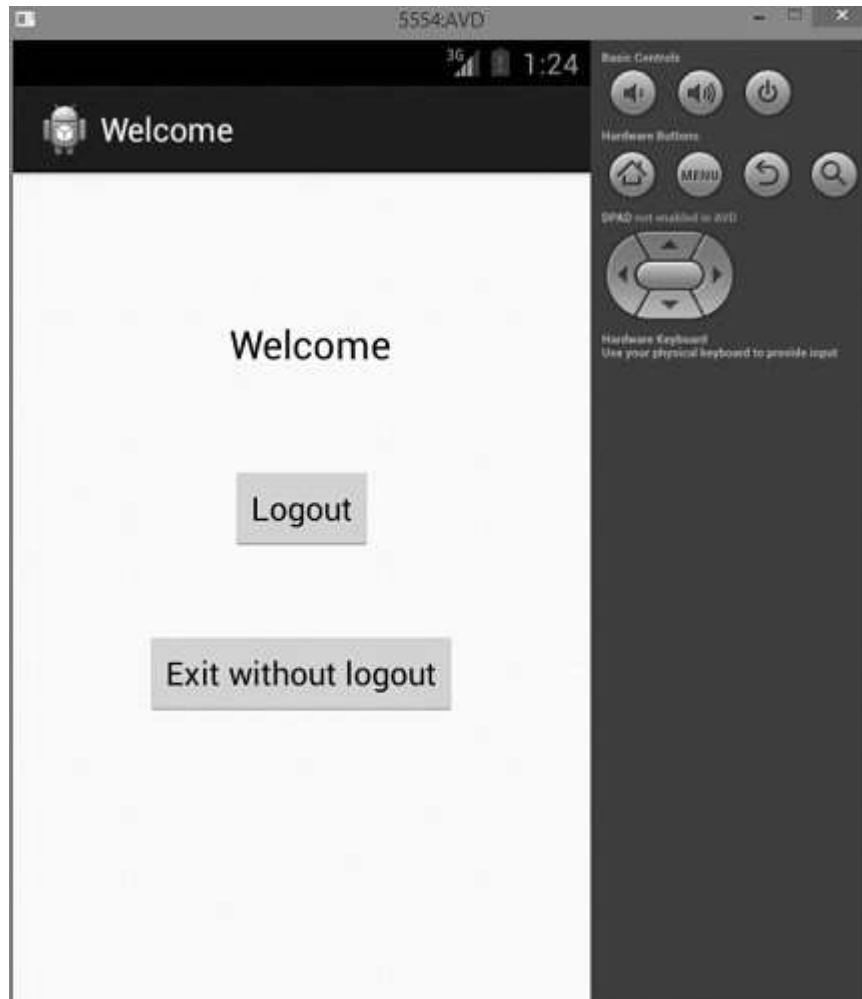
As soon as you click on login button, you will be brought to this Welcome screen. Now your login information is stored in shared preferences.



Now click on **Exit without logout** button and you will be brought back to the home screen. This is shown in the image below:



Now Start the application again. And this time you will not be brought to the login screen, but directly to the welcome screen. This is shown in the image below:



Now click on logout button, and the application will be closed. Now open the application again, and since you have logout your session, so you will be brought back to the front login screen. This is shown in the image below:



66. SIP PROTOCOL

SIP stands for (Session Initiation Protocol). It is a protocol that let applications easily set up outgoing and incoming voice calls, without having to manage sessions, transport-level communication, or audio record or playback directly.

Applications

Some of the common applications of SIP are.

- Video conferencing
- Instant messaging

Requirements

Here are the requirements for developing a SIP application:

- Android OS must be 2.3 or higher
- You must have a data connection or WIFI
- You must have an SIP account in order to use this service.

SIP Classes

Here is a summary of the classes that are included in the Android SIP API:

Sr. NO	Class and description
1	SipAudioCall Handles an Internet audio call over SIP.
2	SipErrorCode Defines error codes returned during SIP actions.
3	SipManager Provides APIs for SIP tasks, such as initiating SIP connections, and provides access to related SIP services.

4	<p>SipProfile</p> <p>Defines a SIP profile, including a SIP account, domain and server. information</p>
5	<p>SipSession</p> <p>Represents a SIP session that is associated with a SIP dialog or a standalone transaction not within a dialog.</p>

Functions of SIP

SIP has following major functions.

- SIP allows for the establishment of user location
- SIP provides a mechanism for call management
- SIP provides feature negotiation, so that all the parties in the call can agree to the features supported among them

Components of SIP

SIP has two major components which are listed below.

- User Agent Client (UAC)
- User Agent Server (UAS)

UAC

UAC or User Agent Client are those end users who generates requests and send those requests to the server. These requests are generated by the client applications running on their systems.

UAS

UAS or User Agent Server are those systems which get the request generated by UAC. The UAS process those requests and then according to the requests it generates responses accordingly.

SipManager

SipManager is an android API for SIP tasks, such as initiating SIP connections, and provides access to related SIP services. This class is the starting point for any SIP actions. You can acquire an instance of it with newInstance().

The SipManager has many functions for managing SIP tasks. Some of the functions are listed below.

Sr. NO	Class and description
1	<p>close(String localProfileUri) Closes the specified profile to not make/receive calls.</p>
2	<p>getCallId(Intent incomingCallIntent) Gets the call ID from the specified incoming call broadcast intent.</p>
3	<p>isOpened(String localProfileUri) Checks if the specified profile is opened in the SIP service for making and/or receiving calls.</p>
4	<p>isSipWifiOnly(Context context) Returns true if SIP is only available on WIFI.</p>
5	<p>isRegistered(String localProfileUri) Checks if the SIP service has successfully registered the profile to the SIP provider (specified in the profile) for receiving calls.</p>
6	<p>isVoipSupported(Context context) Returns true if the system supports SIP-based VOIP API.</p>
7	<p>takeAudioCall(Intent incomingCallIntent, SipAudioCall.Listener listener) Creates a SipAudioCall to take an incoming call.</p>
8	<p>unregister(SipProfile localProfile, SipRegistrationListener listener) Manually unregisters the profile from the corresponding SIP provider to stop receiving further calls.</p>

67. SPELLING CHECKER

The Android platform offers a spelling checker framework that lets you implement and access spell checking in your application.

To use spelling checker, you need to implement **SpellCheckerSessionListener** interface and override its methods. Its syntax is given below:

```
public class HelloSpellCheckerActivity extends Activity implements
SpellCheckerSessionListener {
    @Override
    public void onGetSuggestions(final SuggestionsInfo[] arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onGetSentenceSuggestions(SentenceSuggestionsInfo[] arg0) {
        // TODO Auto-generated method stub
    }
}
```

Next thing you need to do is to create an object of **SpellCheckerSession** class. This object can be instantiated by calling **newSpellCheckerSession** method of TextServicesManager class. This class handles interaction between application and text services. You need to request system service to instantiate it. Its syntax is given below:

```
private SpellCheckerSession mScs;
final TextServicesManager tsm = (TextServicesManager) getSystemService(
Context.TEXT_SERVICES_MANAGER_SERVICE);
mScs = tsm.newSpellCheckerSession(null, null, this, true);
```

The last thing you need to do is to call **getSuggestions** method to get suggestion for any text, you want. The suggestions will be passed onto the **onGetSuggestions** method from where you can do whatever you want.

```
mScs.getSuggestions(new TextInfo(editText1.getText().toString()), 3);
```

This method takes two parameters. First parameter is the string in the form of TextInfo object, and second parameter is the cookie number used to distinguish suggestions.

Apart from the methods, there are other methods provided by the **SpellCheckerSession** class for better handling suggestions. These methods are listed below:

Sr.No	Method & description
1	cancel() Cancels pending and running spell check tasks.
2	close() Finish this session and allow TextServicesManagerService to disconnect the bound spell checker.
3	getSentenceSuggestions(TextInfo[] textInfos, int suggestionsLimit) Get suggestions from the specified sentences.
4	getSpellChecker() Get the spell checker service info, this spell checker session has.
5	isSessionDisconnected() True if the connection to a text service of this session is disconnected and not alive.

Example:

Here is an example demonstrating the use of Spell Checker. It creates a basic spell checking application that allows you to write text and get suggestions.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as HelloSpellCheckerActivity under a package com.example.hellospellchecker. While creating this project, make sure

	you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.hellospellchecker/MainActivity.java**.

```
package com.example.android.hellospellchecker;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.textservice.SentenceSuggestionsInfo;
import android.view.textservice.SpellCheckerSession;
import
android.view.textservice.SpellCheckerSession.SpellCheckerSessionListener;
import android.view.textservice.SuggestionsInfo;
import android.view.textservice.TextInfo;
import android.view.textservice.TextServicesManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class HelloSpellCheckerActivity extends Activity implements
SpellCheckerSessionListener {

    private static final int NOT_A_LENGTH = -1;
```

```
private TextView mMainView;
private SpellCheckerSession mScs;
private EditText editText1;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mMainView = (TextView)findViewById(R.id.main);
    editText1 = (EditText)findViewById(R.id.editText1);
}

@Override
public void onResume() {
    super.onResume();
    final TextServicesManager tsm = (TextServicesManager)
    getSystemService(
    Context.TEXT_SERVICES_MANAGER_SERVICE);
    mScs = tsm.newSpellCheckerSession(null, null, this, true);
}

@Override
public void onPause() {
    super.onPause();
    if (mScs != null) {
        mScs.close();
    }
}

public void go(View view){
    Toast.makeText(getApplicationContext(),
    editText1.getText().toString(),
    Toast.LENGTH_SHORT).show();
    mScs.getSuggestions(new TextInfo(editText1.getText().toString()),
```

```

        3);

    }

    @Override
    public void onGetSuggestions(final SuggestionsInfo[] arg0) {
        final StringBuilder sb = new StringBuilder();

        for (int i = 0; i < arg0.length; ++i) {
            // Returned suggestions are contained in SuggestionsInfo
            final int len = arg0[i].getSuggestionsCount();
            sb.append('\n');
            for (int j = 0; j < len; ++j) {
                sb.append(", " + arg0[i].getSuggestionAt(j));
            }
            sb.append(" (" + len + ")");
        }
        runOnUiThread(new Runnable() {

            public void run() {
                mMainView.append(sb.toString());
            }
        });

    }

    @Override
    public void onGetSentenceSuggestions(SentenceSuggestionsInfo[] arg0) {
        // TODO Auto-generated method stub

    }
}

```

Following is the modified content of the xml **res/layout/main.xml**.

```
<?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/main"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/pre"
    />

    <Button
        android:id="@+id/mainbtn"
        android:layout_width="150dip"
        android:layout_height="50dip"
        android:onClick="go"
        android:text="@string/suggest" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" >

        <requestFocus />
    </EditText>

</LinearLayout>
```

Following is the content of the **res/values/string.xml**.

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

<resources>

    <string name="app_name">HelloSpellChecker</string>
    <string name="suggest">suggest</string>
    <string name="pre">Suggestions</string>

</resources>
```

Following is the content of **AndroidManifest.xml** file.

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


<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.hellospellchecker"
    android:versionCode="1"
    android:versionName="1.0" >

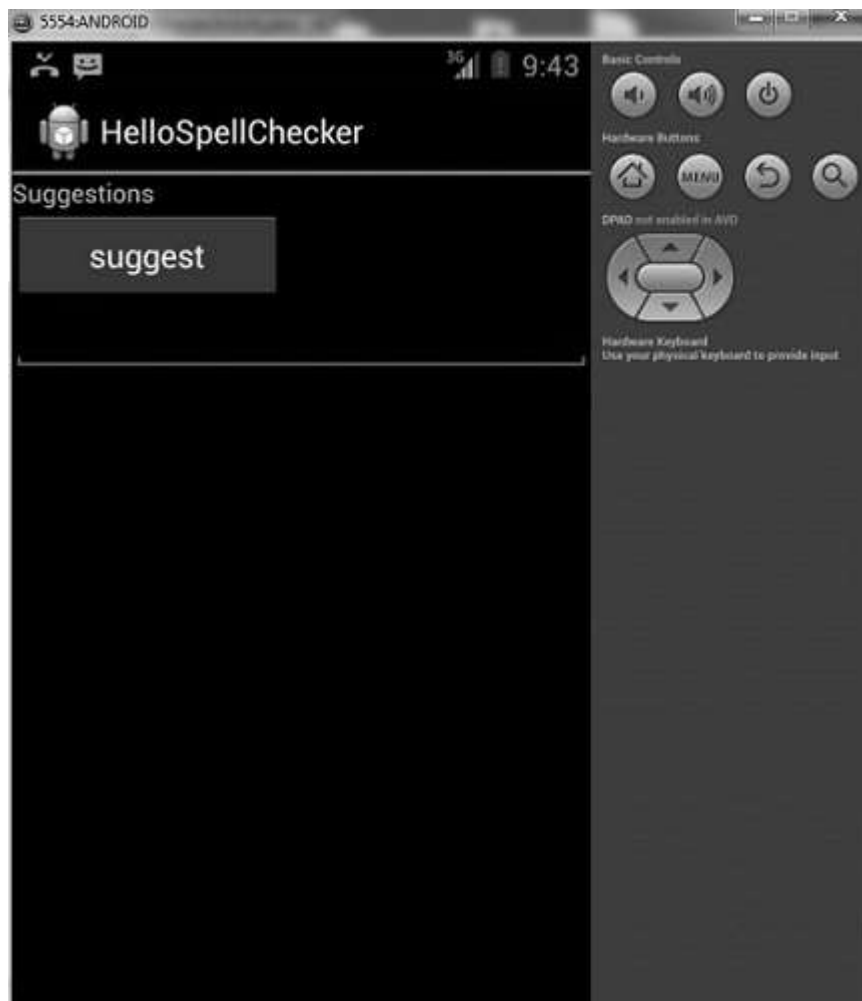
    <uses-sdk android:minSdkVersion="14" />

    <application
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".HelloSpellCheckerActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

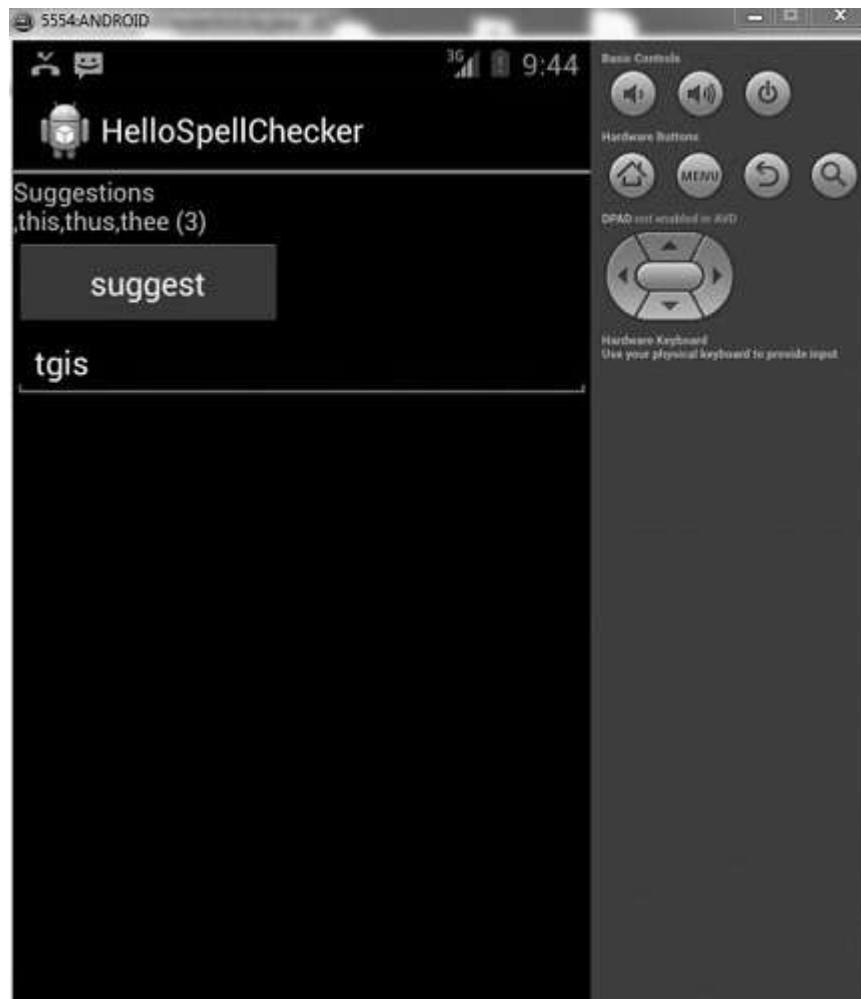
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

```
</manifest>
```

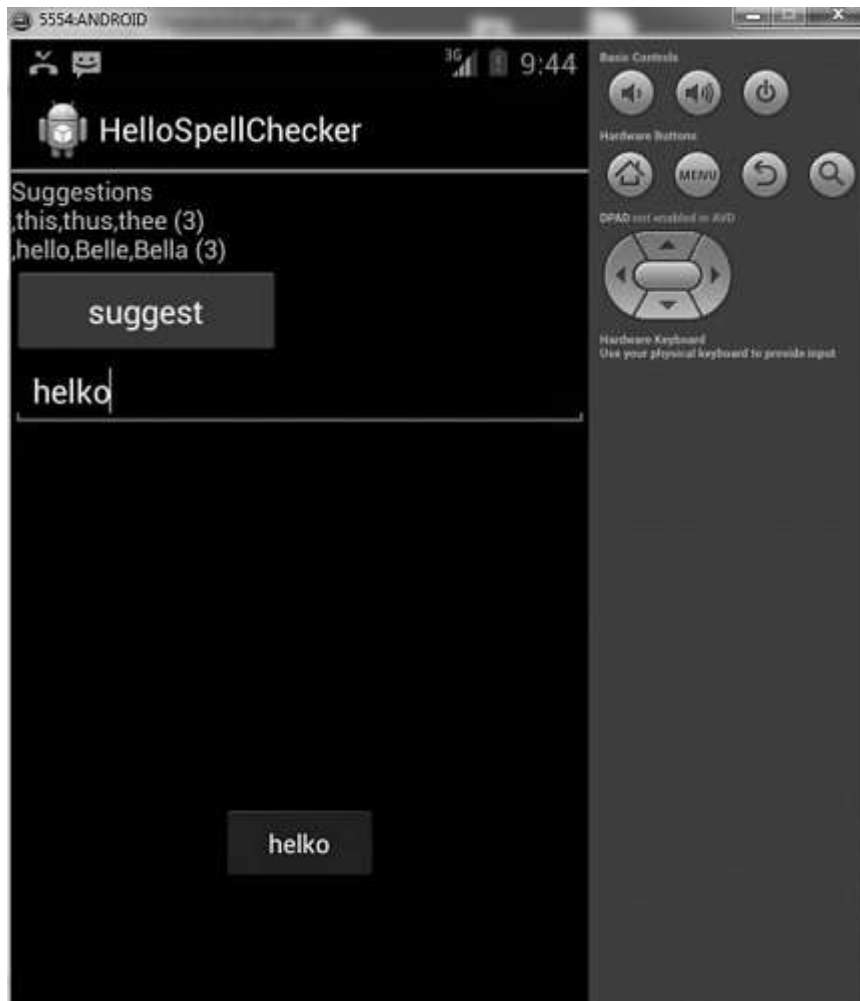
Let's try to run our Spell Checker application we just modified. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if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



Now what you need to do is to enter any text in the field. For example, we have entered some text. Press the suggestions button. The following notification would appear in your AVD along with suggestions:



Now change the text and press the button again, like we did. And this is what comes on screen.



68. SQLITE DATABASE

SQLite is an open source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. To access this database, you don't need to establish any kind of connections for it like JDBC, ODBC etc.

Database - Package

The main package is `android.database.sqlite` that contains the classes to manage your own databases.

Database - Creation

In order to create a database you need to call the method `openOrCreateDatabase` with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object. Its syntax is given below:

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",  
MODE_PRIVATE, null);
```

