Community Experience Distilled

Android Development Tools for Eclipse

Set up, build, and publish Android projects quickly using Android Development Tools for Eclipse

Sanjay Shah
Khirulnizam Abd Rahman

[PACKT] PUBLISHING open source* community experience distilled

Android Development Tools for Eclipse

Set up, build, and publish Android projects quickly using Android Development Tools for Eclipse

Sanjay Shah

Khirulnizam Abd Rahman

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Android Development Tools for Eclipse

First published: July 2013

Production Reference: 1200713

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK..

ISBN 978-1-78216-110-3

www.packtpub.com

Cover Image by J.Blaminsky (milak6@wp.pl)

Credits

Authors

Sanjay Shah

Khirulnizam Abd Rahman

Reviewer

Thomas Iguchi

Acquisition Editor

Wilson D'souza

Commissioning Editors

Sharvari Tawde

Ameya Sawant

Technical Editors

Ruchita Bhansali

Shashank Desai

Larissa Pinto

Project Coordinator

Amey Sawant

Proofreader

Linda Morris

Indexer

Rekha Nair

Production Coordinator

Nilesh R. Mohite

Cover Work

Nilesh R. Mohite

About the Authors

Sanjay Shah has worked on diverse areas of application development across the mobile and web platform with more than 8 years of experience. He is currently working as a Software Architect and works in the area of Cloud Based Big Data Analytics combined with Distributed Cognition leveraging various Java-based technologies. He is fond of philosophy and enjoys life in Nepal, the land of the highest peak in the world, Mt. Everest.

I would like to thank each and every one who knows me and supported me at different aspects of my life. Special thanks to my parents without whom I wouldn't be what I am today.

Khirulnizam Abd Rahman is a Computer Science lecturer in the faculty of Information Science and Technology, Selangor International Islamic College, Malaysia. He has been teaching programming since the year 2000.

He started publishing Android apps in the year 2010, and his apps among others are Malay Proverb Dictionary (Peribahasa) and m-Mathurat. Currently, he is working on the apps for Windows Phone Version 8. PHP, C#, and Java are also the programming languages that he is familiar with.

I would like to express my deepest gratitude to my beloved family; Mahani, Luqman, Muna, and Amir for making my life more colorful. Because of you, I am a grown up person with a heart full of love. In fact, as long as they are happy, I will be happy. Other than programming, teaching, writing and being with my family, I don't have anything else to do.

About the Reviewer

Thomas Iguchi is the founder of Nobu Games LLC, a video games and mobile app development company, in La Crosse, Wisconsin. His latest Android game "Zoolicious" has gained international recognition and awards from various Android news and review websites such as AndroidTapp and Famigo.

Thomas has a wide repertoire of skills, which include programming, graphic designing, and music composition. His interest in computers and programming dates back to his childhood, when he became a self-taught programmer. He later went on to deepen his theoretical knowledge by studying Computer Science at the University of Mainz, Germany, with focus on linguistics, model-driven architecture, and software engineering. Overall he looks back at over 20 years of programming experience with professional expertise in web, mobile applications, and video game development. His consulting work for a mobile app development company, which serviced the financial industry, in Frankfurt, Germany, allowed him to perfect his professional Android programming skills.

For the last 12 years, Thomas has been self-employed working as a web designer, programmer and consultant, as well as video game graphic designer for the coin-op entertainment industry.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Android Development Tools for Eclipse will show you how to use ADT (Android Development Tools) for Eclipse to quickly set up Android projects, create application UI, debug and export a signed (or unsigned) .apk package for distribution using a hands-on practical approach. The book starts with the installation of ADT, discusses important tools and guides you through Android application development from scratch, demonstrating different concepts and implementation, and finally helps you distribute it.

What this book covers

Chapter 1, *Installing Eclipse, ADT, and SDK*, guides you through the installation of Eclipse and ADT(Android Development Tools) needed for Android application development.

Chapter 2, *Important Features of the IDE*, describes several important features in Eclipse and an ADT Environment useful to develop native Android apps.

Chapter 3, *Creating a New Android Project*, guides you through the creation of a new project and demonstrates the usage of simple widgets. It also guides across compiling, debugging, and running the application.

Chapter 4, *Incorporating Multimedia Elements*, will teach you how to include multimedia elements and handle multiple screens in the application.

Chapter 5, *Adding RadioButton, CheckBox, Menu, and Preferences*, deals with adding menus and Preference Screen and the usage of radio button and check box.

Chapter 6, *Handling Multiple Screen Types*, teaches you how to tackle different screen types and orientations.

Chapter 7, *Adding External Library*, guides you through adding external library, that is, the AdMob library and incorporating advertisements in the application.

Chapter 8, *Signing and Distributing APK*, shows the steps involved in signing and distributing the Android application.

What you need for this book

It is advisable to have a laptop or a PC with the following specifications for better performance during development:

- 4 GB RAM
- Window 7 OS
- Dual Core /i-Series processor

Who this book is for

Android Development Tools for Eclipse is aimed at beginners and existing developers who want to learn more about Android development. It is assumed that you have experience in Java programming and you have used IDE for development.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input or output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Installing Eclipse, ADT, and SDK

This chapter serves as an installation instruction for all the development toolkits required to develop Android on Windows environment. It is separated into the following subtopics:

- Brief introduction to the Android platform
- Installing the Java Development Kit (JDK)
- Installing the Android SDK
- Installing the Eclipse (Juno)
- Installing the Android Development Toolkits (ADT) in Eclipse (Juno)
- Linking the Android SDK to the Eclipse

Before we proceed with the installation guide, there is some basic information an Android developer must know.

Introducing the Android platform

In simple terms, Android is a Linux based operating system for touch screen devices developed by Android Inc., financed by Google and was bought in later 2005. The beta version of Android came back in November 2007 and the commercial version 1.0 was released in September 2008. As of 2013, over 500 million active devices use the Android OS worldwide.

What is Android?

Android is a software stack for mobile devices that includes an operating system, middleware and key applications (platform). The Android **Software Development Kit** (**SDK**) provides the tools and **Application Programming Interfaces** (**API**s) necessary to begin developing applications on the Android platform using the Java programming language. The kernel of Android is Linux.

Introducing the Android app

A mobile software application that runs on Android is an Android app. The apps use the extension of `.apk` as the installer file extension. There are several popular examples of mobile apps such as Foursquare, Angry Birds, Fruit Ninja, and so on.

Primarily in an Eclipse environment, we use Java, which is then compiled into Dalvik bytecode (not the ordinary Java bytecode). Android provides **Dalvik virtual machine** (**DVM**) inside Android (not Java virtual machine JVM). Dalvik VM does not ally with Java SE and Java ME libraries and is built on Apache Harmony java implementation.