Apart from this, there are other functions available in the database package that does this job. They are listed below

Sr.No	Method & Description
1	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler) This method only opens the existing database with the appropriate flag mode. The common flags mode could be <code>OPEN_READWRITE</code> <code>OPEN_READONLY</code> .
2	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags) It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases.
3	openOrCreateDatabase(String path,

	<p>SQLiteDatabase.CursorFactory factory)</p> <p>It not only opens, but creates the database if it does not exists. This method is equivalent to openDatabase method.</p>
4	<p>openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)</p> <p>This method is similar to above method but it takes the File object as a path rather than a string. It is equivalent to file.getPath()</p>

Database - Insertion

We can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below:

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialPoint(Username
VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialPoint
VALUES('admin','admin');");
```

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	<p>execSQL(String sql, Object[] bindArgs)</p> <p>This method not only insert data, but also used to update or modify already existing data in database using bind arguments.</p>

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called.rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatabase.rawQuery("Select * from
TutorialPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes-

Sr.No	Method & Description
1	getColumnCount() This method returns the total number of columns of the table.
2	getColumnIndex(String columnName) This method returns the index number of a column by specifying the name of the column.
3	getColumnName(int columnIndex) This method returns the name of the column by specifying the index of the column.
4	getColumnNames() This method returns the array of all the column names of the table.
5	getCount() This method returns the total number of rows in the cursor.
6	getPosition() This method returns the current position of the cursor in the table.
7	isClosed() This method returns true if the cursor is closed and returns false otherwise.

Database - Helper class

For managing all the operations related to the database, a helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and updation of the database. Its syntax is given below:

```
public class DBHelper extends SQLiteOpenHelper {
    public DBHelper(){
        super(context,DATABASE_NAME,null,1);
    }
}
```

```

    }
    public void onCreate(SQLiteDatabase db) {}
    public void onUpgrade(SQLiteDatabase database, int oldVersion, int
newVersion) {}
}

```

Example:

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion, deletion and modification of contacts.

To experiment with this example, you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as AddressBook under a package com.example.addressbook. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work.
4	Create a new Activity as DisplayContact.java that will display the contact on the screen.
5	Modify the res/layout/activity_main to add respective XML components.
6	Modify the res/layout/activity_display_contact.xml to add respective XML components.
7	Modify the res/values/string.xml to add necessary string components.
8	Modify the res/menu/display_contact.xml to add necessary menu components.
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option.

- | | |
|----|---|
| 10 | Run the application and choose a running android device and install the application on it and verify the results. |
|----|---|

Following is the content of the modified main activity file **src/com.example.addressbook/MainActivity.java**.

```
package com.example.addressbook;

import java.util.ArrayList;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {
    public final static String EXTRA_MESSAGE =
        "com.example.AddressBook.MESSAGE";

    private ListView obj;
    DBHelper mydb;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mydb = new DBHelper(this);
```

```
ArrayList array_list = mydb.getAllCotacts();

ArrayAdapter arrayAdapter =
new ArrayAdapter(this, android.R.layout.simple_list_item_1,
array_list);

//adding it to the list view.
obj = (ListView)findViewById(R.id.listView1);
obj.setAdapter(arrayAdapter);

obj.setOnItemClickListener(new OnItemClickListener(){

@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
long arg3) {
    // TODO Auto-generated method stub
    int id_To_Search = arg2 + 1;
    Bundle dataBundle = new Bundle();
    dataBundle.putInt("id", id_To_Search);
    Intent intent = new
    Intent(getApplicationContext(), com.example.addressbook.DisplayCo
ntact.class);
    intent.putExtras(dataBundle);
    startActivity(intent);
}
});

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.mainmenu, menu);
    return true;
}
```



```

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    super.onOptionsItemSelected(item);
    switch(item.getItemId())
    {
        case R.id.item1:
            Bundle dataBundle = new Bundle();
            dataBundle.putInt("id", 0);
            Intent intent = new
            Intent(getApplicationContext(),com.example.addressbook.Displa
            yContact.class);
            intent.putExtras(dataBundle);
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

public boolean onKeyDown(int keycode, KeyEvent event) {
    if (keycode == KeyEvent.KEYCODE_BACK) {
        moveTaskToBack(true);
    }
    return super.onKeyDown(keycode, event);
}
}
}

```

Following is the modified content of display contact activity **src/com.example.addressbook/DisplayContact.java**

```

package com.example.addressbook;

import android.os.Bundle;

```

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class DisplayContact extends Activity {

    int from_Where_I_Am_Coming = 0;
    private DBHelper mydb ;
    TextView name ;
    TextView phone;
    TextView email;
    TextView street;
    TextView place;
    int id_To_Update = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_contact);
        name = (TextView) findViewById(R.id.editTextName);
        phone = (TextView) findViewById(R.id.editTextPhone);
        email = (TextView) findViewById(R.id.editTextStreet);
        street = (TextView) findViewById(R.id.editTextEmail);
        place = (TextView) findViewById(R.id.editTextCity);
    }
}
```

```
mydb = new DBHelper(this);

Bundle extras = getIntent().getExtras();
if(extras !=null)
{
    int Value = extras.getInt("id");
    if(Value>0){
        //means this is the view part not the add contact part.
        Cursor rs = mydb.getData(Value);
        id_To_Update = Value;
        rs.moveToFirst();
        String nam =
            rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_NAME)
            );
        String phon =
            rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_PHONE
            ));
        String emai =
            rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_EMAIL
            ));
        String stree =
            rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_STREE
            T));
        String plac =
            rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_CITY)
            );
        if (!rs.isClosed())
        {
            rs.close();
        }
        Button b = (Button)findViewById(R.id.button1);
        b.setVisibility(View.INVISIBLE);

        name.setText((CharSequence)nam);
        name.setFocusable(false);
        name.setClickable(false);
    }
}
```

```
        phone.setText((CharSequence)phon);
        phone.setFocusable(false);
        phone.setClickable(false);

        email.setText((CharSequence)emai);
        email.setFocusable(false);
        email.setClickable(false);

        street.setText((CharSequence)stree);
        street.setFocusable(false);
        street.setClickable(false);

        place.setText((CharSequence)plac);
        place.setFocusable(false);
        place.setClickable(false);
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    Bundle extras = getIntent().getExtras();
    if(extras !=null)
    {
        int Value = extras.getInt("id");
        if(Value>0){
            getMenuInflater().inflate(R.menu.display_contact, menu);
        }
        else{
            getMenuInflater().inflate(R.menu.main, menu);
        }
    }
    return true;
}
```

```
}

public boolean onOptionsItemSelected(MenuItem item)
{
    super.onOptionsItemSelected(item);
    switch(item.getItemId())
    {
        case R.id.Edit_Contact:
            Button b = (Button)findViewById(R.id.button1);
            b.setVisibility(View.VISIBLE);
            name.setEnabled(true);
            name.setFocusableInTouchMode(true);
            name.setClickable(true);

            phone.setEnabled(true);
            phone.setFocusableInTouchMode(true);
            phone.setClickable(true);

            email.setEnabled(true);
            email.setFocusableInTouchMode(true);
            email.setClickable(true);

            street.setEnabled(true);
            street.setFocusableInTouchMode(true);
            street.setClickable(true);

            place.setEnabled(true);
            place.setFocusableInTouchMode(true);
            place.setClickable(true);

            return true;
        case R.id.Delete_Contact:

            AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

```

        builder.setMessage(R.string.deleteContact)
        .setPositiveButton(R.string.yes, new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                mydb.deleteContact(id_To_Update);
                Toast.makeText(getApplicationContext(), "Deleted
                Successfully", Toast.LENGTH_SHORT).show();
                Intent intent = new
                Intent(getApplicationContext(), com.example.addressbook.MainAc
                tivity.class);
                startActivity(intent);
            }
        })
        .setNegativeButton(R.string.no, new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                // User cancelled the dialog
            }
        });
        AlertDialog d = builder.create();
        d.setTitle("Are you sure");
        d.show();

        return true;
        default:
        return super.onOptionsItemSelected(item);
    }
}

public void run(View view)
{
    Bundle extras = getIntent().getExtras();
    if(extras !=null)
    {

```

```

int Value = extras.getInt("id");
if(Value>0){

    if(mydb.updateContact(id_To_Update,name.getText().toString(),
    phone.getText().toString(), email.getText().toString(),
    street.getText().toString(), place.getText().toString())){

        Toast.makeText(getApplicationContext(), "Updated",
        Toast.LENGTH_SHORT).show();

        Intent intent = new
        Intent(getApplicationContext(),com.example.addressbook.Mai
        nActivity.class);

        startActivity(intent);

    }
    else{

        Toast.makeText(getApplicationContext(), "not Updated",
        Toast.LENGTH_SHORT).show();

    }
}
else{

    if(mydb.insertContact(name.getText().toString(),
    phone.getText().toString(), email.getText().toString(),
    street.getText().toString(), place.getText().toString())){

        Toast.makeText(getApplicationContext(), "done",
        Toast.LENGTH_SHORT).show();

    }
    else{

        Toast.makeText(getApplicationContext(), "not done",
        Toast.LENGTH_SHORT).show();

    }

    Intent intent = new
    Intent(getApplicationContext(),com.example.addressbook.MainAc
    tivity.class);

    startActivity(intent);

}

}

}
}

```

Following is the content of Database class **src/com.example.addressbook/DBHelper.java**

```
package com.example.addressbook;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper {

    public static final String DATABASE_NAME = "MyDBName.db";
    public static final String CONTACTS_TABLE_NAME = "contacts";
    public static final String CONTACTS_COLUMN_ID = "id";
    public static final String CONTACTS_COLUMN_NAME = "name";
    public static final String CONTACTS_COLUMN_EMAIL = "email";
    public static final String CONTACTS_COLUMN_STREET = "street";
    public static final String CONTACTS_COLUMN_CITY = "place";
    public static final String CONTACTS_COLUMN_PHONE = "phone";

    private HashMap hp;

    public DBHelper(Context context)
    {
        super(context, DATABASE_NAME, null, 1);
    }
}
```



```

@Override
public void onCreate(SQLiteDatabase db) {
    // TODO Auto-generated method stub
    db.execSQL(
        "create table contacts " +
        "(id integer primary key, name text,phone text,email text, street
        text,place text)"
    );
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    // TODO Auto-generated method stub
    db.execSQL("DROP TABLE IF EXISTS contacts");
    onCreate(db);
}

public boolean insertContact (String name, String phone, String
email, String street,String place)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();

    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);

    db.insert("contacts", null, contentValues);
    return true;
}

public Cursor getData(int id){
    SQLiteDatabase db = this.getReadableDatabase();

```

```
        Cursor res = db.rawQuery( "select * from contacts where
        id="+id+"", null );
        return res;
    }
    public int numberOfRows(){
        SQLiteDatabase db = this.getReadableDatabase();
        int numRows = (int) DatabaseUtils.queryNumEntries(db,
        CONTACTS_TABLE_NAME);
        return numRows;
    }
    public boolean updateContact (Integer id, String name, String phone,
    String email, String street,String place)
    {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", name);
        contentValues.put("phone", phone);
        contentValues.put("email", email);
        contentValues.put("street", street);
        contentValues.put("place", place);
        db.update("contacts", contentValues, "id = ? ", new String[] {
        Integer.toString(id) } );
        return true;
    }

    public Integer deleteContact (Integer id)
    {
        SQLiteDatabase db = this.getWritableDatabase();
        return db.delete("contacts",
        "id = ? ",
        new String[] { Integer.toString(id) });
    }
    public ArrayList getAllCotacts()
    {
```

```

ArrayList array_list = new ArrayList();
//hp = new HashMap();
SQLiteDatabase db = this.getReadableDatabase();
Cursor res = db.rawQuery( "select * from contacts", null );
res.moveToFirst();
while(res.isAfterLast() == false){

    array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN_NAME)));
    res.moveToNext();
}
return array_list;
}
}

```

Following is the content of the **res/layout/activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" >
    </ListView>

```

```
</RelativeLayout>
```

Following is the content of the **res/layout/activity_display_contact.xml**

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/scrollView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:context=".DisplayContact" >

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="370dp"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    >

    <EditText
        android:id="@+id/editTextName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_marginTop="5dp"
        android:layout_marginLeft="82dp"
        android:ems="10"
        android:inputType="text" >

</EditText>

    <EditText
        android:id="@+id/editTextEmail"
```

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_alignLeft="@+id/editTextStreet"  
android:layout_below="@+id/editTextStreet"  
android:layout_marginTop="22dp"  
android:ems="10"  
android:inputType="textEmailAddress" />
```

```
<TextView
```

```
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editTextName"  
    android:layout_alignParentLeft="true"  
    android:text="@string/name"  
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<Button
```

```
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/editTextCity"  
    android:layout_alignParentBottom="true"  
    android:layout_marginBottom="28dp"  
    android:onClick="run"  
    android:text="@string/save" />
```

```
<TextView
```

```
    android:id="@+id/textView2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editTextEmail"  
    android:layout_alignLeft="@+id/textView1"  
    android:text="@string/email"
```

```
        android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextPhone"
    android:layout_alignLeft="@+id/textView1"
    android:text="@string/phone"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/editTextEmail"
    android:layout_alignLeft="@+id/textView5"
    android:text="@string/street"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<EditText
    android:id="@+id/editTextCity"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/editTextName"
    android:layout_below="@+id/editTextEmail"
    android:layout_marginTop="30dp"
    android:ems="10"
    android:inputType="text" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

        android:layout_alignBaseline="@+id/editTextCity"
        android:layout_alignBottom="@+id/editTextCity"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/editTextEmail"
        android:text="@string/country"
        android:textAppearance="?android:attr/textAppearanceMedium" />

<EditText
    android:id="@+id/editTextStreet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextName"
    android:layout_below="@+id/editTextPhone"
    android:ems="10"
    android:inputType="text" >

    <requestFocus />
</EditText>

<EditText
    android:id="@+id/editTextPhone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextStreet"
    android:layout_below="@+id/editTextName"
    android:ems="10"
    android:inputType="phone|text" />

</RelativeLayout>
</ScrollView>

```

Following is the content of the **res/value/string.xml**

```

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

```

```

<string name="app_name">Address Book</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="Add_New">Add New</string>
<string name="edit">Edit Contact</string>
<string name="delete">Delete Contact</string>
<string name="title_activity_display_contact">DisplayContact</string>

<string name="name">Name</string>
<string name="phone">Phone</string>
<string name="email">Email</string>
<string name="street">Street</string>
<string name="country">City/State/Zip</string>

<string name="save">Save Contact</string>

<string name="deleteContact">Are you sure, you want to delete
it.</string>
<string name="yes">Yes</string>
<string name="no">No</string>

</resources>

```

Following is the content of the **res/menu/mainmenu.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1"
        android:icon="@drawable/add"
        android:title="@string/Add_New"
        android:showAsAction="ifRoom|withText">
    </item>
</menu>

```

Following is the content of the **res/menu/display_contact.xml**


```

<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item
    android:id="@+id/Edit_Contact"
    android:orderInCategory="100"
    android:title="@string/edit"/>
  <item
    android:id="@+id/Delete_Contact"
    android:orderInCategory="100"
    android:title="@string/delete"/>

</menu>

```

This is the default **AndroidManifest.xml** of this project

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.addressbook"
  android:versionCode="1"
  android:versionName="1.0" >


  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

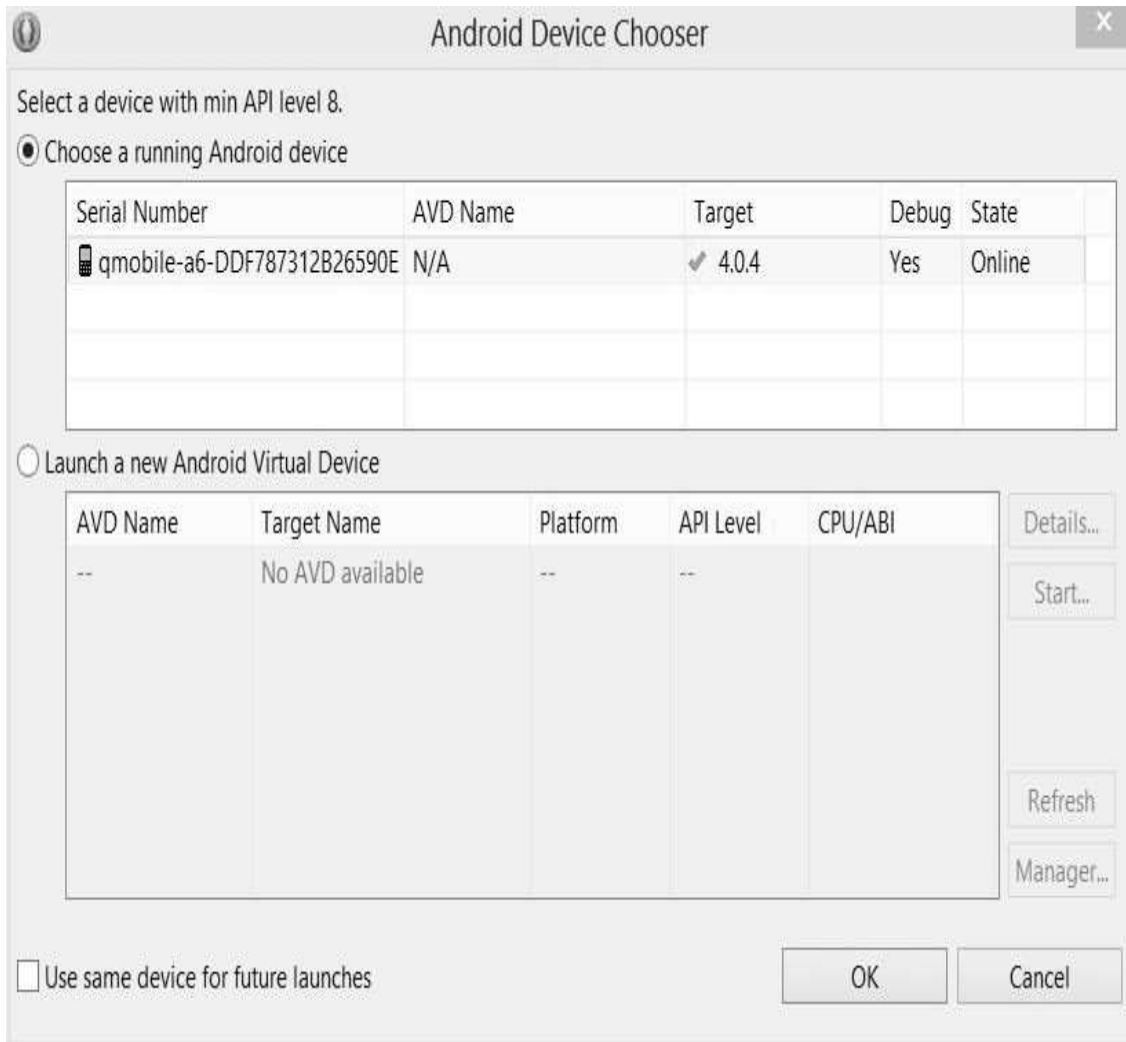
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.addressbook.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

```

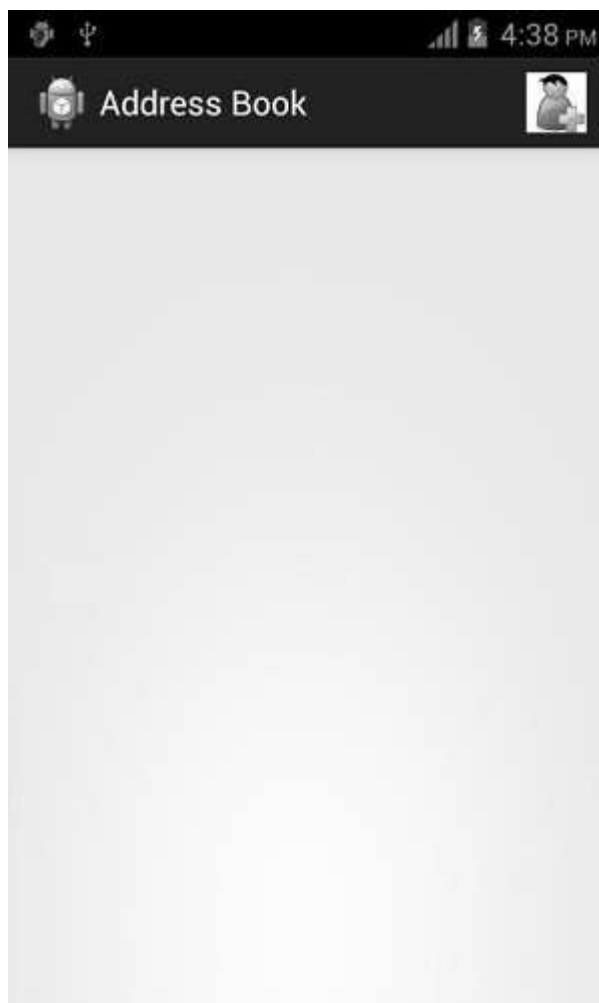
```
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name="com.example.addressbook.DisplayContact"
    android:label="@string/title_activity_display_contact" >
    </activity>
</application>

</manifest>
```

Let's try to run your Camera application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Click on the add button on the top right corner of the menu screen to add a new contact. It will display the following screen:

The screenshot shows an Android application interface with a black header bar containing the text 'DisplayContact' and an Android robot icon. Below the header, there is a form with the following fields and values:

Name	john
Phone	52668
Street	xyz
Email	john@gmail.com
City/ State/Zip	US

At the bottom of the form, there is a grey button labeled 'Save Contact'.

It will display the following fields. Please enter the required information and click on save contact. It will bring you back to main screen.

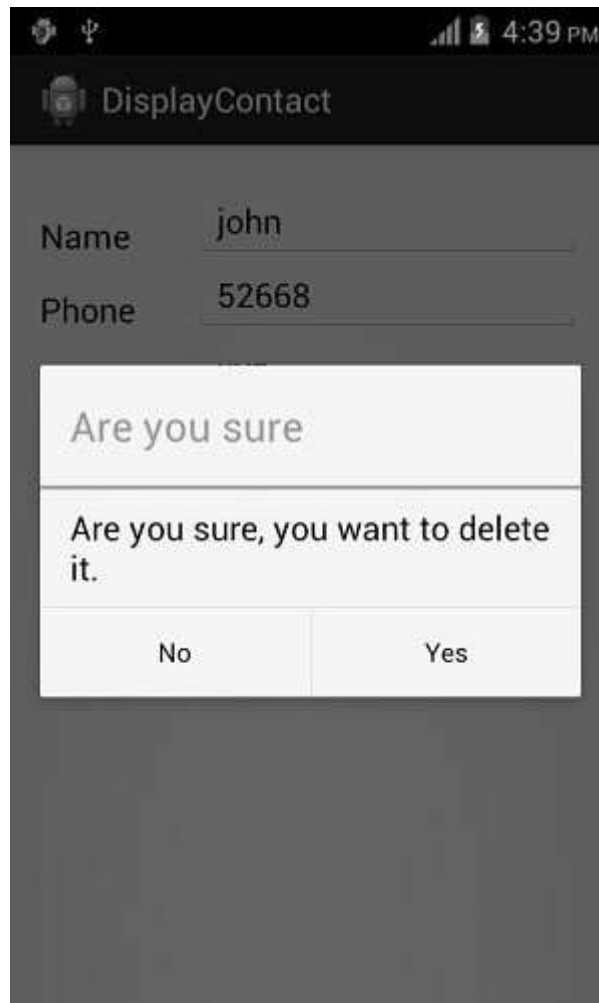


Now our contact john has been added. Tap on this to edit or delete the contact. It will bring you to the following screen. Now select menu from your mobile. And there will be two options there.

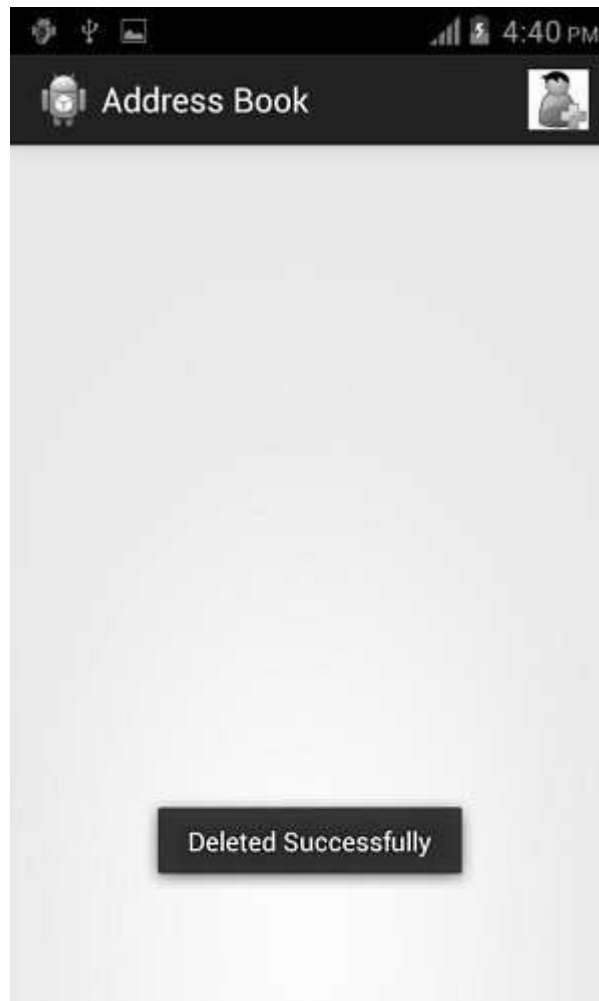


The screenshot shows an Android application interface with a dark header bar containing the text "DisplayContact" and an Android robot icon. Below the header, there are five text input fields with labels on the left and values on the right: "Name" with "john", "Phone" with "52668", "Street" with "xyz", "Email" with "john@gmail.com", and "City/State/Zip" with "US". At the bottom of the screen, a light gray menu is open, displaying two options: "Edit Contact" and "Delete Contact". The top status bar shows icons for USB, Wi-Fi, and battery, along with the time "4:37 PM".

Select delete contact and a dialog box would appear asking you about deleting this contact. It would be like this -



Select Yes from the above screen that appears and a notification will appear that the contact has been deleted successfully. It would appear like this -



In order to see where your database is created, open your eclipse, connect your mobile, go to right corner and select DDMS. Now browse the file explorer tab. Now browse this folder **/data/data/<your.package.name>/databases<database-name>**.

69. SUPPORT LIBRARY

When you develop an app on a latest version of android like 4.0 and you also want it to run on those devices which are running older versions of android like 3.2 etc. you can't do that until you add backward compatability to your code.

To provide this backward compatability android provides you the **Android Support Library** package. The Android Support Library package is a set of code libraries that provide backward-compatible versions of Android framework APIs as well as features that are only available through the library APIs. Each Support Library is backward-compatible to a specific Android API level.

Including the Support Libraries in your Android project is considered a best practice for application developers, depending on the range of platform versions your app is targeting and the APIs that it uses.

Support Library Features

The Android Support Library package contains several libraries that can be included in your application. Each of these libraries supports a specific range of Android platform versions and set of features.

In order to effectively use the libraries, it is important to consider the API level that you want to target, as each library supports different API level.

Following is a brief description of android support libraries and API level they support.

Sr.No	Version & Features
1	v4 Support Library This library is designed to be used with Android 1.6 (API level 4) and higher.
2	v7 Support Library There are several libraries designed to be used with Android 2.1 (API level 7) and higher.
3	v8 Support Library This library is designed to be used with Android (API level 8) and higher.

4

v13 Support Library

This library is designed to be used for Android 3.2 (API level 13) and higher.

Please remember that use of Android Support Library in your app code is encouraged and preferred. By using these libraries you can increase your target market and target audience.

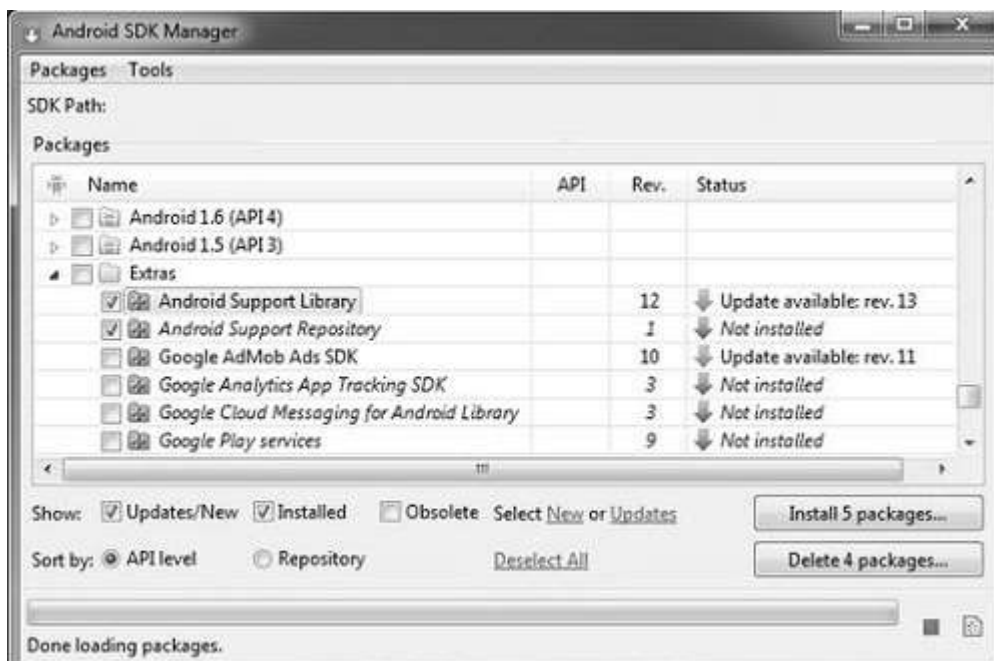
Downloading the Support Libraries

Note: Before installing the support library packages you should be clear of the feature you want to use in your app.

The Android Support Library package is available through the Android SDK Manager.

To download the support library package through the SDK Manager, follow these steps:

- Start the android SDK Manager.
- In the SDK Manager window, scroll to the end of the Packages list, find the Extras folder.
- Select the Android Support Library item.
- Click the **Install packages** button.



After downloading, the tool installs the Support Library files to your existing Android SDK directory. The library files are located in the following sub-directory of your SDK:/extras/android/support/ directory.

Choosing Support Libraries

Before adding a Support Library to your application, decide what features you want to include and the lowest Android versions you want to support.

Changes in Android.Manifest

If you are increasing the backward compatibility of your existing application to an earlier version of the Android API with the Support Library, make sure to update your application's manifest. Specifically, you should update the **android:minSdkVersion** element of the tag in the manifest to the new, lower version number, as shown below:

```
<uses-sdk
    android:minSdkVersion="7"
    android:targetSdkVersion="17" />
```

This change tells Google Playstore app that your application can be installed on devices with Android 2.1 (API level 7) and higher.

API Version

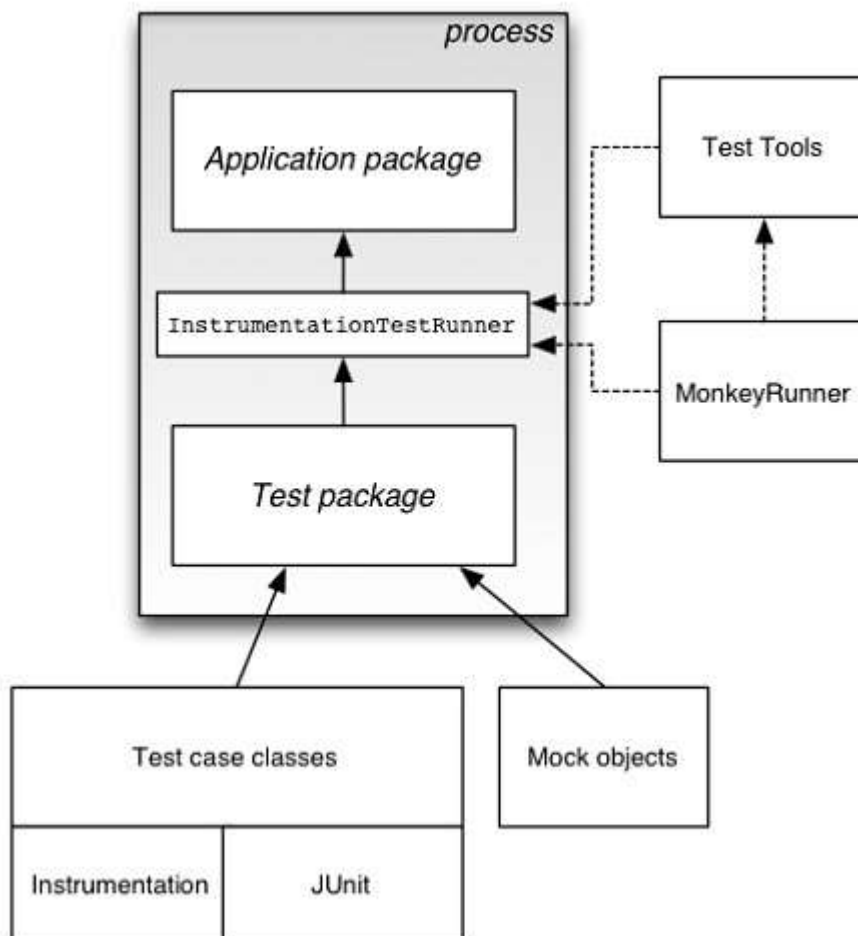
Note: If you are including the v4 support and v7 support libraries in your application, you should specify a minimum SDK version of "7" (and not "4"). The highest support library level you include in your application determines the lowest API version in which it can operate.

70. TESTING

The Android framework includes an integrated testing framework that help you test all aspects of your application. The SDK tools include tools for setting up and running test applications. Whether you are working in Eclipse with ADT or working from the command line, the SDK tools help you set up and run your tests within an emulator or the device you are targeting.

Test Structure

Android's build and test tools assume that test projects are organized into a standard structure of tests, test case classes, test packages, and test projects.



Testing Tools in Android

There are many tools that can be used for testing android applications. Some are official like Junit, Monkey and some are third party tools that can be used to

test android applications. In this chapter we are going to explain these two tools to test android applications.

- JUnit
- Monkey

JUnit

You can use the JUnit **TestCase** class to do unit testing on a class that doesn't call Android APIs. TestCase is also the base class for AndroidTestCase, which you can use to test Android-dependent objects. Besides providing the JUnit framework, AndroidTestCase offers Android-specific setup, teardown, and helper methods.

To use TestCase, extend your class with TestCase class and implement a method call setUp(). Its syntax is given below:

```
public class MathTest extends TestCase {
    protected double fValue1;
    protected double fValue2;

    protected void setUp() {
        fValue1= 2.0;
        fValue2= 3.0;
    }
}
```

For each test, implement a method which interacts with the fixture. Verify the expected results with assertions specified by calling assertTrue(String, boolean) with a boolean.

```
public void testAdd() {
    double result= fValue1 + fValue2;
    assertTrue(result == 5.0);
}
```

The assert methods compare values you expect from a test to the actual results and throw an exception if the comparison fails.

Once the methods are defined you can run them. Its syntax is given below:

```
TestCase test= new MathTest("testAdd");
test.run();
```

Monkey

The UI/Application Exerciser Monkey, usually called "monkey", is a command-line tool that sends pseudo-random streams of keystrokes, touches, and gestures to a device. You run it with the Android Debug Bridge (adb) tool.

You use it to stress-test your application and report back errors that are encountered. You can repeat a stream of events by running the tool each time with the same random number seed.

Monkey features

Monkey has many features, but it can all be summed up to these four categories.

- Basic configuration options
- Operational constraints
- Event types and frequencies
- Debugging options

Monkey Usage

In order to use monkey, open up a command prompt and just navigate to the following directory.

```
android->sdk->platform-tools
```

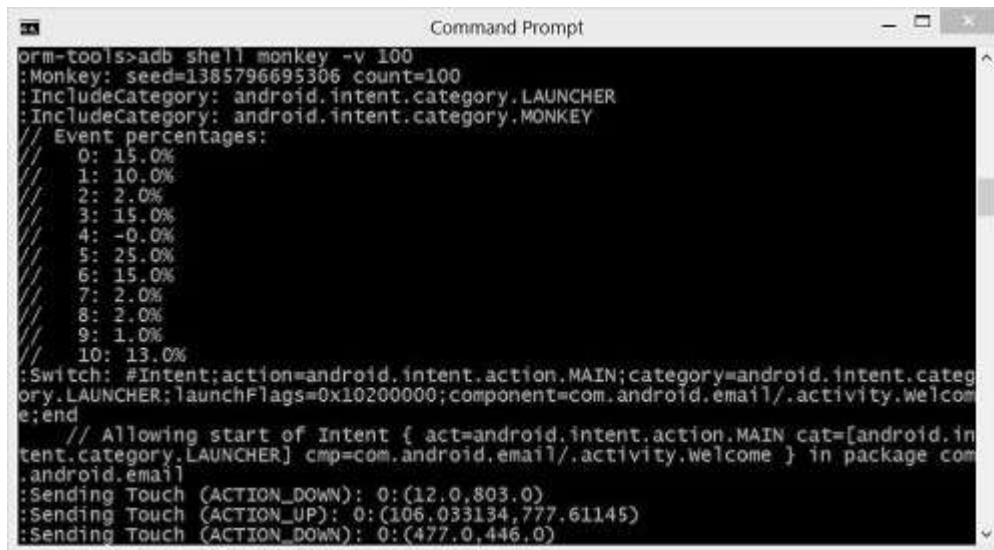
Once inside the directory, attach your device with the PC, and run the following command:

```
adb shell monkey -v 100
```

This command can be broken down into these steps:

- adb - Android Debug Bridge. A tool used to connect and send commands to your Android phone from a desktop or laptop computer.
- shell - shell is just an interface on the device that translates our commands to system commands.
- monkey - monkey is the testing tool.
- v - v stands for verbose method.
- 100- it is the frequency count or the number of events to be sent for testing.