What is Dalvik Virtual?

Dalvik VM is a register-based architecture, authored by *Dan Bornstein*. It is being optimized for low memory requirements and the virtual machine was slimmed down to use less space and less power consumption.

Understanding API level

API level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

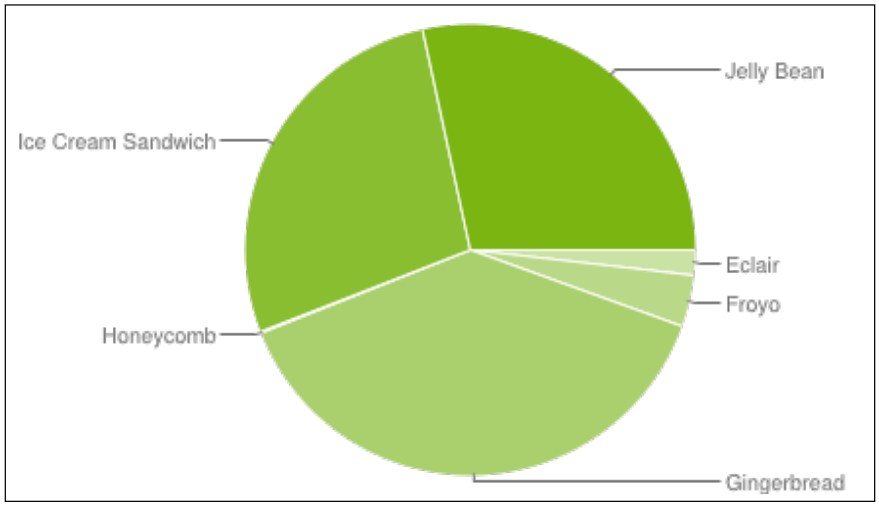
The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:

- A core set of packages and classes
- A set of XML elements and attributes for declaring a manifest file
- A set of XML elements and attributes for declaring and accessing resources
- A set of Intents
- A set of permissions that applications can request, as well as permission enforcements included in the system

How many versions (distributions) Android has?

The latest distribution statistics until May 1, 2013, are shown in the following screenshot. It indicates that Android 2.3.3 has the largest market share; however, Android 4.1.x is gaining momentum and will have the dominant share. It is important to know that if the app is primarily targeted to an Android version, it will not run on the previous version of Android.

For instance, if you are developing an app for Android 2.2 (API level 8), then the application will not run on Android 2.1 (API level 7) and below. However, the app is compatible for Android 2.2 and later.



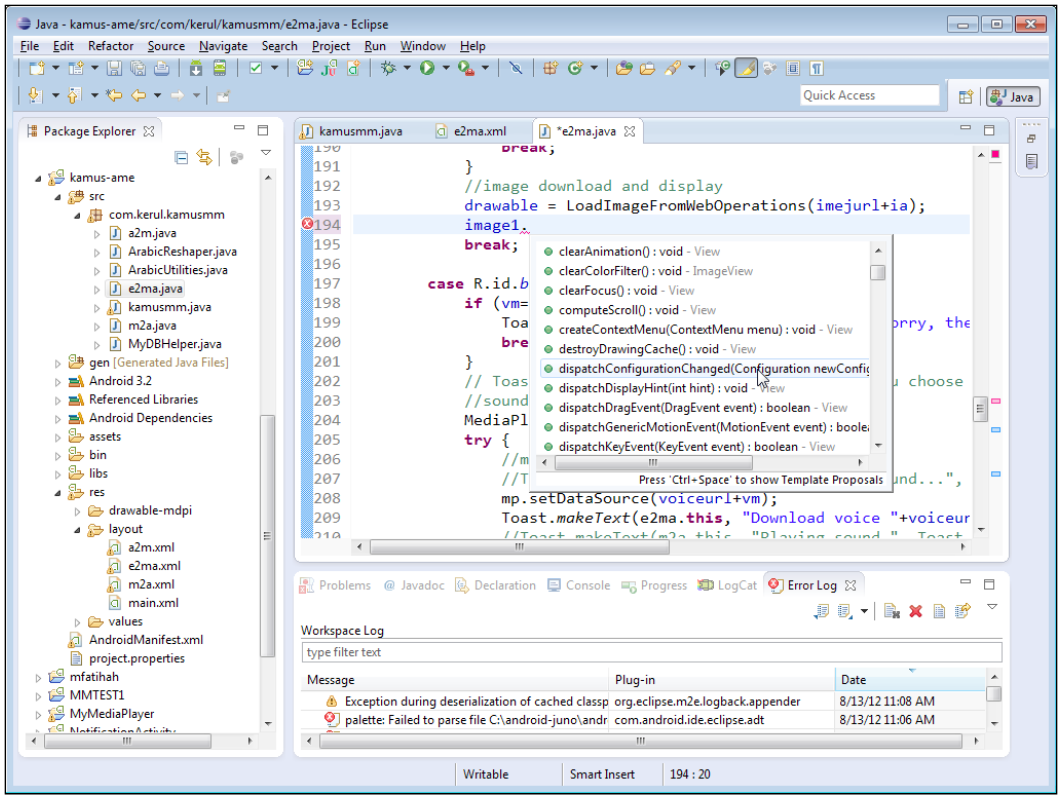
Pie chart of the Android API level distribution
(Source: http://developer.android.com/about/dashboards/index.html)

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	3.7%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	38.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	27.5%
4.1.x	Jelly Bean	16	26.1%
4.2.x		17	2.3%

The Android API level distribution
(Source: http://developer.android.com/about/dashboards/index.html)

Preparing for Android development

In this part of the chapter, we will see how to install the development environment for Android on the Eclipse Juno (4.2). Eclipse is the major IDE for Android development (see the following screenshot). We need to install eclipse extension ADT (Android Development Toolkit) for development of the Android Application:



ADT on Eclipse in action

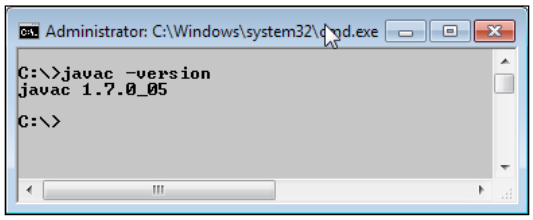
To download Android packages a Google API internet connection is a must, hence take this in notice before moving further. The steps on Windows using Eclipse Juno are as follows:

Software needed:

- Latest JDK1.6.x from Oracle
- Latest Android SDK
- Eclipse 4.2 (Juno)