This is also shown in the figure:



```

orm-tools>adb shell monkey -v 100
:Monkey: seed=1385796695306 count=100
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
Event percentages:
0: 15.0%
1: 10.0%
2: 2.0%
3: 15.0%
4: -0.0%
5: 25.0%
6: 15.0%
7: 2.0%
8: 2.0%
9: 1.0%
10: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;component=com.android.email/.activity.welcome;end
// Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.android.email/.activity.welcome } in package com.android.email
:Sending Touch (ACTION_DOWN): 0:(12.0,803.0)
:Sending Touch (ACTION_UP): 0:(106.033134,777.61145)
:Sending Touch (ACTION_DOWN): 0:(477.0,446.0)

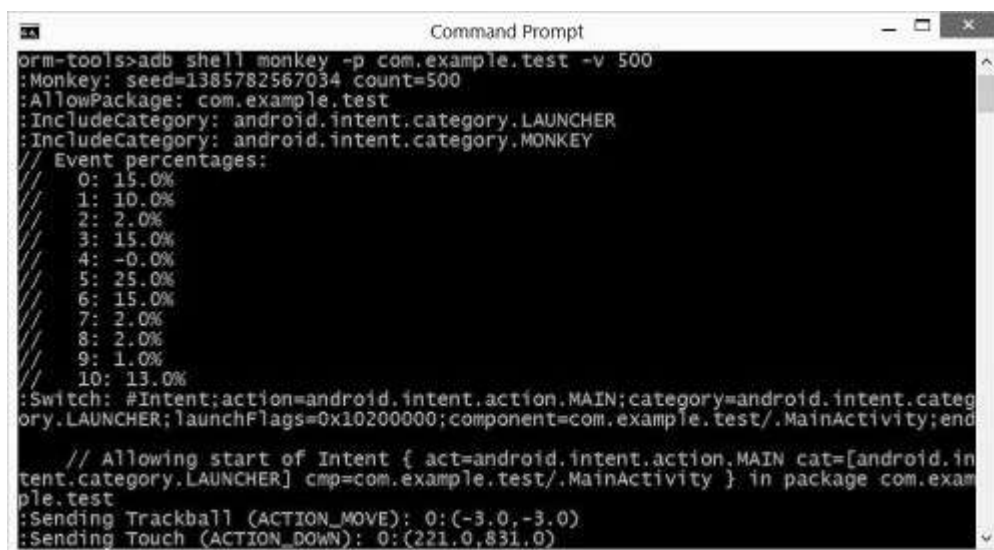
```

Here, you run the monkey tool on the default android UI application. Now in order to run it to your application, here is what you have to do.

First run the example code given in the example section in your device. After running, follow the steps of monkey usage and finally type this command.

```
adb shell monkey -p com.example.test -v 500
```

This has also been shown in the figure below. By typing this command, you are actually generating 500 random events for testing.



```

orm-tools>adb shell monkey -p com.example.test -v 500
:Monkey: seed=1385782567034 count=500
:AllowPackage: com.example.test
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
Event percentages:
0: 15.0%
1: 10.0%
2: 2.0%
3: 15.0%
4: -0.0%
5: 25.0%
6: 15.0%
7: 2.0%
8: 2.0%
9: 1.0%
10: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;component=com.example.test/.MainActivity;end
// Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.example.test/.MainActivity } in package com.example.test
:Sending Trackball (ACTION_MOVE): 0:(-3.0,-3.0)
:Sending Touch (ACTION_DOWN): 0:(221.0,831.0)

```

Example

The below example demonstrates the use of Testing. It creates a basic application which can be used for monkey.

To experiment with this example, you need to run this on an actual device and then follow the monkey steps explained in the beginning.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Test under a package com.example.test. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Activity code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Create src/MainActivity2.java file to add Activity code.
5	Modify layout XML file res/layout/activity_main_activity2.xml add any GUI component if required.
6	Modify res/values/string.xml file and add necessary string components.
7	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.test/MainActivity.java**.

```

package com.example.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

```

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

    public void activity2(View view){
        Intent intent = new
            Intent(this,com.example.test.MainActivity2.class);
        startActivity(intent);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

Here is the content of **src/com.example.test/MainActivity2.java**.

```
package com.example.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;

public class MainActivity2 extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_activity2);
    }
}
```

```

public void activity1(View view){
    Intent intent = new
        Intent(this,com.example.test.MainActivity.class);
    startActivity(intent);
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main_activity2, menu);
    return true;
}
}

```

Here is the content of **activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="87dp"
        android:text="@string/test1"
    >

```

```

        android:textAppearance="?android:attr/textAppearanceLarge" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:onClick="activity2"
    android:text="@string/go2" />
</RelativeLayout>

```

Here is the content of **activity_main_activity2.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity2" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="125dp"
        android:text="@string/test2"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button

```

```

    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:onClick="activity1"
    android:text="@string/go1" />

```

```
</RelativeLayout>
```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">test</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="test1">This is activity 1</string>
    <string name="test2">This is activity 2</string>
    <string name="go1">Go to activity 1</string>
    <string name="go2">Go to activity 2</string>
    <string name="title_activity_main_activity2">MainActivity2</string>

</resources>

```

Here is the content of **AndroidManifest.xml**.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.test"
    android:versionCode="1"
    android:versionName="1.0" >


    <uses-sdk

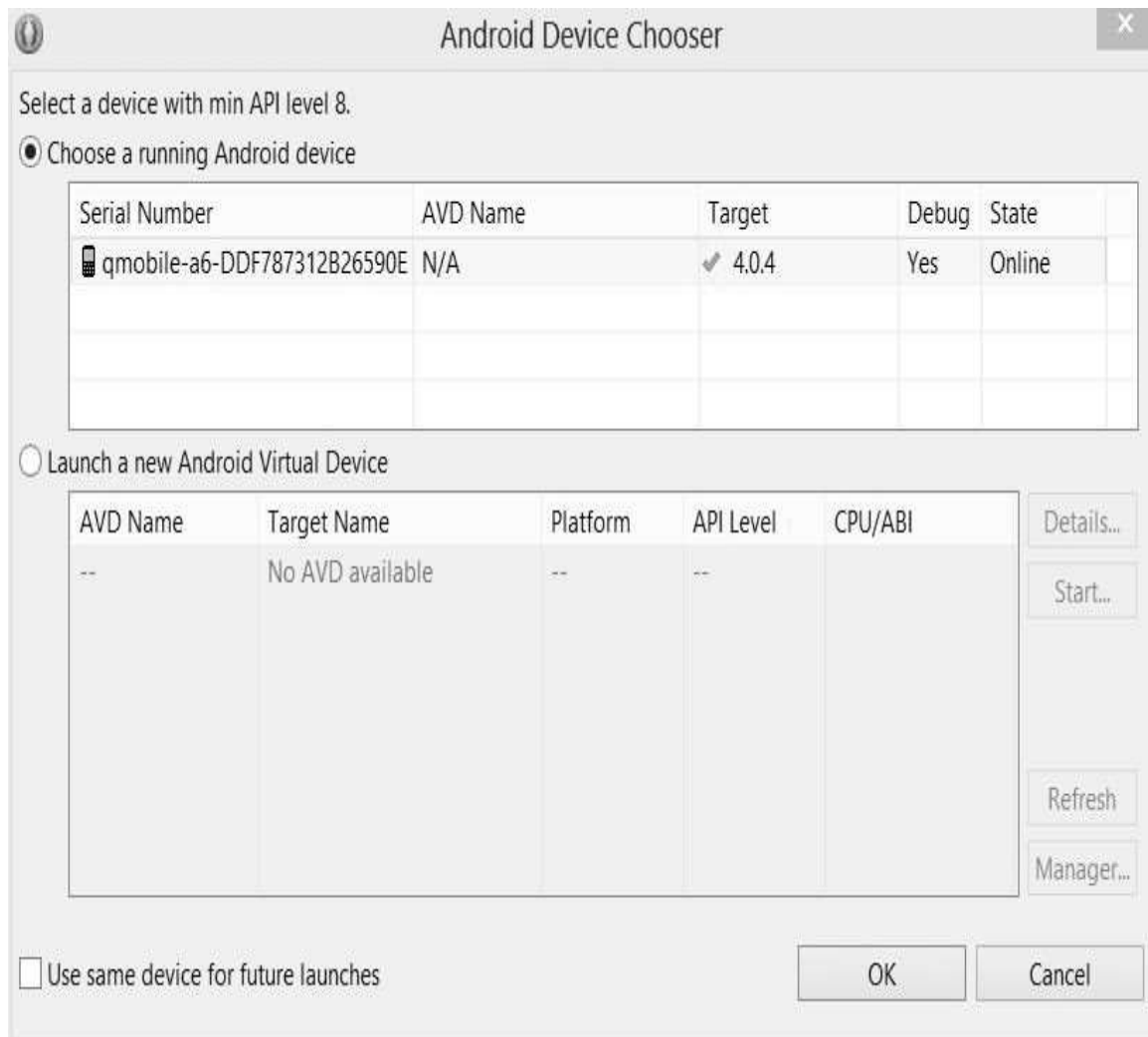
```

```
    android:minSdkVersion="8"
    android:targetSdkVersion="14" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.test.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="com.example.test.MainActivity2"
        android:label="@string/title_activity_main_activity2" >
    </activity>
</application>
</manifest>
```

Let's try to run your Android Testing application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display application screen. Now just follow the steps mentioned at the top under the monkey section in order to perform testing on this application.

71. TEXT TO SPEECH

Android allows you to convert your text into voice. Not only you can convert it, but it also allows you to speak text in variety of different languages.

Android provides **TextToSpeech** class for this purpose. To use this class, you need to instantiate an object of this class and also specify the **initListener**. Its syntax is given below:

```
private EditText write;
ttobj=new TextToSpeech(getApplicationContext(), new
TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
    }
}
);
```

In this listener, you have to specify the properties for TextToSpeech object, such as its language, pitch etc. Language can be set by calling **setLanguage()** method. Its syntax is given below:

```
ttobj.setLanguage(Locale.UK);
```

The method setLanguage takes a Locale object as parameter. The list of some of the locales available are given below:

Sr.No	Locale
1	US
2	CANADA_FRENCH
3	GERMANY
4	ITALY
5	JAPAN

6	CHINA
---	--------------

Once you have set the language, you can call **speak** method of the class to speak the text. Its syntax is given below:

```
ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

Apart from the speak method, there are some other methods available in the TextToSpeech class. They are listed below:

Sr.No	Method & description
1	<p>addSpeech(String text, String filename)</p> <p>This method adds a mapping between a string of text and a sound file.</p>
2	<p>getLanguage()</p> <p>This method returns a Locale instance describing the language.</p>
3	<p>isSpeaking()</p> <p>This method checks whether the TextToSpeech engine is busy speaking.</p>
4	<p>setPitch(float pitch)</p> <p>This method sets the speech pitch for the TextToSpeech engine.</p>
5	<p>setSpeechRate(float speechRate)</p> <p>This method sets the speech rate.</p>
6	<p>shutdown()</p> <p>This method releases the resources used by the TextToSpeech engine.</p>
7	<p>stop()</p> <p>This method stops the speak.</p>

Example

The below example demonstrates the use of TextToSpeech class. It creates a basic application that allows you to set write text and speak it.

To experiment with this example, you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as TextToSpeech under a package com.example.texttospeech. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add TextToSpeech code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.texttospeech/MainActivity.java**.

```
package com.example.texttospeech;

import java.util.Locale;
import java.util.Random;

import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    TextToSpeech ttoobj;
```

```
private EditText write;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    write = (EditText)findViewById(R.id.editText1);
    ttobj=new TextToSpeech(getApplicationContext(),
    new TextToSpeech.OnInitListener() {
        @Override
        public void onInit(int status) {
            if(status != TextToSpeech.ERROR){
                ttobj.setLanguage(Locale.UK);
            }
        }
    });
}
@Override
public void onPause(){
    if(ttobj !=null){
        ttobj.stop();
        ttobj.shutdown();
    }
    super.onPause();
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
public void speakText(View view){
    String toSpeak = write.getText().toString();
    Toast.makeText(getApplicationContext(), toSpeak,
```

```

        Toast.LENGTH_SHORT).show();
        ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);

    }
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_marginBottom="188dp"
        android:layout_marginRight="67dp"
        android:onClick="speakText"
        android:text="@string/text1" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/button1"
        android:layout_centerHorizontal="true"

```

```
        android:layout_marginBottom="81dp"
        android:ems="10" >
        <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="@string/write"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

Here is the content of **Strings.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TextToSpeech</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="text1">Text to Speech</string>
    <string name="write">Write Text</string>
</resources>
```


Here is the content of **AndroidManifest.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.texttospeech"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

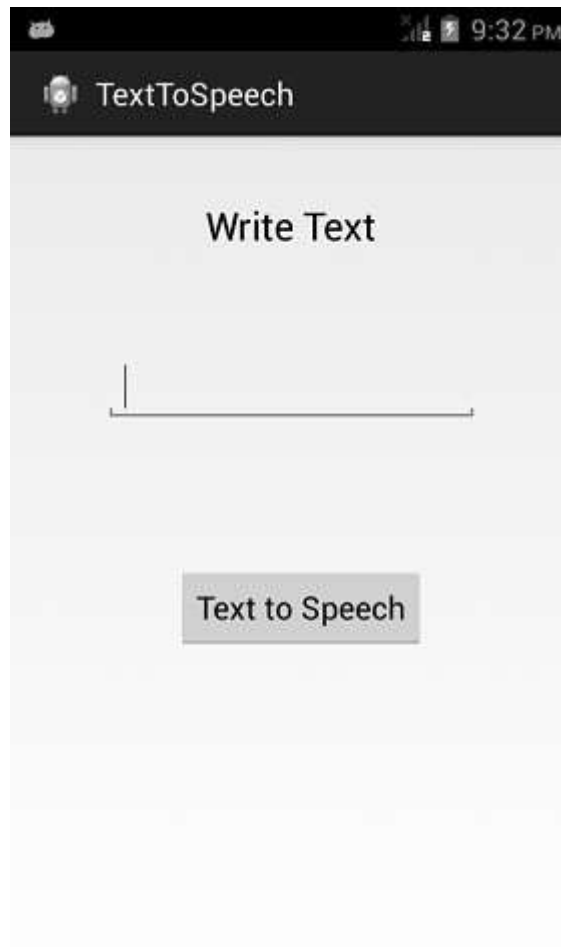
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.texttospeech.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Let's try to run your TextToSpeech application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display the following screen:



Now just type some text in the field and click on the text to speech button below. A notification would appear and text will be spoken. It is shown in the image below:



Now type something else and repeat the step again with different locale. You will again hear sound. This is shown below:



72. TEXTURE VIEW

If you want to display a live video stream or any content stream such as video or an OpenGL scene, you can use TextureView provided by android in order to do that.

In order to use TextureView, all you need to do is get its SurfaceTexture. The SurfaceTexture can then be used to render content. To do this, you just need to do instantiate an object of this class and implement SurfaceTextureListener interface. Its syntax is given below:

```
private TextureView myTexture;
public class MainActivity extends Activity implements
SurfaceTextureListener{
protected void onCreate(Bundle savedInstanceState) {
    myTexture = new TextureView(this);
    myTexture.setSurfaceTextureListener(this);
    setContentView(myTexture);
}
}
```

After that, what you need to do is to override its methods. The methods are listed as follows:

```
@Override
public void onSurfaceTextureAvailable(SurfaceTexture arg0, int arg1, int
arg2) {
}
@Override
public boolean onSurfaceTextureDestroyed(SurfaceTexture arg0) {
}
@Override
public void onSurfaceTextureSizeChanged(SurfaceTexture arg0, int arg1,int
arg2) {
}
@Override
public void onSurfaceTextureUpdated(SurfaceTexture arg0) {
}
```

Any view that is displayed in the texture view can be rotated and its alpha property can be adjusted by using **setAlpha** and **setRotation** methods. Its syntax is given below:

```
myTexture.setAlpha(1.0f);
myTexture.setRotation(90.0f);
```

Apart from these methods, there are other methods available in TextureView class. They are listed below:

Sr.No	Method & description
1	<p>getSurfaceTexture()</p> <p>This method returns the SurfaceTexture used by this view.</p>
2	<p>getBitmap(int width, int height)</p> <p>This method returns a Bitmap representation of the content of the associated surface texture.</p>
3	<p>getTransform(Matrix transform)</p> <p>This method returns the transform associated with this texture view.</p>
4	<p>isOpaque()</p> <p>This method indicates whether this View is opaque.</p>
5	<p>lockCanvas()</p> <p>This method starts editing the pixels in the surface.</p>
6	<p>setOpaque(boolean opaque)</p> <p>This method indicates whether the content of this TextureView is opaque.</p>
7	<p>setTransform(Matrix transform)</p> <p>This method sets the transform to associate with this texture view.</p>
8	<p>unlockCanvasAndPost(Canvas canvas)</p> <p>This method finishes editing pixels in the surface.</p>

Example

The below example demonstrates the use of TextureView class. It creates a basic application that allows you to view camera inside a texture view and change its angle, orientation etc.

To experiment with this example, you need to run this on an actual device on which camera is present.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as TextureView under a package com.example.textureview. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Activity code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
5	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.textureview/MainActivity.java**.