Installing the JDK

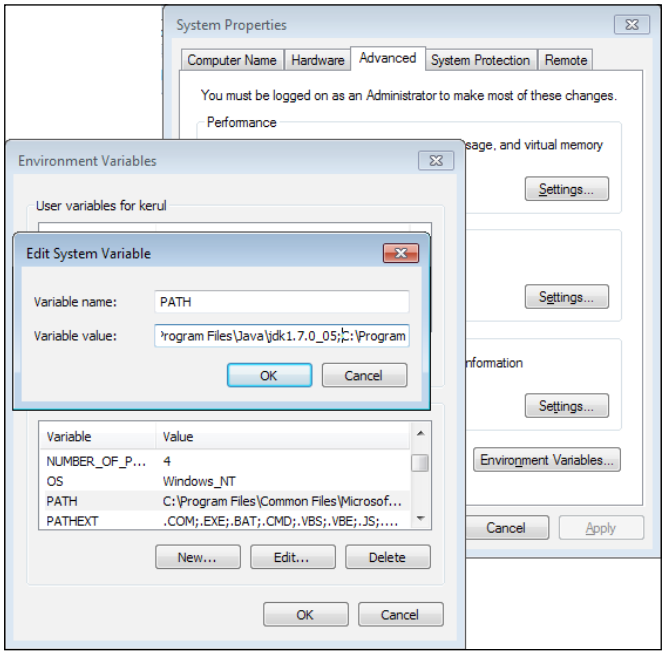
To check whether your PC has an existing JDK and it is installed correctly, go to command prompt, and type `javac -version` (as shown in the following screenshot). It is recommended to install JDK 1.6.x for Android Application Development as it may complain that the compiler compliance level is greater than 6, and could run into problems:



Checking the JDK version

You may download JDK 1.6 (Java Development Toolkit) from the download site and install it. Make sure that JAVA_HOME is set after the installation, and check the version executing the preceding command.`http://www.oracle.com/technetwork/java/javase/downloads/index.html` (see the following screenshot).

This step can be skipped if we have java 1.6.x installed:

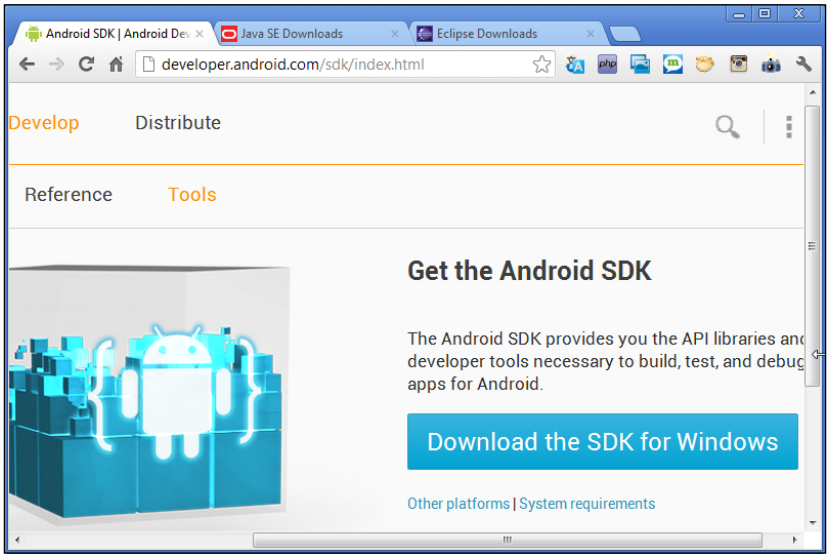


Java PATH setting

Installing the Android SDK

Create a folder named `android-dev` (android-dev is just a suggestion; you may create another name instead). The folder `android-dev` will be used consistently throughout this chapter. This folder is to hold all the software that is needed for Android development. This folder is needed again in another procedure.

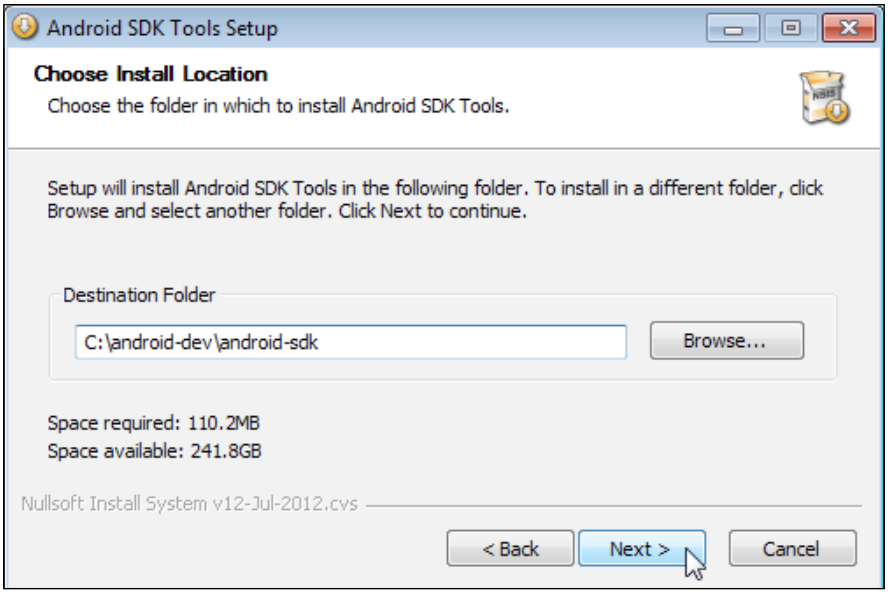
Download from `http://developer.android.com/sdk/index.html`, and install this software in the `android-dev` folder. Bear in mind this download only provides the basic tools of Android SDK, not the complete installation. Later, we need to download the Android system images, APIs, examples, documentations and other libraries:



Android SDK download page

After completion of the download, install the SDK in the folder mentioned earlier; in `C:\android-dev\android-sdk` as shown in following screenshot.