```
package com.example.textureview;

import java.io.IOException;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.graphics.SurfaceTexture;
import android.hardware.Camera;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.TextureView;
import android.view.TextureView.SurfaceTextureListener;
import android.view.View;
```

```
import android.widget.FrameLayout;

public class MainActivity extends Activity implements
SurfaceTextureListener {

    private TextureView myTexture;
    private Camera mCamera;

    @SuppressWarnings("NewApi")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myTexture = new TextureView(this);
        myTexture.setSurfaceTextureListener(this);
        setContentView(myTexture);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @SuppressWarnings("NewApi")
    @Override
    public void onSurfaceTextureAvailable(SurfaceTexture arg0, int arg1,
int arg2) {
        mCamera = Camera.open();
        Camera.Size previewSize = mCamera.getParameters().getPreviewSize();
        myTexture.setLayoutParams(new FrameLayout.LayoutParams(
        previewSize.width, previewSize.height, Gravity.CENTER));
        try {
            mCamera.setPreviewTexture(arg0);
```

```

        } catch (IOException t) {
        }

        mCamera.startPreview();
        myTexture.setAlpha(1.0f);
        myTexture.setRotation(90.0f);
    }

    @Override
    public boolean onSurfaceTextureDestroyed(SurfaceTexture arg0) {
        mCamera.stopPreview();
        mCamera.release();
        return true;
    }

    @Override
    public void onSurfaceTextureSizeChanged(SurfaceTexture arg0, int arg1,
        int arg2) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onSurfaceTextureUpdated(SurfaceTexture arg0) {
        // TODO Auto-generated method stub
    }
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

```

```

tools:context=".MainActivity" >

<TextureView
    android:id="@+id/textureView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
</RelativeLayout>

```

Here is the default content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.texturereview"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.CAMERA"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.texturereview.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>


```

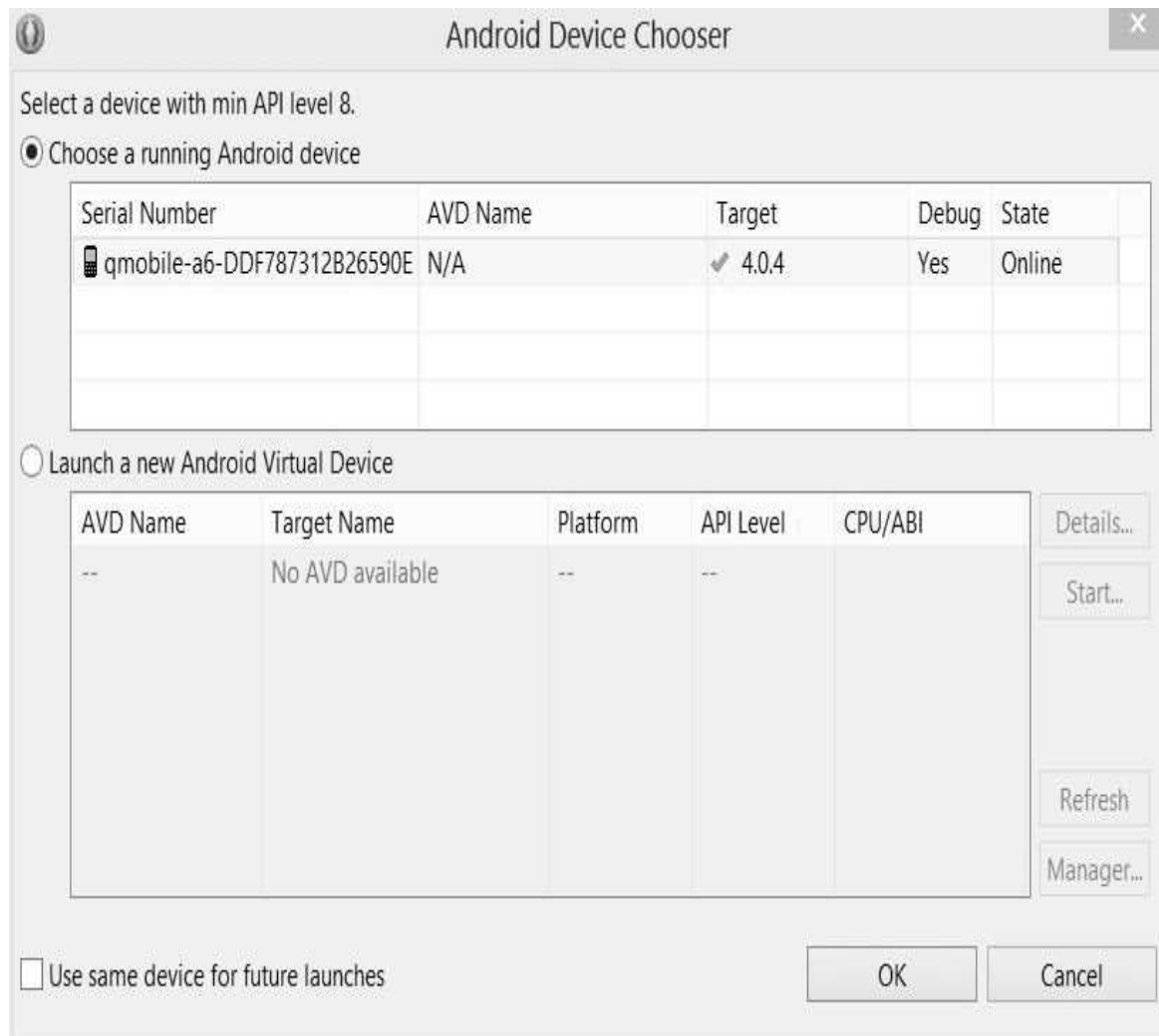


```

    </activity>
</application>
</manifest>

```

Let's try to run your TextureView application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen. This screen has alpha property set to **0.5** and rotation set to **45**.



This screen has alpha property set to **1.5** and rotation set to **45**.



This screen has alpha property set to **1.0** and rotation set to **90**.



73. TWITTER INTEGRATION

Android allows your application to connect to twitter and share data or any kind of updates on twitter. This chapter is about integrating twitter into your application.

There are two ways through which you can integrate twitter and share something from your application. These ways are listed below:

- Twitter SDK (Twitter4J)
- Intent Share

Integrating Twitter SDK

This is the first way of connecting with Twitter. You have to register your application and then receive some Application Id, and then you have to download the twitter SDK and add it to your project. The steps are listed below:

Registering your application

Create a new twitter application at dev.twitter.com/apps/new and fill all the information. It is shown below:

Application Details

Name: *

Your application name. This is used to attribute the source of a tweet and in user-facing

Description: *

Your application description, which will be shown in user-facing authorization screens.

Website: *

Your application's publicly accessible home page, where users can go to download, ma
source attribution for tweets created by your application and will be shown in user-fac
(If you don't have a URL yet, just put a placeholder here but remember to change it later

Callback URL:

Where should we return after successfully authenticating? For @Anywhere application
specify their `oauth_callback` URL on the request token step, regardless of the valu

Now under settings tab, change the access to read, write and access messages and save the settings. It is shown below:

Application Type

Access:

Read only

Read and Write

Read, Write and Access direct messages

What type of access does your application need? Note: @Anywhere applications requ
Find out more about our Application Permission Model

If everything works fine, you will receive a consumer ID with the secret. Just copy the application id and save it somewhere. It is shown in the image below:

OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should nev

Access level	Read, write, and direct messages About the application permission model
Consumer key	vkof05c2pvHSH8yrud29PA
Consumer secret	1uuNne4wIeYdRdhLaH2dXD1VNZws6Io6rn6a2
Request token URL	https://api.twitter.com/oauth/request
Authorize URL	https://api.twitter.com/oauth/authori

Downloading SDK and integrating it

Download twitter sdk <http://twitter4j.org/en/>. Copy the twitter4J jar into your project libs folder.

Posting tweets on twitter application

Once everything is complete, you can run the twitter 4J samples which can be found <http://twitter4j.org/en/code-examples.html>.

In order to use twitter, you need to instantiate an object of twitter class. It can be done by calling the static method **getSingleton()**. Its syntax is given below.

```
// The factory instance is re-useable and thread safe.
Twitter twitter = TwitterFactory.getSingleton();
```

In order to update the status, you can call `updateStatus()` method. Its syntax is given below:

```
Status status = twitter.updateStatus(latestStatus);
System.out.println("Successfully updated the status to [" +
status.getText() + "].");
```

Intent share

Intent share is used to share data between applications. In this strategy, we will not handle the SDK stuff, but let the twitter application handles it. We will simply call the twitter application and pass the data to share. This way, we can share something on twitter.

Android provides intent library to share data between activities and applications. To use it as share intent, we have to specify the type of the share intent to **ACTION_SEND**. Its syntax is given below:

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
```

Next thing you need is to define the type of data to pass, and then pass the data. Its syntax is given below:

```
shareIntent.setType("text/plain");
shareIntent.putExtra(Intent.EXTRA_TEXT, "Hello, from tutorialspoint");
startActivity(Intent.createChooser(shareIntent, "Share your thoughts"));
```

Apart from these methods, there are other methods available that allows intent handling. They are listed below:

Sr.No	Method & description
1	addCategory(String category) This method adds a new category to the intent.
2	createChooser(Intent target, CharSequence title) Convenience function for creating a ACTION_CHOOSER Intent.
3	getAction() This method retrieves the general action to be performed, such as ACTION_VIEW.
4	getCategories() This method returns the set of all categories in the intent.nt and the current scaling event.
5	putExtra(String name, int value) This method adds extended data to the intent.
6	toString() This method returns a string containing a concise, human-readable description of this object.

Example

Here is an example demonstrating the use of IntentShare to share data on twitter. It creates a basic application that allows you to share some text on twitter.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as IntentShare under a package com.example.intentshare. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.intentshare/MainActivity.java**.

```
package com.example.intentshare;

import java.io.File;
import java.io.FileOutputStream;

import com.example.intentshare.R;

import android.app.Activity;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.Intent;
```

```
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.view.Menu;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends Activity {

    private ImageView img;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        img = (ImageView) findViewById(R.id.imageView1);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void open(View view){
        Intent shareIntent = new Intent();
        shareIntent.setAction(Intent.ACTION_SEND);
        shareIntent.setType("text/plain");
        shareIntent.putExtra(Intent.EXTRA_TEXT, "Hello, from
        tutorialspoint");
        startActivity(Intent.createChooser(shareIntent, "Share your
        thoughts"));
    }
}
```

```

    }
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="98dp"
        android:layout_marginTop="139dp"
        android:onClick="open"
        android:src="@drawable/tp" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="48dp"
        android:text="@string/tap"

```

```

        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">IntentShare</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="tap">Tap the button to share something</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.intentshare"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.intentshare.MainActivity"
            android:label="@string/app_name" >

```

```

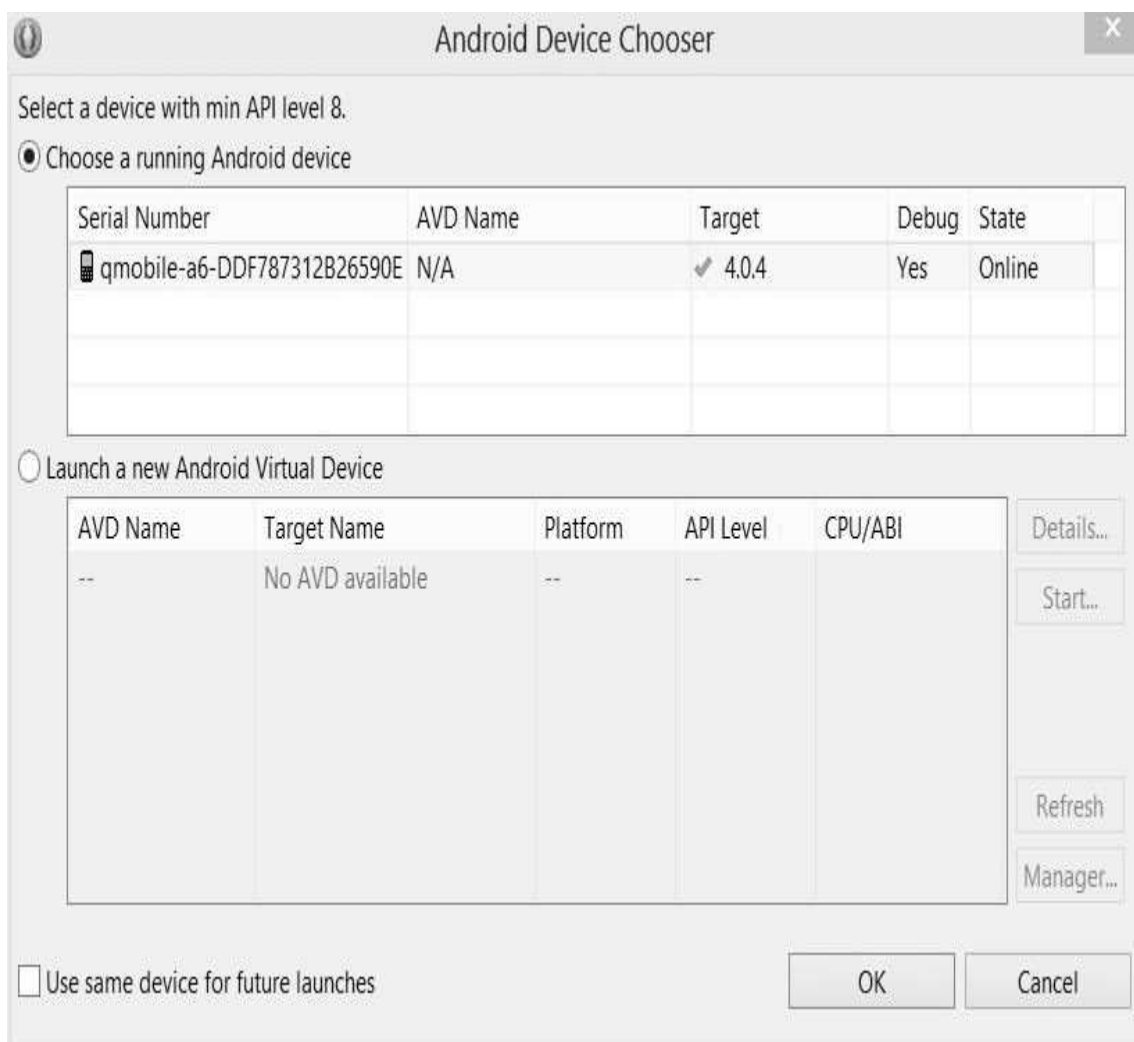
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>

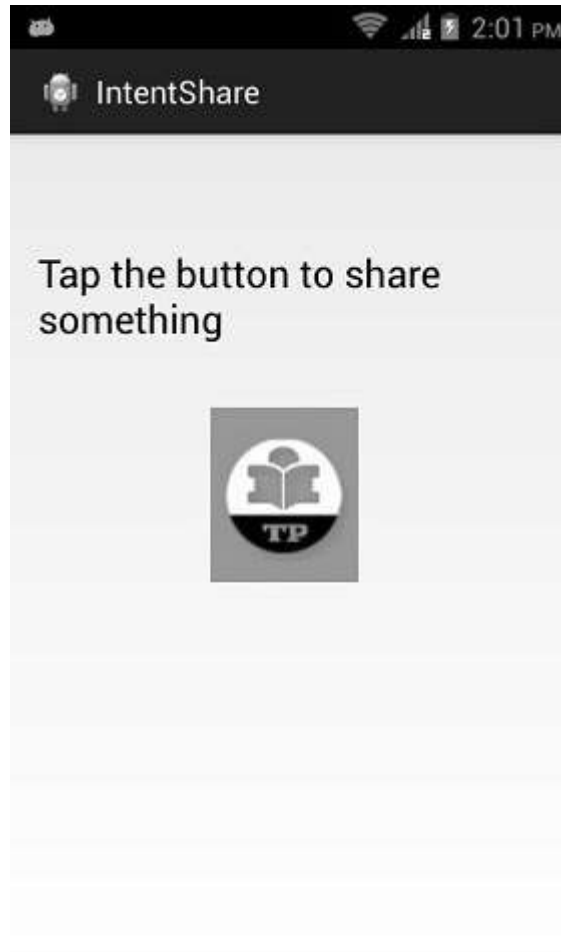
</manifest>

```

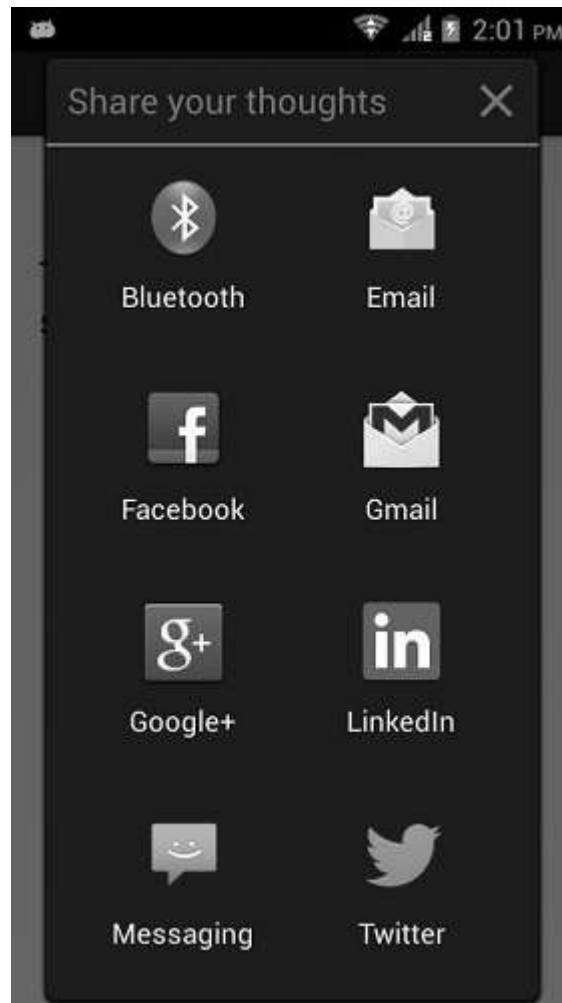
Let's try to run your IntentShare application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



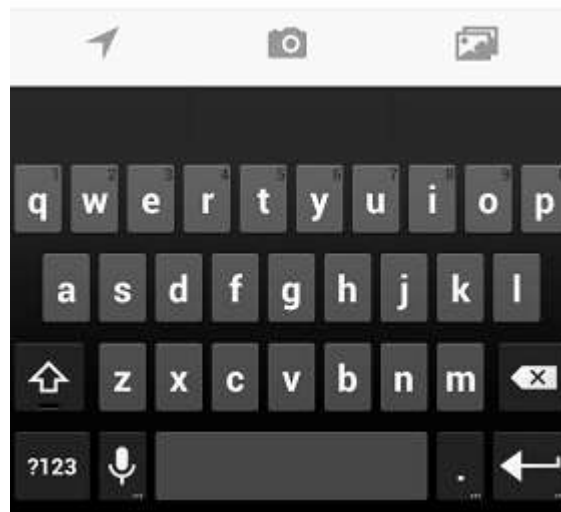
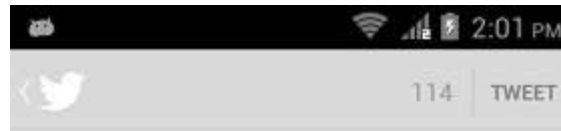
Select your mobile device as an option and then check your mobile device which will display your default screen:



Now just tap on the image logo and you will see a list of share providers.



Now just select twitter from that list and then write any message. It is shown in the image below:



Now just select the tweet button and then it would be posted on your twitter page. It is shown below:

Tweets



74. UI DESIGN

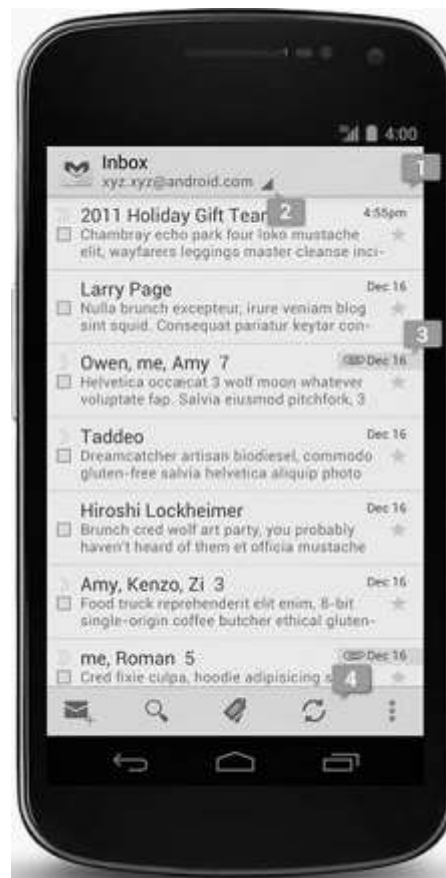
We will look at the different UI components of android screen in this chapter. We will also cover the tips to make a better UI design and also explain how to design a UI.

UI screen components

A typical user interface of an android application consists of action bar and the application content area.

- Main Action Bar
- View Control
- Content Area
- Split Action Bar

These components have also been shown in the image below:



Understanding Screen Components

The basic unit of android application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into equivalent Android GUI class with attributes represented by methods.

View and ViewGroups

An activity is consisted of views. A view is just a widget that appears on the screen. It could be button etc. One or more views can be grouped together into one ViewGroup. Example of ViewGroup include layouts.

Types of layout

There are many types of layout. Some of which are listed below:

- Linear Layout
- Absolute Layout
- Table Layout
- Frame Layout
- Relative Layout

Linear Layout

Linear layout is further divided into horizontal and vertical layout. It means it can arrange views in a single column or in a single row. Here is the code of linear layout (vertical) that includes a text view.