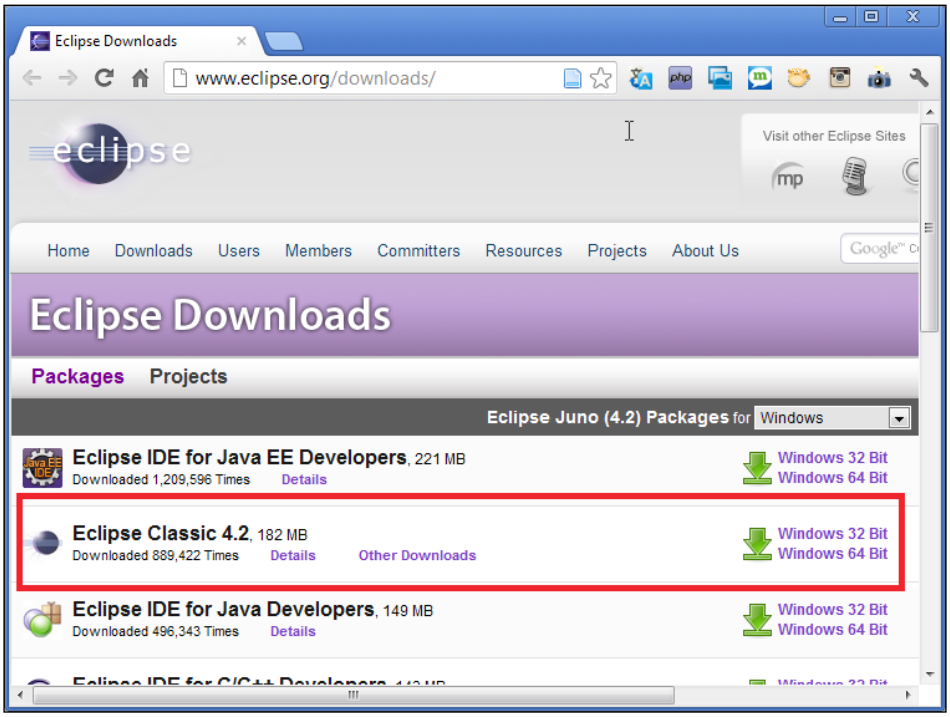
During the installation, the Android SDK will detect the Java Development Kit in the machine. If we have installed the latest JDK, it should have no problems:



Android SDK installation path

Installing the Eclipse (Juno)

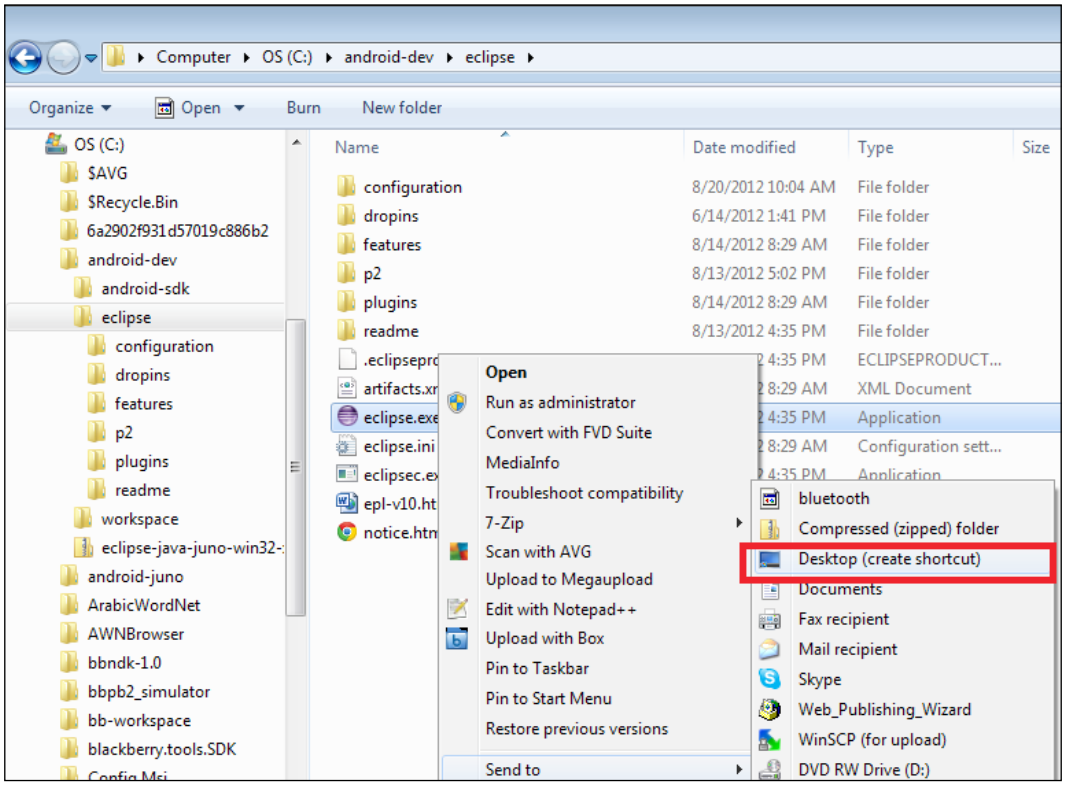
Eclipse Juno (4.2) is available for download at `http://www.eclipse.org/downloads/`:



Download page of Eclipse Classic

The Eclipse comes in a ZIP file, so just unzip it and find the `eclipse.exe` file to run it.

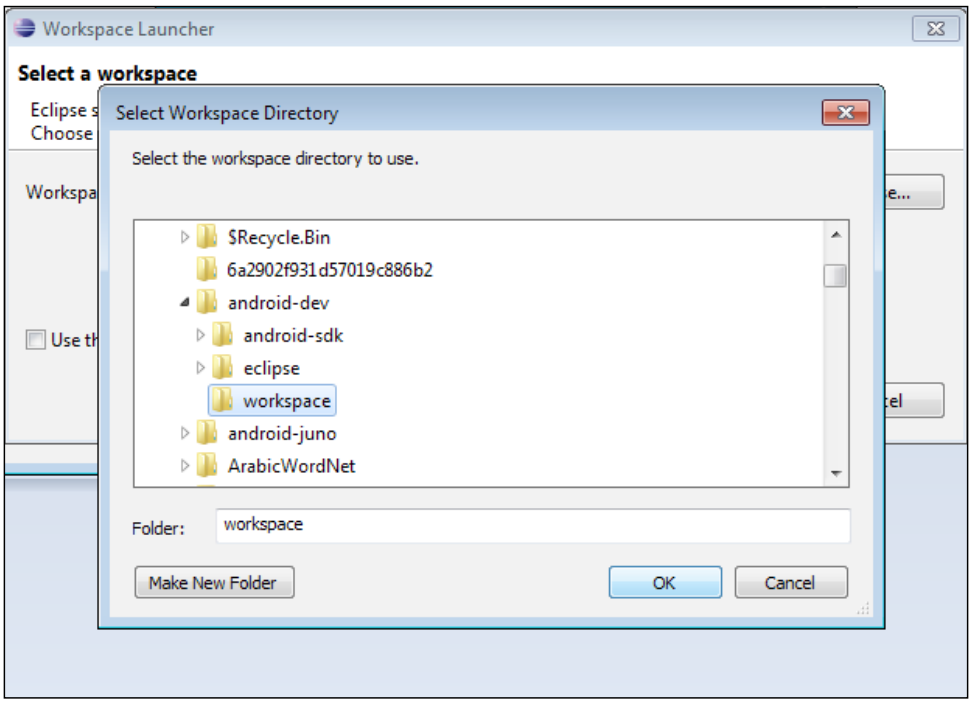
Immediately extract Eclipse in the folder as created earlier (in `C:\android-dev`). After the extraction, create a desktop shortcut to make life easier, as depicted in the following screenshot:



Create Eclipse shortcut

Installing the ADT in Eclipse Juno

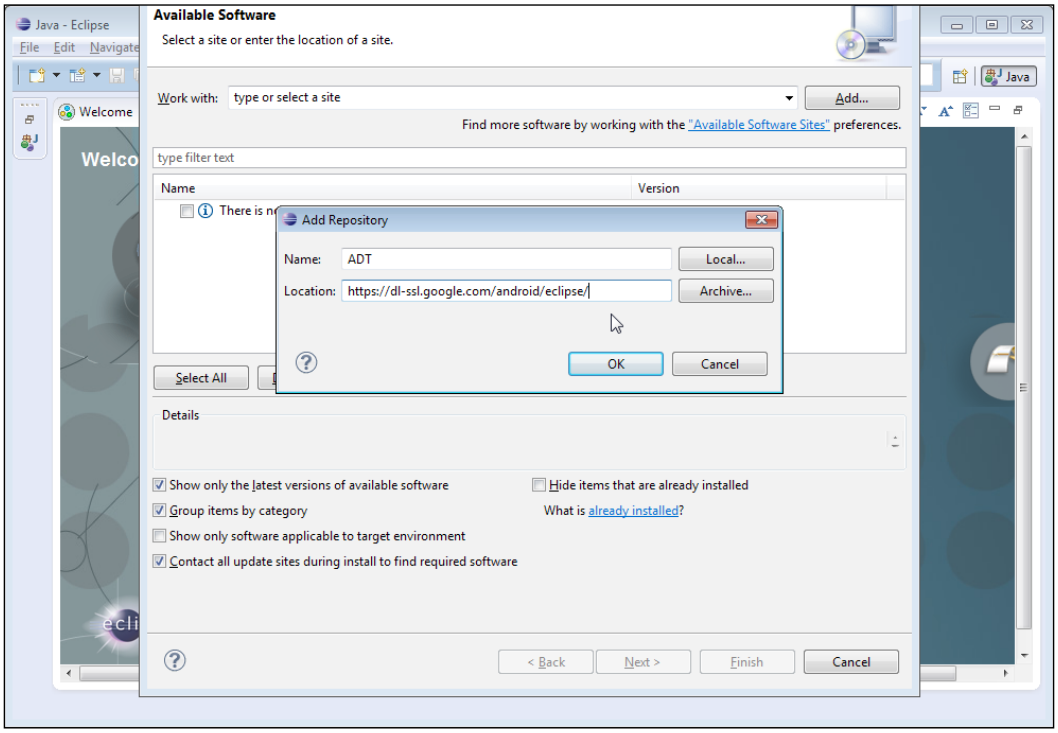
Run Eclipse by identifying the Eclipse installation folder and double-click `eclipse.exe` (or double-click the shortcut in the **Desktop**). Provide a folder to store all the projects' source codes. And once again, create this folder under the `android-dev` folder, as shown in the following screenshot:



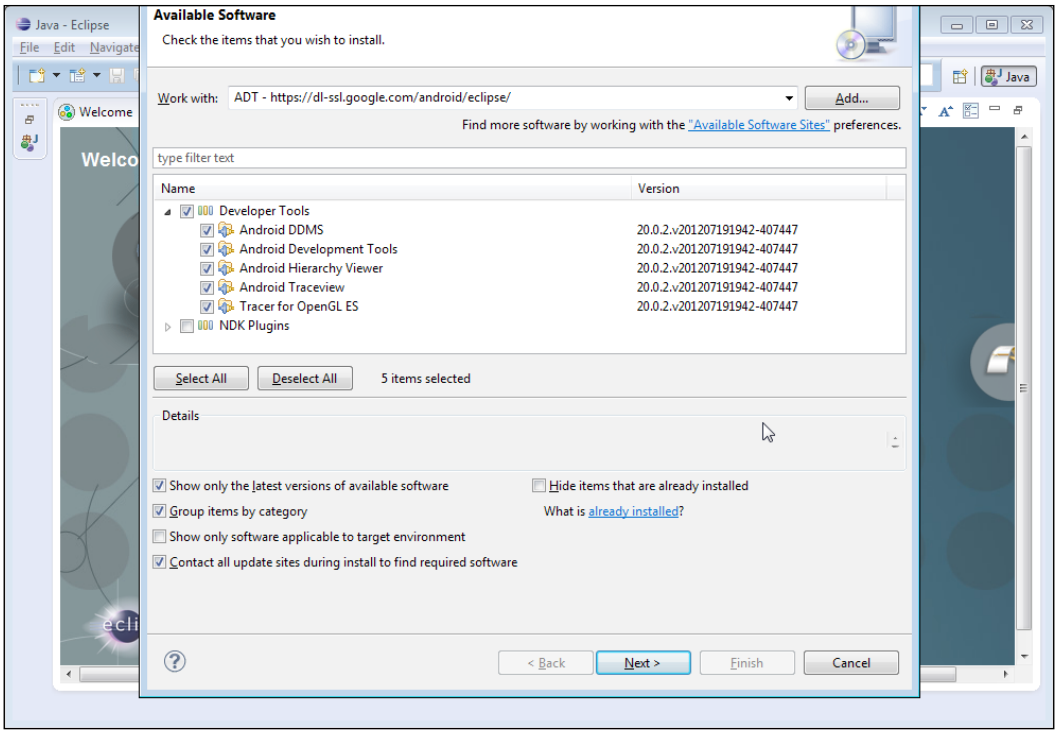
Select Eclipse Workspace

This new Eclipse installation does not provide the Android Developer Toolkits (ADT) plugins. To install this plugin navigate to **Window** | **Preferences** to open the Preferences panel. Click on **Install/Update** | **Available Software Sites** (on the left panel). Click on the **Add** button (on the right panel) to add a software download site (again an Internet connection is needed).