```
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
</LinearLayout>
```

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. It can be declared like this.

```
<AbsoluteLayout
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  xmlns:android="http://schemas.android.com/apk/res/android" >
  <Button
    android:layout_width="188dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_x="126px"
    android:layout_y="361px" />
</AbsoluteLayout>
```

TableLayout

The TableLayout groups views into rows and columns. It can be declared like this.

```
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_height="fill_parent"
  android:layout_width="fill_parent" >
  <TableRow>
  <TextView
    android:text="User Name:"
    android:width="120dp"
  />
  <EditText
    android:id="@+id/txtUserName"
    android:width="200dp" />
  </TableRow>
</TableLayout>
```

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other. It can be declared like this.

```
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >
</RelativeLayout>
```

FrameLayout

The FrameLayout is a placeholder on screen that you can use to display a single view. It can be declared like this.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/lblComments"
    android:layout_below="@+id/lblComments"
    android:layout_centerHorizontal="true" >
    <ImageView
        android:src = "@drawable/droid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</FrameLayout>
```

Apart from these attributes, there are other attributes that are common in all views and ViewGroups. They are listed below:

Sr.No	View & description
1	layout_width Specifies the width of the View or ViewGroup.
2	layout_height

	Specifies the height of the View or ViewGroup.
3	layout_marginTop Specifies extra space on the top side of the View or ViewGroup.
4	layout_marginBottom Specifies extra space on the bottom side of the View or ViewGroup.
5	layout_marginLeft Specifies extra space on the left side of the View or ViewGroup.
6	layout_marginRight Specifies extra space on the right side of the View or ViewGroup.
7	layout_gravity Specifies how child Views are positioned.
8	layout_weight Specifies how much of the extra space in the layout should be allocated to the View.

Units of Measurement

When you are specifying the size of an element on an Android UI, you should remember the following units of measurement.

Sr.No	Unit & description
1	dp Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen.
2	sp Scale-independent pixel. This is similar to dp and is recommended for specifying font sizes.
3	pt Point. A point is defined to be 1/72 of an inch, based on the physical screen size.

4	px Pixel. Corresponds to actual pixels on the screen.
---	---

Screen Densities

Sr.No	Density & DPI
1	Low density (ldpi) 120 dpi
2	Medium density (mdpi) 160 dpi
3	High density (hdpi) 240 dpi
4	Extra High density (xhdpi) 320 dpi

Optimizing layouts

Here are some of the guidelines for creating efficient layouts.

- Avoid unnecessary nesting
- Avoid using too many Views
- Avoid deep nesting

75. UI PATTERNS

This chapter tells you the different UI Patterns which are available by android to design apps that behave in a consistent and foreseeable way.

UI Patterns components

A good android application should follow following UI patterns:

- Action Bar
- Confirming and Acknowledging
- Settings
- Help
- Selection

Now we will discuss the above mentioned UI Patterns in detail.

Action Bar

The action bar is a dedicated bar at the top of each screen that is generally persistent throughout the app. It provides you several key function which are as following:

- Makes important actions prominent and accessible
- Supports consistent navigation and view switching within apps
- Reduces clutter by providing an action overflow for rarely used actions
- Provides a dedicated space for giving your app an identity

Action Bar Components

Action Bar has four major components which can be seen in the following image.



These components name and functionality is discussed below:

Sr.No	Action Bar Components
1	<p>App Icon</p> <p>The app icon establishes your app's identity. It can be replaced with a different logo or branding if you wish.</p>
2	<p>View control</p> <p>If your app displays data in different views, this segment of the action bar allows users to switch views.</p>
3	<p>Action buttons</p> <p>Shows the most important actions of your app in the actions section.</p>
4	<p>Action overflow</p> <p>Moves less often used actions to the action overflow.</p>

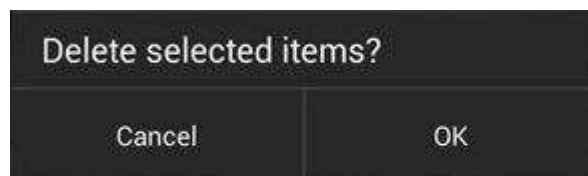
Confirming and Acknowledging

When a user invokes an action on your app's UI, it is a good practice to **confirm** or **acknowledge** that action through a toast or a dialog box.

There is a difference between Confirming and Acknowledging.

Confirming

When we ask the user to verify that they truly want to proceed with an action that they just invoked, it is called confirming. As you can see in the following image:



Acknowledging

When we display a toast to let the user know that the action they just invoked has been completed, is called acknowledging, as you can see in the following image:



Event created.

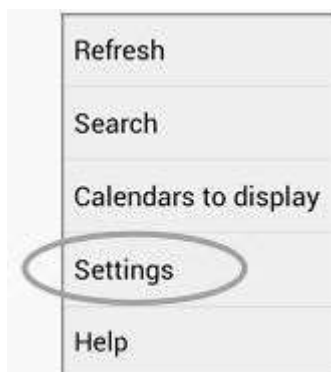
Settings

The place in your app where users can indicate their preferences for how your app should behave is called as Settings. The use of settings can benefit your app's users in the following ways:

- **Settings** help users to pre-determine what will happen in certain situations.
- Use of **settings** in your app help users to feel in control.

Placement of Settings

It is preferred by the android developers to always make "settings" option part of action overflow which is mentioned above. As users did not frequently use this options so the common practice is to place it below all other items except "Help". As you can see in the following picture:



Help

Some of your app users may run into some difficulty while using your app and they will be looking for some answers which they want within the app. So always make "help" a part of your app.

Placement of Help

Like "Settings" the standard design of placing "Help" option is in **action overflow**. Always make it very last item in the menu and always label it "Help". Even if your app screen has no other action overflow items, "Help" should appear there. As you can see this in the following picture:



Selection

Android 3.0 version changed the long press gesture to the global gesture to select data. The long press gesture is now used to select data, combining contextual actions and selection management functions for selected data into a new element called the **contextual action bar (CAB)**.

Using Contextual Action Bar (CAB)

The selection CAB is a temporary action bar that overlays your app's current action bar while data is selected. It appears after the user long presses on a selectable data item. As you can see in the following picture:



From the CAB bar, user can perform following actions:

- Select additional data items by touching them.
- Triggers an action from the CAB that applies to all highlighted data items.
- Dismiss the CAB via the navigation bar's Back button or the CAB's checkmark button.

76. UI TESTING

Android SDK provides the following tools to support automated, functional UI testing on your application.

- uiautomatorviewer
- uiautomator

uiautomatorviewer

A GUI tool to scan and analyze the UI components of an Android application.

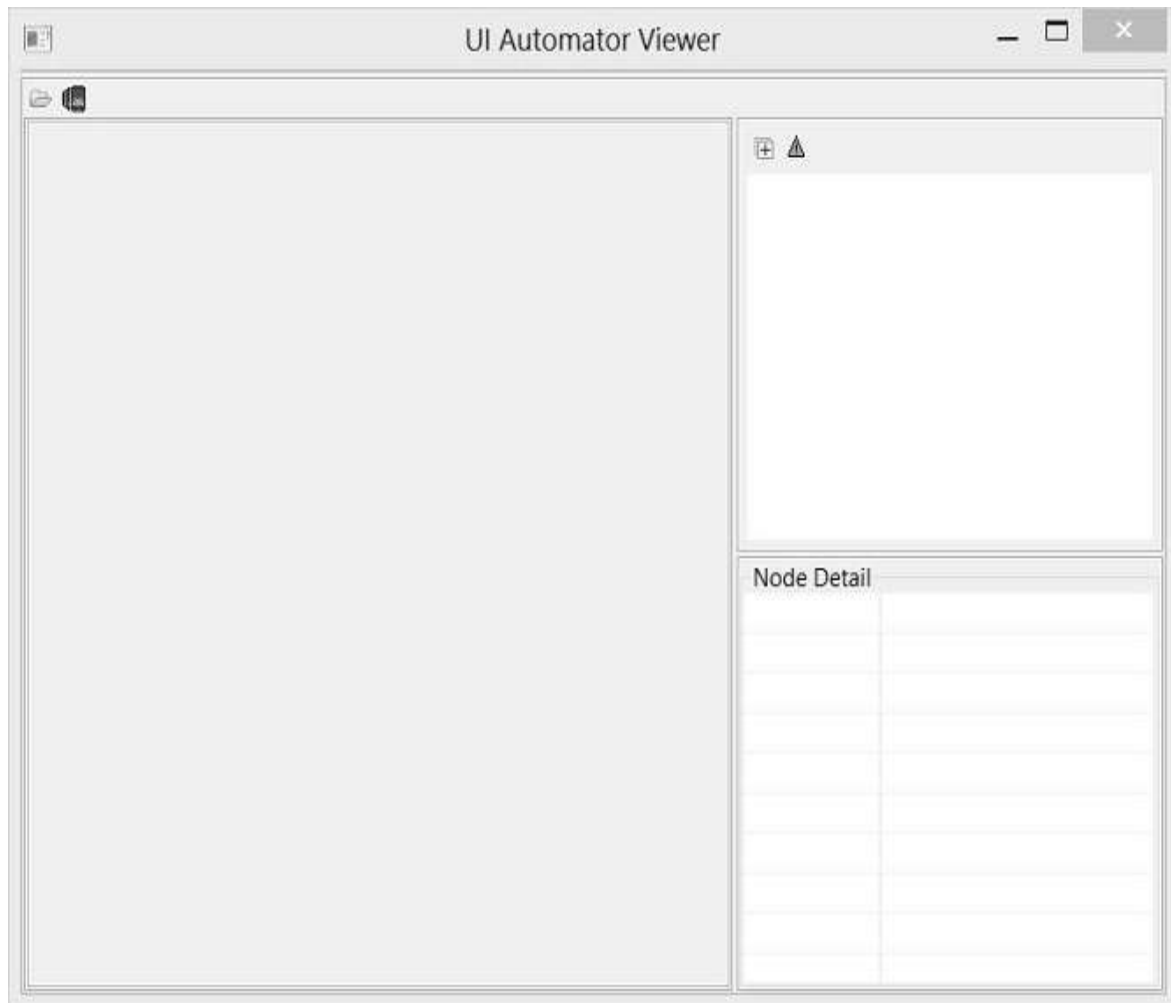
The uiautomatorviewer tool provides a convenient visual interface to inspect the layout hierarchy and view the properties of the individual UI components that are displayed on the test device. Using this information, you can later create uiautomator tests with selector objects that target specific UI components to test.

To analyze the UI components of the application that you want to test, perform the following steps after installing the application given in the example.

- Connect your Android device to your development machine.
- Open a terminal window and navigate to /tools/
- Run the tool with this command.

```
uiautomatorviewer
```

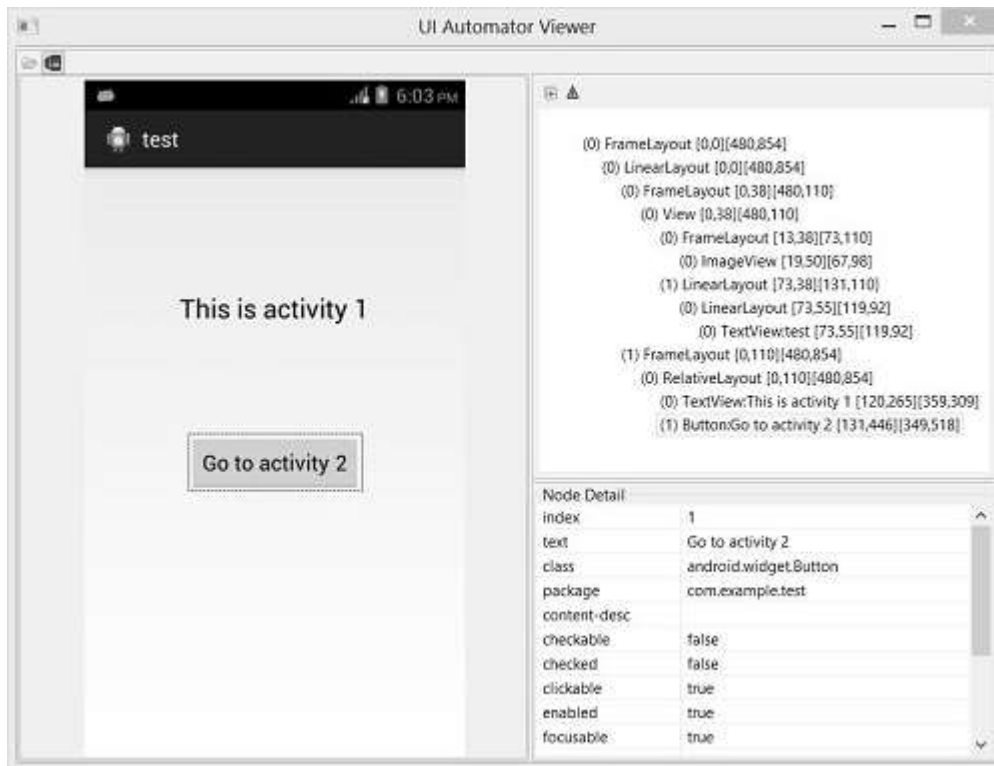
You will see the following window appear. It is the default window of the UI Automator Viewer.



- Click on the devices icon at the top right corner. It will start taking the UI XML snapshot of the screen currently opened in the device. It would be something like this.

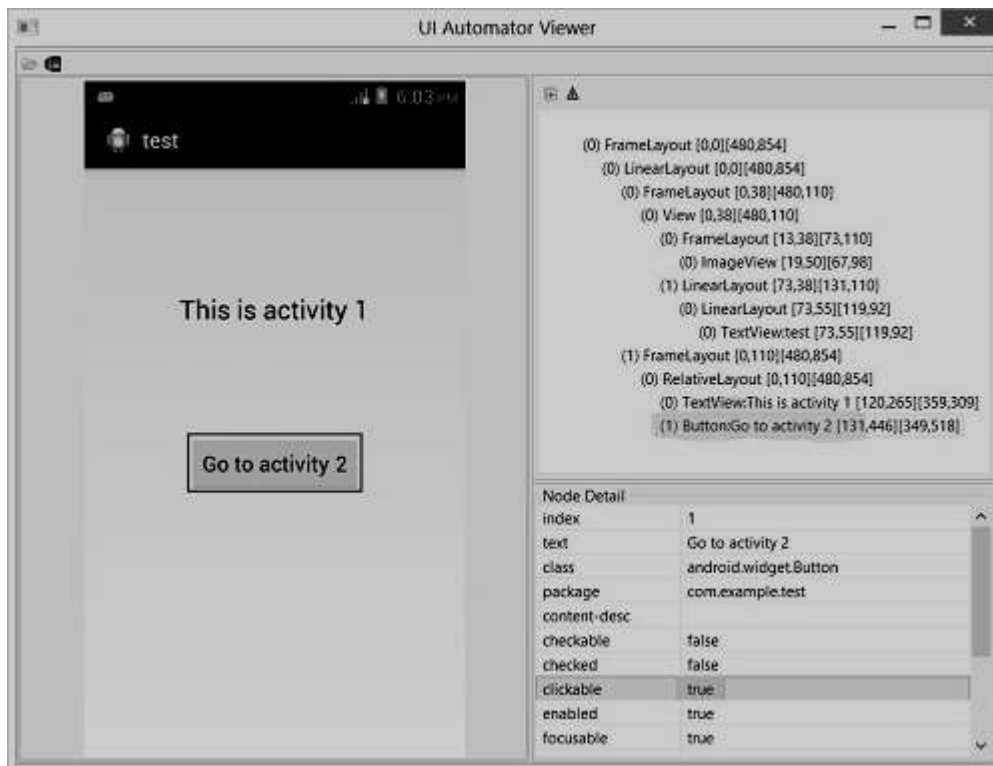


After that, you will see the snapshot of your device screen in the uiautomatorviewer window.

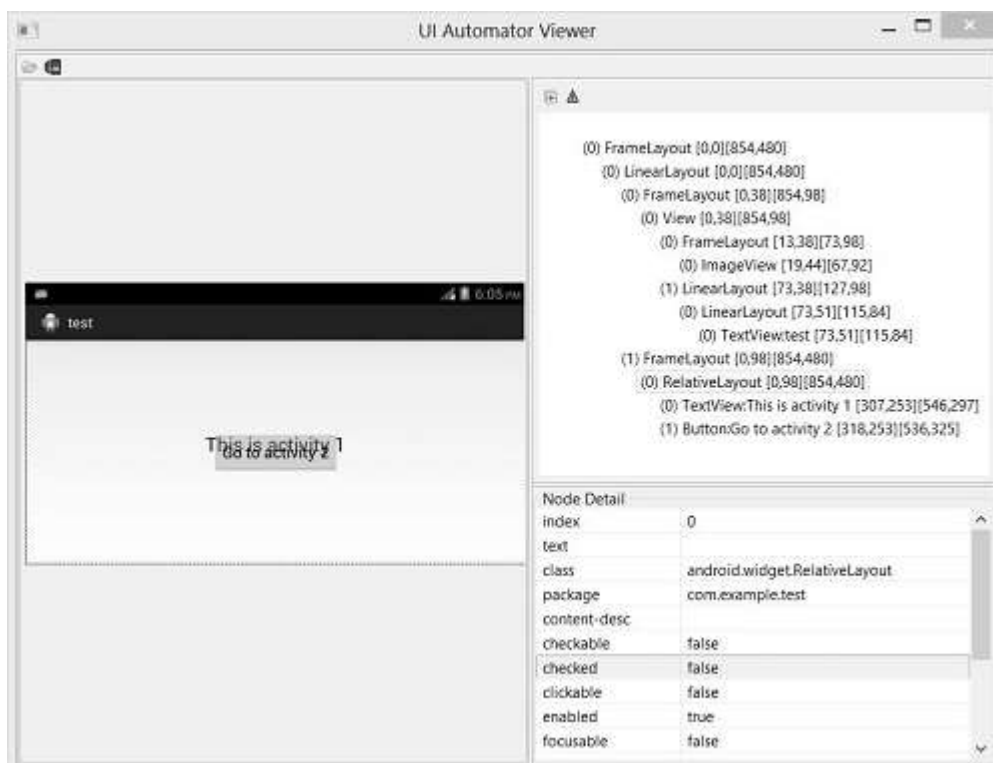


On the right side of this window, you will see two partitions. The upper partition explains the Nodes structure, the way the UI components are arranged and contained. Clicking on each node gives detail in the lower partition.

As an example, consider the below figure. When you click on the button, you can see in the upper partition that Button is selected, and in the lower partition, its details are shown. Since this button is clickable, its property of clickable is set to true.