Another window will appear. Provide `ADT` in the **Name** (for example), and the **Location** `https://dl-ssl.google.com/android/eclipse/`(as provided in `http://developer.android.com/sdk/eclipse-adt.html`):



In the **Available Software** dialog, select the checkbox next to **Developer Tools** and click on **Next**. In the next window, you'll see a list of the tools to be downloaded. Select all except **NDK plugins** and click on **Next**. We will be discussing the tools in the next chapters:

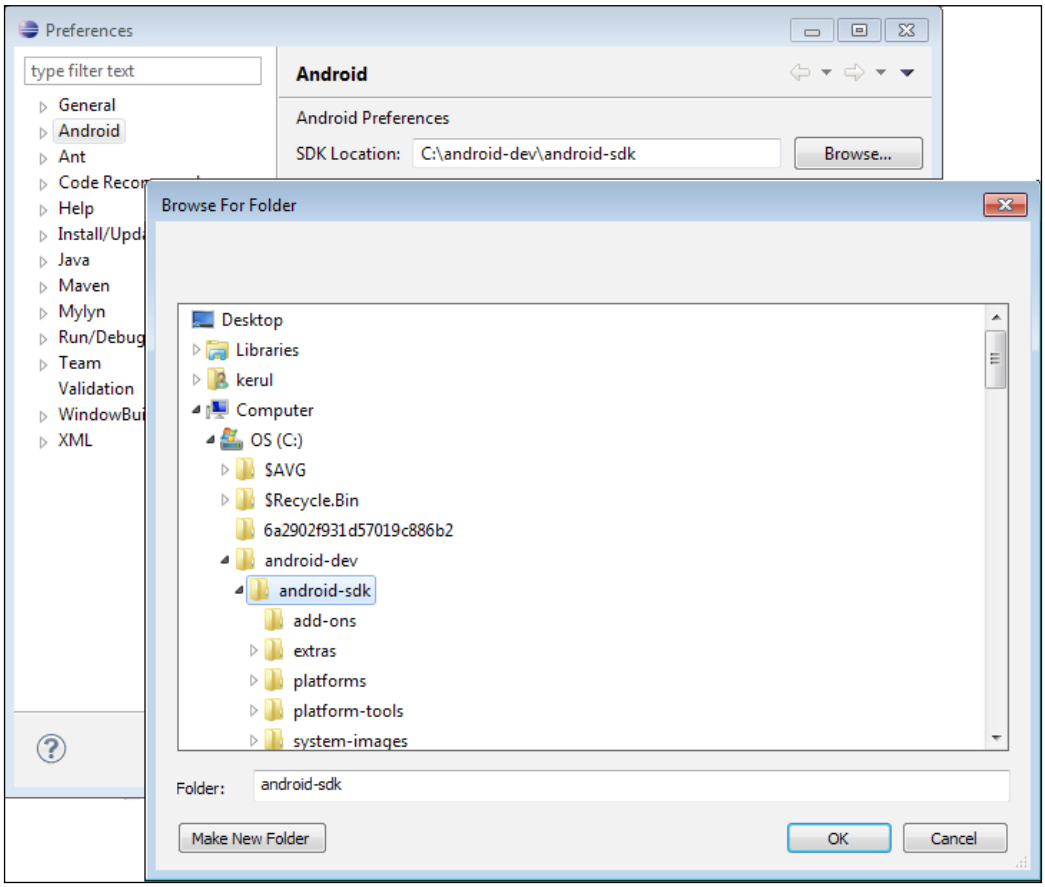


Selecting the ADT and SDK tools

Read and accept the license agreements, then click on **Finish**. If you get a security warning saying that the authenticity or validity of the software can't be established, click on **OK**. When the installation completes, restart **Eclipse**.

Linking the Android SDK to the Eclipse

Run Eclipse. In the **Windows** | **Preferences**, click on **Android**. Locate the folder of the `android-sdk` from the step where you installed the android-sdk, as shown in the following screenshot:

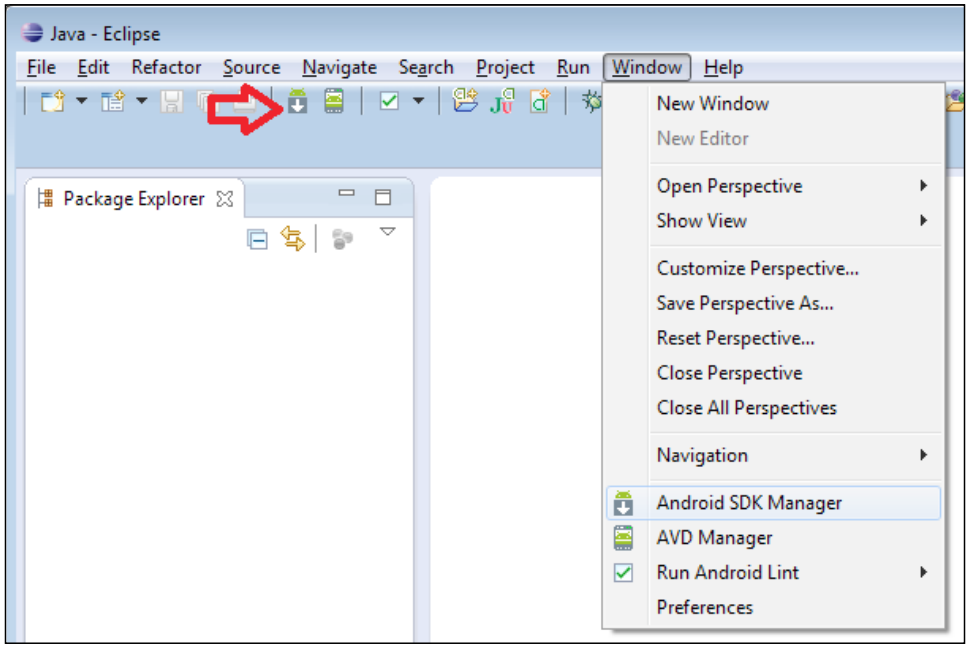


Android preferences in Eclipse

Click on **Apply** and hit **OK**.

The next thing to do is to download the Android APIs and the operating system images. Installing Android SDK is time consuming. It requires a smooth broadband line because after the installation you need to download the API package for Android and Google API.

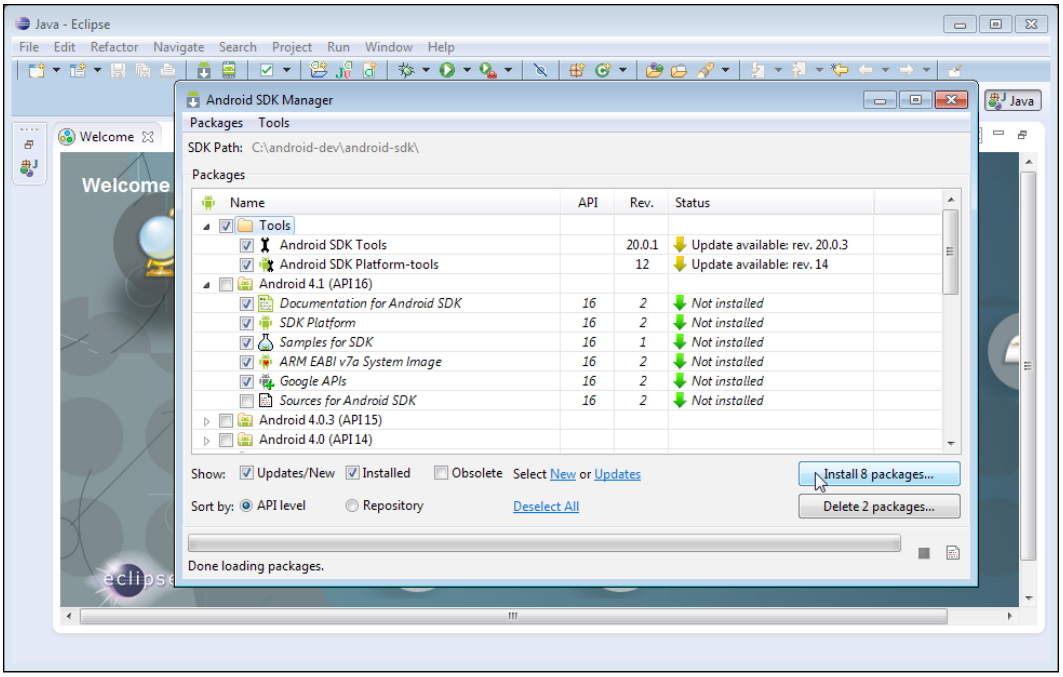
To start this, click on the **Android SDK Manager** icon, as shown in the following screenshot:



The Android SDK Manager icon

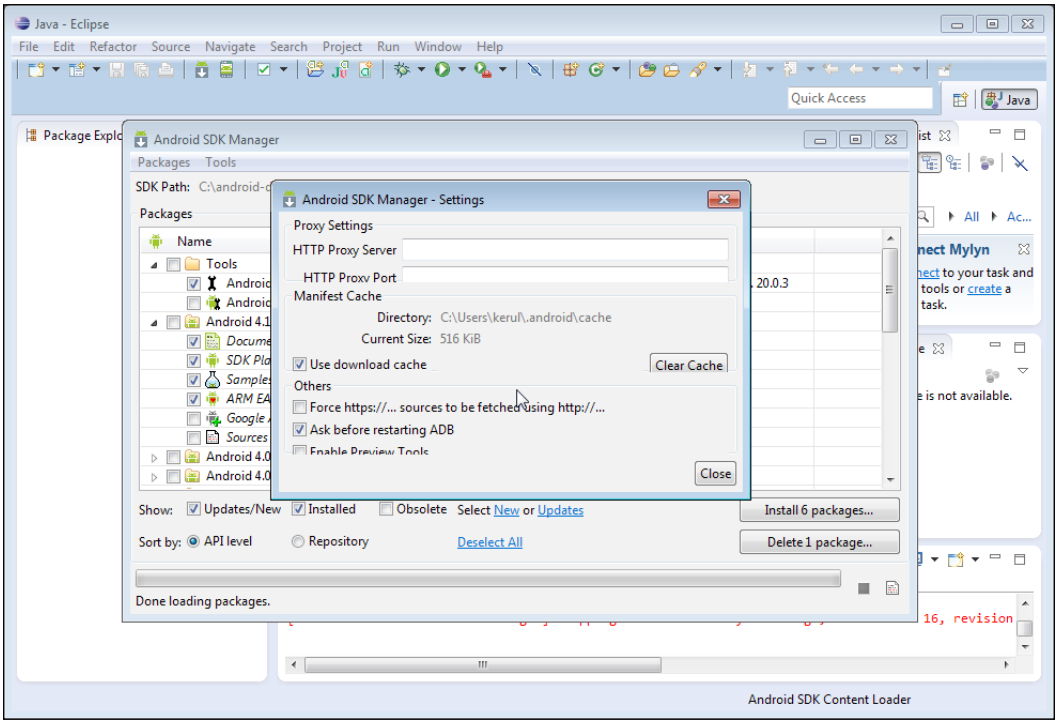
You will be provided with the list of all SDK Platforms for all Android versions. I suggest you be selective, just download your target platform first. If you are to develop an app for **Froyo** (Android 2.2) you need to download the API version 8. Later, when you have more time, you could come back and download for the other version. If you do not have any time and Internet data constraints then you may download all. It will fetch API packages, Android OS images, debugging tools and other softwares related to Android development.

For this time, we will download the latest SDK with **Jellybean** system image and **API level** 16, as shown in the following screenshot:

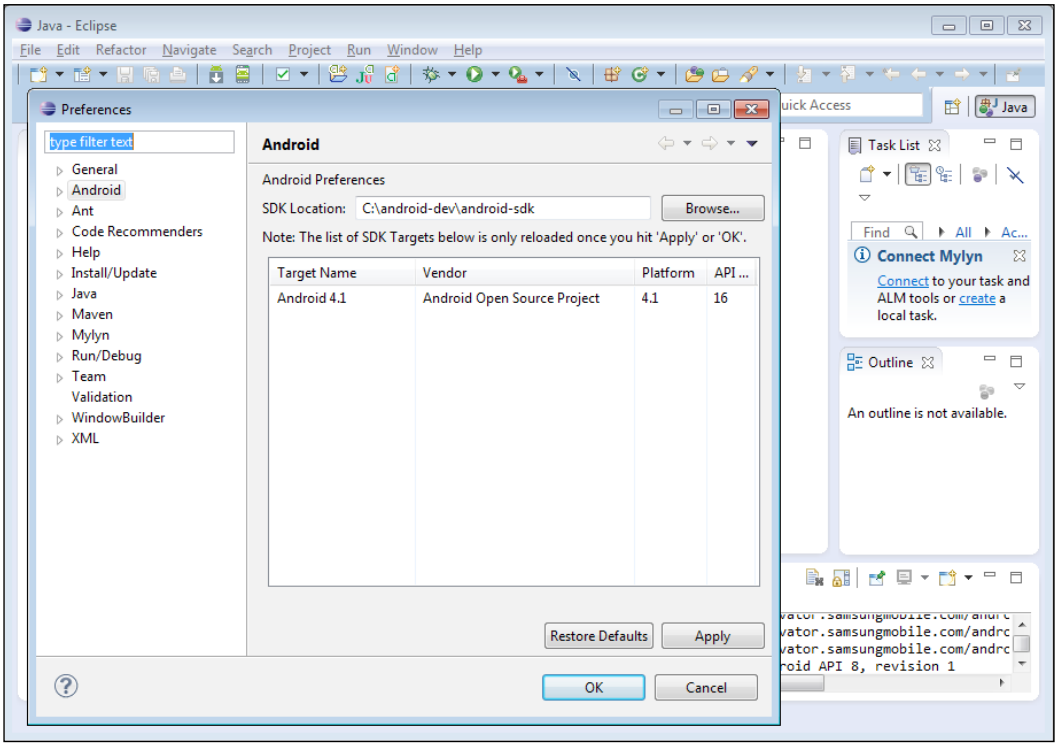


Installing SDK with API level 16

Before hitting the **Install** button, there is one important tip I'd like to share. While conducting this procedure, we may encounter a connection reset problem for no specific reason. To get over this issue, on the **Android SDK Manager** window, navigate to **Tools | Options**. Uncheck the Force **https://...sources to be fetched using https://...** option, and **Close** (shown in the following screenshot). You may start the SDK and API installation now:



After the SDK, APIs and system images have been downloaded, restart Eclipse. The wait is worth it! After almost a couple of hours of installation and downloading packages, I got this nice graphical interface for the screen layout arrangement, as shown in the following screenshot. Check the **Android Preferences** window, and you may see the **Android 4.1** in the API list. To add another API, again you need to download through the Android SDK Manager:



List of Android APIs

To avoid earlier steps on setting up ADT with Eclipse and kick start development please download the ADT bundle from `http://developer.android.com/sdk/index.html` and follow the steps for setting up at `http://developer.android.com/sdk/installing/bundle.html`.