UI Automator Viewer also helps you to examine your UI in different orientations. For example, just change your device orientation to landscape, and again capture the screenshot. It is shown in the figure below:



uiautomator

Now you can create your own test cases and run it with uiautomatorviewer to examine them. In order to create your own test case, you need to perform the following steps:

- From the Project Explorer, right-click on the new project that you created, then select Properties > Java Build Path, and do the following:
- Click Add Library > JUnit then select JUnit3 to add JUnit support.
- Click Add External JARs... and navigate to the SDK directory. Under the platforms directory, select the latest SDK version and add both the uiautomator.jar and android.jar files.
- Extend your class with UiAutomatorTestCase.
- Right the necessary test cases.
- Once you have coded your test, follow these steps to build and deploy your test JAR to your target Android test device.
- Create the required build configuration files to build the output JAR. To generate the build configuration files, open a terminal and run the following command:

```
<android-sdk>/tools/android create uitest-project -n <name> -t 1 -p
<path>
```

<name> is the name of the project that contains your uiautomator test source files, and <path> is the path to the corresponding project directory.

- From the command line, set the ANDROID_HOME variable.

```
set ANDROID_HOME=<path_to_your_sdk>
```

- Go to the project directory where your build.xml file is located and build your test JAR.

```
ant build
```

- Deploy your generated test JAR file to the test device by using the adb push command.

```
adb push /data/local/tmp/
```

- Run your test by following command:

```
adb shell uiautomator runtest LaunchSettings.jar -c
com.uia.example.my.LaunchSettings
```

Example

The below example demonstrates the use of UITesting. It creates a basic application which can be used for uiautomatorviewer.

To experiment with this example, you need to run this on an actual device and then follow the uiautomatorviewer steps explained in the beginning.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Test under a package com.example.test. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add Activity code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Create src/MainActivity2.java file to add Activity code.
5	Modify layout XML file res/layout/activity_main_activity2.xml add any GUI component if required.
6	Modify res/values/string.xml file and add necessary string components.
7	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/com.example.test/MainActivity.java**.

```
package com.example.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
```



```
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void activity2(View view){
        Intent intent = new
        Intent(this,com.example.test.MainActivity2.class);
        startActivity(intent);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

Here is the content of **src/com.example.test/MainActivity2.java**.

```
package com.example.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
```

```

import android.view.View;

public class MainActivity2 extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_activity2);
    }

    public void activity1(View view){
        Intent intent = new
        Intent(this,com.example.test.MainActivity.class);
        startActivity(intent);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main_activity2, menu);
        return true;
    }
}

```

Here is the content of **activity_main.xml**

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"

```

```

        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".MainActivity" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="87dp"
            android:text="@string/test1"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="true"
            android:onClick="activity2"
            android:text="@string/go2" />
    </RelativeLayout>

```

Here is the content of **activity_main_activity2.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity2" >

```

```

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="125dp"
    android:text="@string/test2"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:onClick="activity1"
    android:text="@string/go1" />
</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

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

    <string name="app_name">test</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="test1">This is activity 1</string>
    <string name="test2">This is activity 2</string>
    <string name="go1">Go to activity 1</string>
    <string name="go2">Go to activity 2</string>
    <string name="title_activity_main_activity2">MainActivity2</string>

</resources>

```

Here is the content of **AndroidManifest.xml**.


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.test"
    android:versionCode="1"
    android:versionName="1.0" >

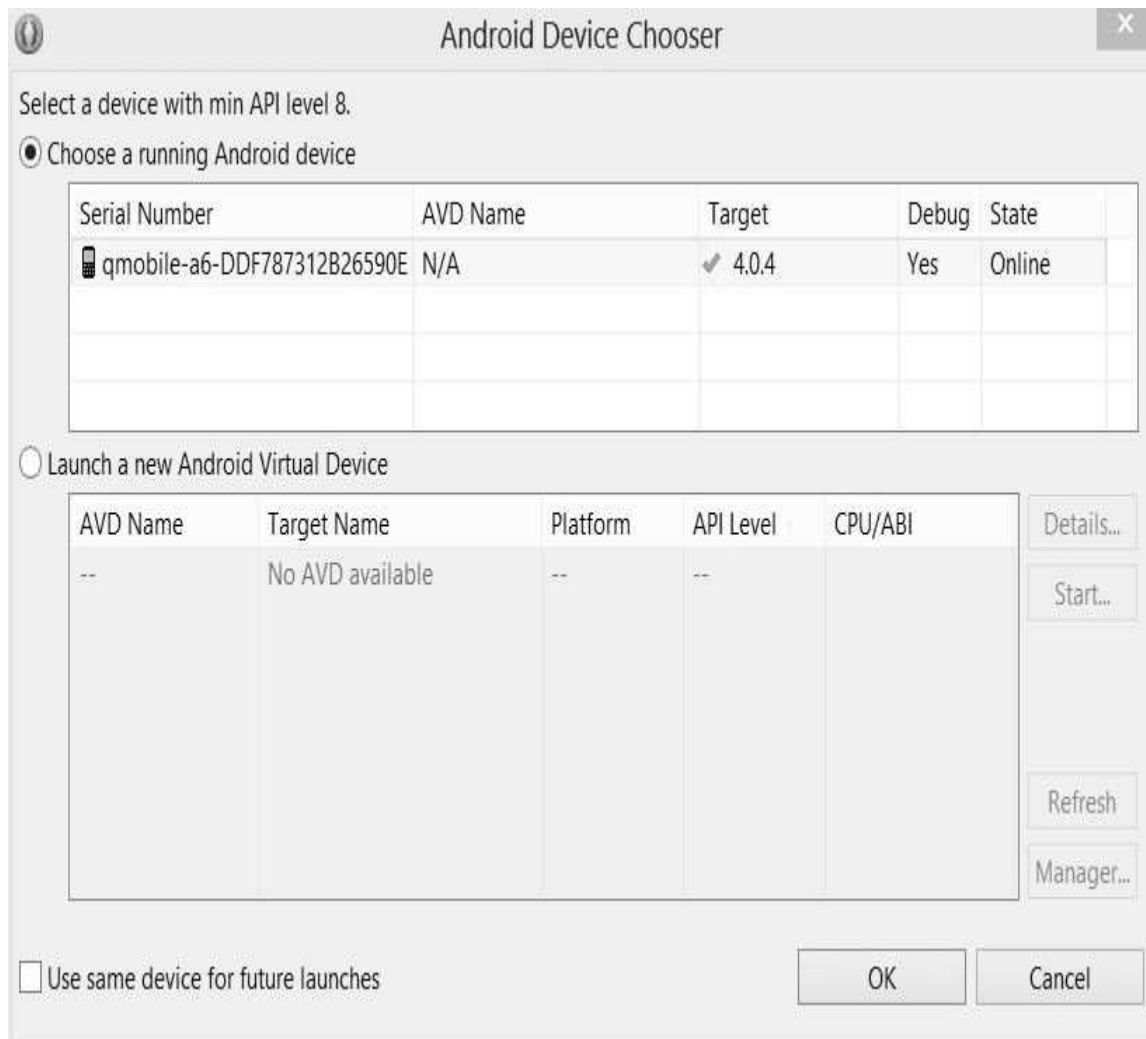
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="14" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.test.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.example.test.MainActivity2"
            android:label="@string/title_activity_main_activity2" >
        </activity>
    </application>
</manifest>
```

Let's try to run your UI Testing application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from

Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display application screen. Now just follow the steps mentioned at the top under the uiautomatorviewer section in order to perform uitesting on this application.

77. WEBVIEW

WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView turns your application to a web application.

In order to add WebView to your application, you have to add **<WebView>** element to your xml layout file. Its syntax is as follows:

```
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

In order to use it, you have to get a reference of this view in Java file. To get a reference, create an object of the class WebView. Its syntax is:

```
WebView browser = (WebView) findViewById(R.id.webview);
```

In order to load a web url into the WebView, you need to call a method **loadUrl(String url)** of the WebView class, specifying the required url. Its syntax is:

```
browser.loadUrl("http://www.tutorialspoint.com");
```

Apart from just loading url, you can have more control over your WebView by using the methods defined in WebView class. They are listed as follows:

Sr.No	Method & Description
1	canGoBack() This method specifies whether the WebView has a back history item.
2	canGoForward() This method specifies whether the WebView has a forward history item.
3	clearHistory()

	This method clears the WebView forward and backward history.
4	destroy() This method destorys the internal state of WebView.
5	findAllAsync(String find) This method find all instances of string and highlight them.
6	getProgress() This method gets the progress of the current page.
7	getTitle() This method returns the title of the current page.
8	getUrl() This method returns the url of the current page.

If you click on any link inside the webpage of the WebView, that page will not be loaded inside your WebView. To do that you need to extend your class from **WebViewClient** and override its method. Its syntax is:

```
private class MyBrowser extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
    }
}
```

Example

Here is an example demonstrating the use of WebView Layout. It creates a basic web application that will ask you to specify a url and will load this url website in the WebView.

To experiment with this example, you need to run this on an actual device on which internet is running.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as WebView under a package com.example.webview. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add WebView code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Modify the AndroidManifest.xml to add the necessary permissions.
6	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.webview/MainActivity.java**.

```
package com.example.webview;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.Window;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {
```

```
private EditText field;
private WebView browser;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    field = (EditText)findViewById(R.id.urlField);
    browser = (WebView)findViewById(R.id.webView1);
    browser.setWebViewClient(new MyBrowser());
}

public void open(View view){
    String url = field.getText().toString();
    browser.getSettings().setLoadsImagesAutomatically(true);
    browser.getSettings().setJavaScriptEnabled(true);
    browser.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);
    browser.loadUrl(url);
}

private class MyBrowser extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
}
```

```

    return true;
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <EditText
        android:id="@+id/urlField"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:ems="10" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"

```

```

        android:layout_height="wrap_content"
        android:layout_below="@+id/urlField"

        android:layout_centerHorizontal="true"
        android:onClick="open"
        android:text="@string/browse" />

<WebView
    android:id="@+id/webView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_alignParentBottom="true"
    android:layout_below="@+id/button1" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">WebView</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">URL:</string>
    <string name="browse">Browse</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.webview"
    android:versionCode="1"
    android:versionName="1.0" >


```

```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.INTERNET"/>

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.webview.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

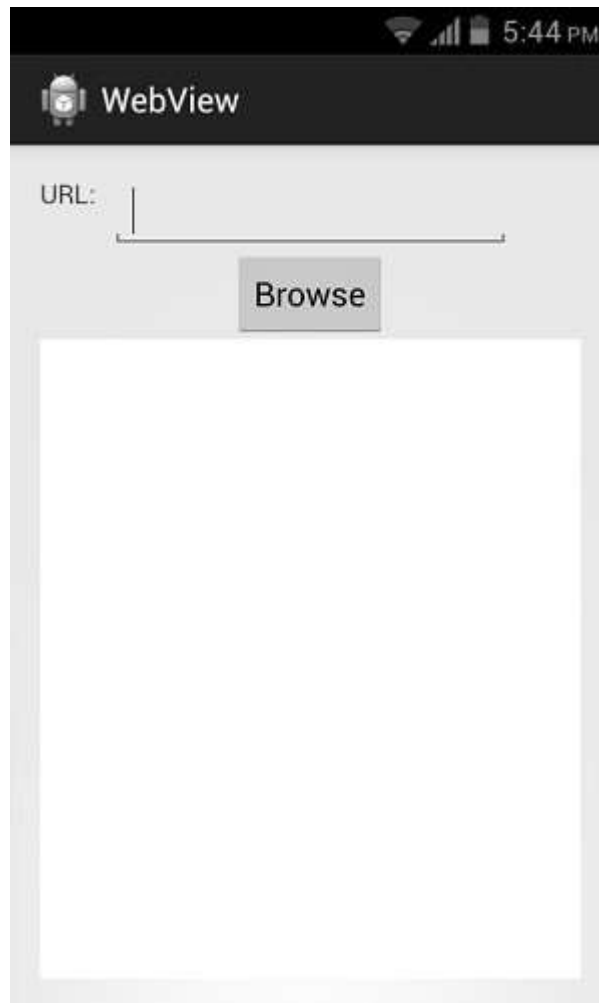
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Let's try to run your WebView application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display your default screen:



Now just specify a url on the url field and press the browse button that appears, to launch the website. But before that please make sure that you are connected to the internet. After pressing the button, the following screen would appear:



Note: By just changing the url in the url field, your WebView will open your desired website.

78. WI-FI

Android allows applications to view the access of the state of the wireless connections at a very low level. Application can access almost all the information of a wi-fi connection.

The information that an application can access includes connected network's link speed, IP address, negotiation state, other networks information. Applications can also scan, add, save, terminate and initiate Wi-Fi connections.

Android provides **WifiManager** API to manage all aspects of WIFI connectivity. We can instantiate this class by calling **getSystemService** method. Its syntax is given below:

```
WifiManager mainWifiObj;  
mainWifiObj = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

In order to scan a list of wireless networks, you also need to register your BroadcastReceiver. It can be registered using **registerReceiver** method with argument of your receiver class object. Its syntax is given below:

```
class WifiScanReceiver extends BroadcastReceiver {  
    public void onReceive(Context c, Intent intent) {  
    }  
}  
  
WifiScanReceiver wifiReceiver = new WifiScanReceiver();  
registerReceiver(wifiReceiver, new  
IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
```

The wi-fi scan can be started by calling the **startScan** method of the WifiManager class. This method returns a list of ScanResult objects. You can access any object by calling the **get** method of list. Its syntax is given below:

```
List<ScanResult> wifiScanList = mainWifiObj.getScanResults();  
String data = wifiScanList.get(0).toString();
```

Apart from just Scanning, you can have more control over your WIFI by using the methods defined in WifiManager class. They are listed as follows:

Sr.No	Method & Description
1	addNetwork(WifiConfiguration config) This method adds a new network description to the set of configured networks.
2	createWifiLock(String tag) This method creates a new WifiLock.
3	disconnect() This method disassociates from the currently active access point.
4	enableNetwork(int netId, boolean disableOthers) This method allows a previously configured network to be associated with.
5	getWifiState() This method gets the Wi-Fi enabled state.
6	isWifiEnabled() This method returns whether Wi-Fi is enabled or disabled.
7	setWifiEnabled(boolean enabled) This method enables or disables Wi-Fi.
8	updateNetwork(WifiConfiguration config) This method updates the network description of an existing configured network.

Example

Here is an example demonstrating the use of WIFI. It creates a basic application that scans a list of wireless networks and populate them in a list view.

To experiment with this example, you need to run this on an actual device on which wi-fi is turned on.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as WIFI under a package com.example.wifi. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add WebView code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the AndroidManifest.xml to add the necessary permissions.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.wifi/MainActivity.java**.

```
package com.example.wifi;

import java.util.List;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
```

```
import android.widget.Toast;

public class MainActivity extends Activity {

    WifiManager mainWifiObj;
    WifiScanReceiver wifiReceiver;
    ListView list;
    String wifis[];
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        list = (ListView)findViewById(R.id.listView1);
        mainWifiObj = (WifiManager) getSystemService(Context.WIFI_SERVICE);
        wifiReceiver = new WifiScanReceiver();
        mainWifiObj.startScan();
    }

    protected void onPause() {
        unregisterReceiver(wifiReceiver);
        super.onPause();
    }

    protected void onResume() {
        registerReceiver(wifiReceiver, new IntentFilter(
            WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
        super.onResume();
    }

    class WifiScanReceiver extends BroadcastReceiver {
        @SuppressWarnings("UseValueOf")
        public void onReceive(Context c, Intent intent) {
            List<ScanResult> wifiScanList = mainWifiObj.getScanResults();
            wifis = new String[wifiScanList.size()];
        }
    }
}
```

```

        for(int i = 0; i < wifiScanList.size(); i++){
            wifis[i] = ((wifiScanList.get(i)).toString());
        }

        list.setAdapter(new
        ArrayAdapter<String>(getApplicationContext(),
        android.R.layout.simple_list_item_1,wifis));
    }
}
}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:drawSelectorOnTop="false"
        android:background="@android:color/background_dark"
        android:listSelector="@android:color/darker_gray" >

```

```

</ListView>
</RelativeLayout>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.wifi"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
    />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"
    />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.wifi.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />


                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity

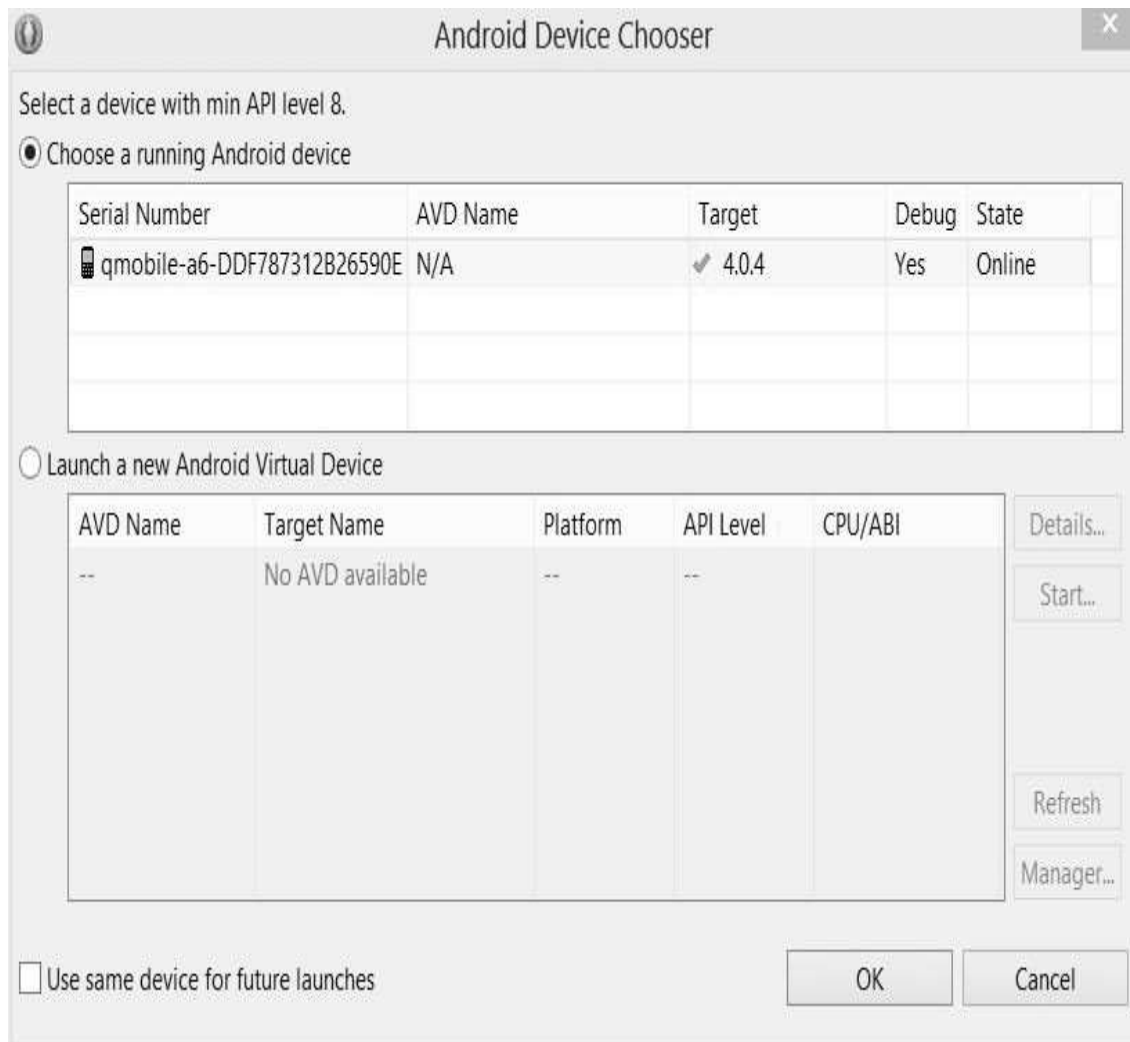
```

```

        android:name="com.example.wifi.ListWifiActivity"
        android:label="@string/title_activity_list_wifi" >
    </activity>
</application>
</manifest>

```

Let's try to run your WIFI application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display your mobile screen filled with wireless networks around you. It is shown below:



Note the information that has been returned to you. It contains much information about each of the wireless network detected.

79. WIDGETS

A widget is a small gadget or control of your android application placed on the home screen. Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them. You have probably seen some common widgets, such as music widget, weather widget, clock widget etc.

Widgets could be of many types such as information widgets, collection widgets, control widgets and hybrid widgets. Android provides us a complete framework to develop our own widgets.

Widget - XML file

In order to create an application widget, first thing you need is AppWidgetProviderInfo object, which you will define in a separate widget XML file. To do that, right click on your project and create a new folder called **xml**. Now right click on the newly created folder and create a new XML file. The resource type of the XML file should be set to **AppWidgetProvider**. In the xml file, define some properties which are as follows:

```
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dp"
    android:updatePeriodMillis="0"
    android:minHeight="146dp"
    android:initialLayout="@layout/activity_main">
</appwidget-provider>
```

Widget - Layout file

Now you have to define the layout of your widget in your default XML file. You can drag components to generate auto xml.

Widget - Java file

After defining layout, now create a new JAVA file or use existing one, and extend it with **AppWidgetProvider** class and override its update method as follows.

In the update method, you have to define the object of two classes which are PendingIntent and RemoteViews. Its syntax is:

```
PendingIntent pending = PendingIntent.getActivity(context, 0, intent, 0);
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.activity_main);
```

In the end you have to call an update method `updateAppWidget()` of the `AppWidgetManager` class. Its syntax is:

```
appWidgetManager.updateAppWidget(currentWidgetId,views);
```

A part from the `updateAppWidget` method, there are other methods defined in this class to manipulate widgets. They are as follows:

Sr.No	Method & Description
1	onDeleted(Context context, int[] appWidgetIds) This is called when an instance of <code>AppWidgetProvider</code> is deleted.
2	onDisabled(Context context) This is called when the last instance of <code>AppWidgetProvider</code> is deleted
3	onEnabled(Context context) This is called when an instance of <code>AppWidgetProvider</code> is created.
4	onReceive(Context context, Intent intent) It is used to dispatch calls to the various methods of the class

Widget - Manifest file

You also have to declare the `AppWidgetProvider` class in your manifest file as follows:

```
<receiver android:name="ExampleAppWidgetProvider" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data android:name="android.appwidget.provider"
        android:resource="@xml/example_appwidget_info" />
</receiver>
```

Example

Here is an example demonstrating the use of application Widget. It creates a basic widget applications that will open this current website in the browser.

To experiment with this example, you need to run this on an actual device on which internet is running.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as Widget under a package com.example.widget. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add widget code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Create a new folder and xml file under res/xml/mywidget.xml to add respective XML components.
5	Modify the res/values/string.xml to add necessary string components.
6	Modify the AndroidManifest.xml to add the necessary permissions.
7	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.widget/MainActivity.java**.

```
package com.example.widget;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.app.PendingIntent;
import android.appwidget.AppWidgetManager;
```

```
import android.appwidget.AppWidgetProvider;
import android.content.Context;

import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.webkit.WebView.FindListener;
import android.widget.Button;
import android.widget.RemoteViews;
import android.widget.Toast;

public class MainActivity extends AppWidgetProvider{

    @Override
    public void onUpdate(Context context, AppWidgetManager
    appWidgetManager,
    int[] appWidgetIds) {
        for(int i=0; i<appWidgetIds.length; i++){
            int currentWidgetId = appWidgetIds[i];
            String url = "http://www.tutorialspoint.com";
            Intent intent = new Intent(Intent.ACTION_VIEW);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            intent.setData(Uri.parse(url));
            PendingIntent pending = PendingIntent.getActivity(context, 0,
            intent, 0);
            RemoteViews views = new RemoteViews(context.getPackageName(),
            R.layout.activity_main);
            views.setOnClickPendingIntent(R.id.button1, pending);
            appWidgetManager.updateAppWidget(currentWidgetId,views);
            Toast.makeText(context, "widget added", Toast.LENGTH_SHORT).show();

        }
    }
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<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"
    android:gravity="top"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="@string/website"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginLeft="18dp"
        android:text="@string/app_name" />

</RelativeLayout>
```

Following is the content of the **res/xml/mywidget.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dp"
    android:updatePeriodMillis="0"
    android:minHeight="146dp"
    android:initialLayout="@layout/activity_main">
</appwidget-provider>
```

Following is the content of the **res/values/string.xml**.

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

    <string name="app_name">Widget</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="website">TutorialsPoint.com</string>


</resources>
```

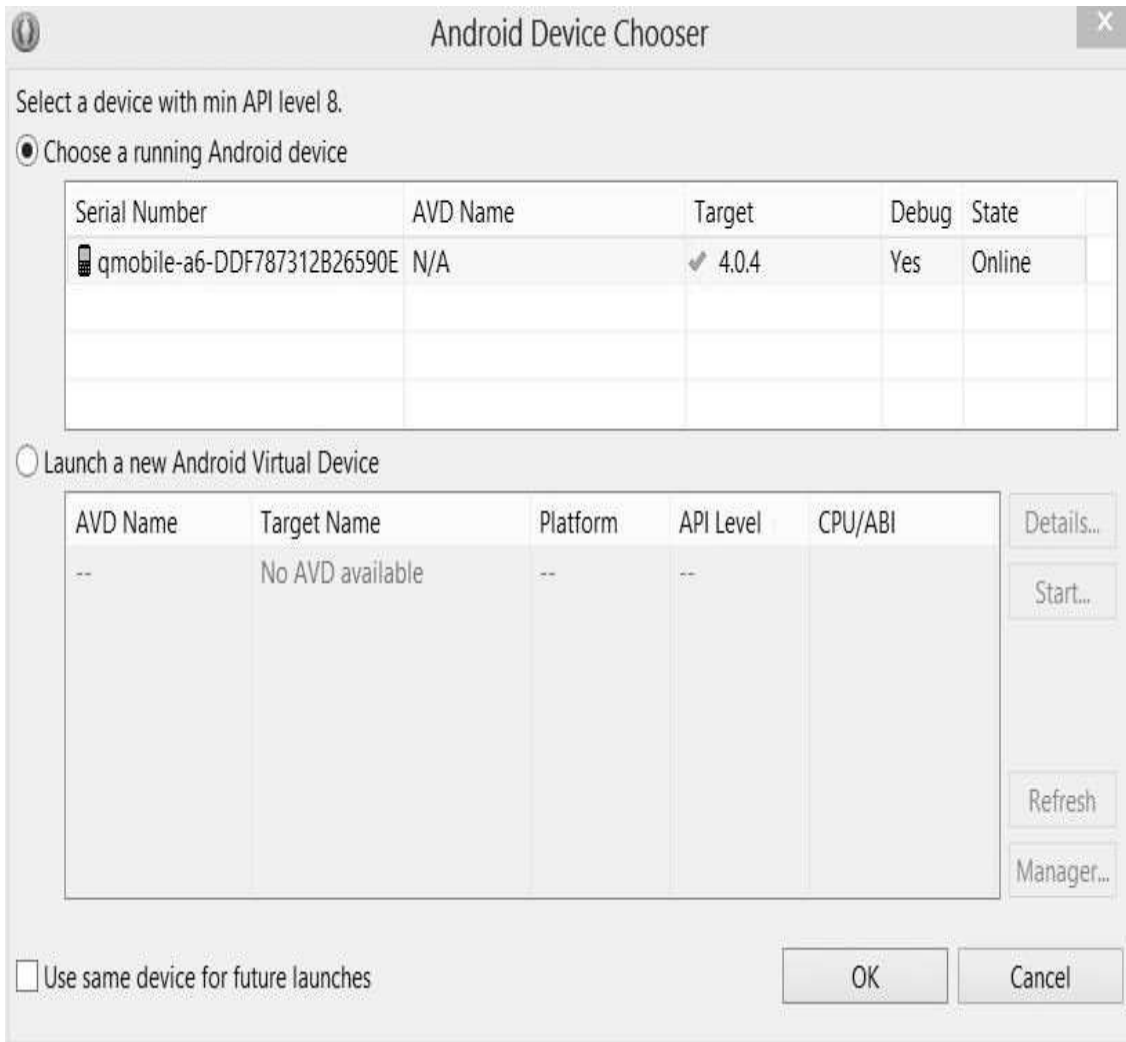
Following is the content of **AndroidManifest.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.widget"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET"/>
```

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="MainActivity" >
        <intent-filter>
            <action
                android:name="android.appwidget.action.APPWIDGET_UPDATE"
            />
        </intent-filter>
        <meta-data android:name="android.appwidget.provider"
            android:resource="@xml/mywidget" />
    </receiver>
</application>
</manifest>
```

Let's try to run your Widget application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display your default screen:

Go to your widget section and add your created widget to the desktop or homescreen. It would look something like this:



Now just tap on the widget button that appears, to launch the browser. But before that please make sure that you are connected to the internet. After pressing the button, the following screen would appear:



Note. By just changing the url in the java file, your widget will open your desired website in the browser.

80. XML PARSER

XML stands for Extensible Markup Language. XML is a very popular format and commonly used for sharing data on the internet. This chapter explains how to parse the XML file and extract necessary information from it.

Android provides three types of XML parsers which are **DOM, SAX and XMLPullParser**. Among all of them android recommend XMLPullParser because it is efficient and easy to use. So we are going to use XMLPullParser for parsing XML.

The first step is to identify the fields in the XML data in which you are interested in. For example, in the XML given below we are interested in getting temperature only.

```
<?xml version="1.0"?>
<current>
  <city id="2643743" name="London">
    <coord lon="-0.12574" lat="51.50853"/>
    <country>GB</country>
    <sun rise="2013-10-08T06:13:56" set="2013-10-08T17:21:45"/>
  </city>
  <temperature value="289.54" min="289.15" max="290.15" unit="kelvin"/>
  <humidity value="77" unit="%"/>
  <pressure value="1025" unit="hPa"/>
</country>
```

XML - Elements

An xml file consist of many components. Here is the table defining the components of an XML file and their description.

Sr.No	Component & description
1	Prolog An XML file starts with a prolog. The first line that contains the information about a file is prolog
2	Events

	An XML file has many events. Event could be like this. Document starts, Document ends, Tag starts, Tag ends and Text etc.
3	Text Apart from tags and events, and xml file also contains simple text. Such as GB is a text in the country tag.
4	Attributes Attributes are the additional properties of a tag such as value etc.

XML - Parsing

In the next step, we will create XMLPullParser object, but in order to create that we will first create XmlPullParserFactory object and then call its newPullParser() method to create XMLPullParser. Its syntax is given below:

```
private XmlPullParserFactory xmlFactoryObject =
XmlPullParserFactory.newInstance();

private XmlPullParser myparser = xmlFactoryObject.newPullParser();
```

The next step involves specifying the file for XmlPullParser that contains XML. It could be a file or could be a Stream. In our case it is a stream. Its syntax is given below:

```
mysparser.setInput(stream, null);
```

The last step is to parse the XML. An XML file consist of events, Name, Text, AttributesValue etc. So XMLPullParser has a seperate function for parsing each of the component of XML file. Its syntax is given below:

```
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)
{
    String name=myParser.getName();
    switch (event){
        case XmlPullParser.START_TAG:
            break;
        case XmlPullParser.END_TAG:
            if(name.equals("temperature")){
                temperature = myParser.getAttributeValue(null,"value");
            }
    }
}
```

```

    }
    break;
}
event = myParser.next();
}

```

The method **getEventType** returns the type of event that happens. e.g: Document start, tag start etc. The method **getName** returns the name of the tag and since we are only interested in temperature, so we just check in conditional statement that if we get a temperature tag, we call the method **getAttributeValue** to return us the value of temperature tag.

Apart from these methods, there are other methods provided by this class for better parsing XML files. These methods are listed below:

Sr.No	Method & description
1	getAttributeCount() This method just returns the number of attributes of the current start tag.
2	getAttributeName(int index) This method returns the name of the attribute specified by the index value.
3	getColumnNumber() This method returns the current column number, starting from 0.
4	getDepth() This method returns the current depth of the element.
5	getLineNumber() Returns the current line number, starting from 1.
6	getNamespace() This method returns the namespace URI of the current element.
7	getPrefix()

	This method returns the prefix of the current element.
8	getName() This method returns the name of the tag.
9	getText() This method returns the text for that particular element.
10	isWhitespace() This method checks whether the current TEXT event contains only whitespace characters.

Example

Here is an example demonstrating the use of XMLPullParser class. It creates a basic Weather application that allows you to parse XML from google weather api and shows the result.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as XMLParser under a package com.example.xmlparser. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Create a new java file under src/HandleXML.java to fetch and parse XML data.
6	Modify AndroidManifest.xml to add necessary internet permission.

- | | |
|---|---|
| 7 | Run the application and choose a running android device and install the application on it and verify the results. |
|---|---|

Following is the content of the modified main activity file **src/com.example.xmlparser/MainActivity.java**.

```
package com.example.xmlparser;

import java.io.IOException;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.ParseException;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private String url1 =
        "http://api.openweathermap.org/data/2.5/weather?q=";
    private String url2 = "&mode=xml";
    private EditText location, country, temperature, humidity, pressure;
    private HandleXML obj;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        location = (EditText)findViewById(R.id.editText1);

        country = (EditText)findViewById(R.id.editText2);
        temperature = (EditText)findViewById(R.id.editText3);
        humidity = (EditText)findViewById(R.id.editText4);
        pressure = (EditText)findViewById(R.id.editText5);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        // if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void open(View view){
        String url = location.getText().toString();
        String finalUrl = url1 + url + url2;
        country.setText(finalUrl);
        obj = new HandleXML(finalUrl);
        obj.fetchXML();
        while(obj.parsingComplete);
        country.setText(obj.getCountry());
        temperature.setText(obj.getTemperature());
        humidity.setText(obj.getHumidity());
        pressure.setText(obj.getPressure());

    }
}
```


Following is the content of **src/com.example.xmlparser/HandleXML.java**.

```
package com.example.xmlparser;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class HandleXML {

    private String country = "county";
    private String temperature = "temperature";
    private String humidity = "humidity";
    private String pressure = "pressure";
    private String urlString = null;
    private XmlPullParserFactory xmlFactoryObject;
    public volatile boolean parsingComplete = true;
    public HandleXML(String url){
```

```
        this.urlString = url;
    }
    public String getCountry(){
        return country;
    }
    public String getTemperature(){
        return temperature;
    }
    public String getHumidity(){
        return humidity;
    }
    public String getPressure(){
        return pressure;
    }

    public void parseXMLAndStoreIt(XmlPullParser myParser) {
        int event;
        String text=null;
        try {
            event = myParser.getEventType();
            while (event != XmlPullParser.END_DOCUMENT) {
                String name=myParser.getName();
                switch (event){
                    case XmlPullParser.START_TAG:
                        break;
                    case XmlPullParser.TEXT:
                        text = myParser.getText();
                        break;

                    case XmlPullParser.END_TAG:
                        if(name.equals("country")){
                            country = text;
                        }
                        else if(name.equals("humidity")){
```

```

        humidity = myParser.getAttributeValue(null,"value");
    }
    else if(name.equals("pressure")){
        pressure = myParser.getAttributeValue(null,"value");
    }
    else if(name.equals("temperature")){
        temperature =
myParser.getAttributeValue(null,"value");
    }
    else{
    }
    break;
    }
    event = myParser.next();

    }
    parsingComplete = false;
} catch (Exception e) {
    e.printStackTrace();
}

}

public void fetchXML(){
    Thread thread = new Thread(new Runnable(){
        @Override
        public void run() {
            try {
                URL url = new URL(urlString);
                HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
                conn.setReadTimeout(10000 /* milliseconds */);
                conn.setConnectTimeout(15000 /* milliseconds */);
                conn.setRequestMethod("GET");
                conn.setDoInput(true);

```

```

        conn.connect();
        InputStream stream = conn.getInputStream();

        xmlFactoryObject = XmlPullParserFactory.newInstance();
        XmlPullParser myparser = xmlFactoryObject.newPullParser();

        myparser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES
, false);
        myparser.setInput(stream, null);
        parseXMLAndStoreIt(myparser);
        stream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

});

thread.start();

}

}

```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="15dp"
    android:text="@string/location"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/textView1"
    android:layout_alignParentRight="true"
    android:ems="10" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="68dp"
    android:text="@string/country"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView2"
```

```
        android:layout_marginTop="19dp"
        android:text="@string/temperature"
        android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView3"
    android:layout_below="@+id/textView3"
    android:layout_marginTop="32dp"
    android:text="@string/humidity"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView4"
    android:layout_below="@+id/textView4"
    android:layout_marginTop="21dp"
    android:text="@string/pressure"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView3"
    android:layout_toRightOf="@+id/textView3"
    android:ems="10" >

    <requestFocus />
</EditText>
```

```
<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView3"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignLeft="@+id/editText2"
    android:ems="10" />
```

```
<EditText
    android:id="@+id/editText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView5"
    android:layout_alignLeft="@+id/editText1"
    android:ems="10" />
```

```
<EditText
    android:id="@+id/editText5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView5"
    android:layout_alignBottom="@+id/textView5"
    android:layout_alignRight="@+id/editText4"
    android:ems="10" />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_below="@+id/editText1"
    android:onClick="open"
```

```

        android:text="@string/weather" />

</RelativeLayout>

```

Following is the content of the **res/values/string.xml**.

```

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

    <string name="app_name">XMLParser</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="location">Location</string>
    <string name="country">Country:</string>
    <string name="temperature">Temperature:</string>
    <string name="humidity">Humidity:</string>
    <string name="pressure">Pressure:</string>
    <string name="weather">Weather</string>

</resources>

```

Following is the content of **AndroidManifest.xml** file.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.xmlparser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET"/>


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"

```



```
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.xmlparser.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

Let's try to run our XMLParser 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 following Emulator window:



Now you need to enter any location in the location field. For example, we have entered London. Press the weather button, when you enter the location. The following screen would appear in you AVD:



Now when you press the weather button, the application will contact the Google Weather API and will request for your necessary XML location file and will parse it. In case of London following file would be returned:



```

<?xml version="1.0" encoding="UTF-8"?>
- <current>
  - <city name="London" id="2643743">
    <coord lat="51.50853" lon="-0.12574"/>
    <country>GB</country>
    <sun set="2013-10-08T17:21:45" rise="2013-10-08T06:13:56"/>
  </city>
  <temperature unit="kelvin" max="292.15" min="289.15" value="290.84"/>
  <humidity unit="%" value="68"/>
  <pressure unit="hPa" value="1025"/>
  - <wind>
    <speed name="Gentle Breeze" value="5.1"/>
    <direction name="West-southwest" value="250" code="WSW"/>
  </wind>
  <clouds name="few clouds" value="20"/>
  <precipitation mode="no"/>
  <weather value="few clouds" icon="02d" number="801"/>
  <lastupdate value="2013-10-08T10:20:00"/>
</current>

```

Note: This temperature is in kelvin, so if you want to convert it into more understandable format, you have to convert it into Celsius.