In the next chapter, we will look into tools of an ADT environment that eases the development.

Summary

In this chapter, we learnt how to install the Eclipse Juno (the IDE), the Android SDK and the testing platform. The next chapter will discuss the important elements of the IDE before we create a new Android Application project.

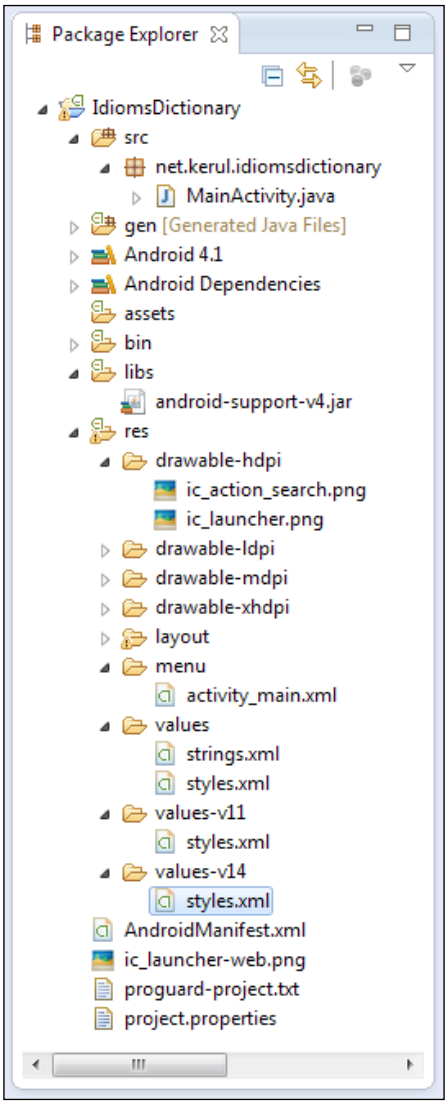
2
Important Features of the IDE

This chapter describes several important features in Eclipse and an ADT Environment useful to develop an Android app. It is separated into the following topics:

- Project explorer
- Code editor
- Graphical user interface designer
- Properties window
- Debugging pane
- Dalvik Debug Monitor Server (DDMS)
- SDK manager
- Android virtual device manager
- Running an application
- Getting help

Project explorer

The project explorer is a tool to view all folders and files under a project. By double-clicking the item, one can open and edit the file. When we create a new project, which will be discussed thoroughly in *Chapter 3,Creating a New Project*, the ADT will automatically create all these default folders and files, as shown in the following screenshot. Depending on the project, we may ignore or modify all these files.
These are brief descriptions of the default folders and files in your Android project:



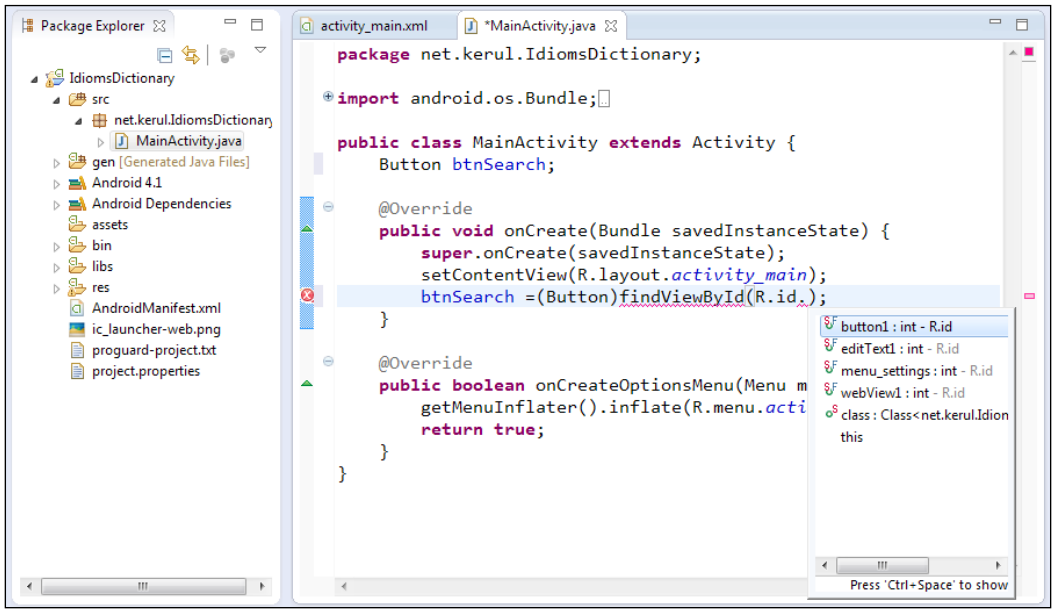
Project explorer

The table that follows contains the brief description of the important folders and files available in the project tree:

Folder	Functions
`/src`	the Java codes are here
`/gen`	generated automatically
`/assets`	put your fonts, videos, sounds here. Is more like a file system and can also place css, javascript files and so on.
`/libs`	external library (normally in JAR)
`/res`	images, layout, and global variables
`/drawable-xhdpi`	for extra high specification devices (for examples Tablet, Galaxy SIII, HTC One X)
`/drawable-hdpi`	for high specification phones (Examples: SGSI, SGSII)
`/drawable-mdpi`	for medium specification phones (Examples: Galaxy W, HTC Desire)
`/drawable-ldpi`	for low specification phones (Examples: Galaxy Y, HTC WildFire)
`/layout`	all XML files for the screen(s) layout
`/menu`	XML files for the screen menu
`/values`	global constants
`/values-v11`	template style definitions for devices with Honeycomb (Android API level 11)
`/values-v14`	template style definitions for devices with ICS (Android API level 14)
`AndroidManifest.xml`	One of the important files to define the apps. This is the first file located by the Android OS in order to run the app. It contains the app's properties, activity declarations and list of permissions.

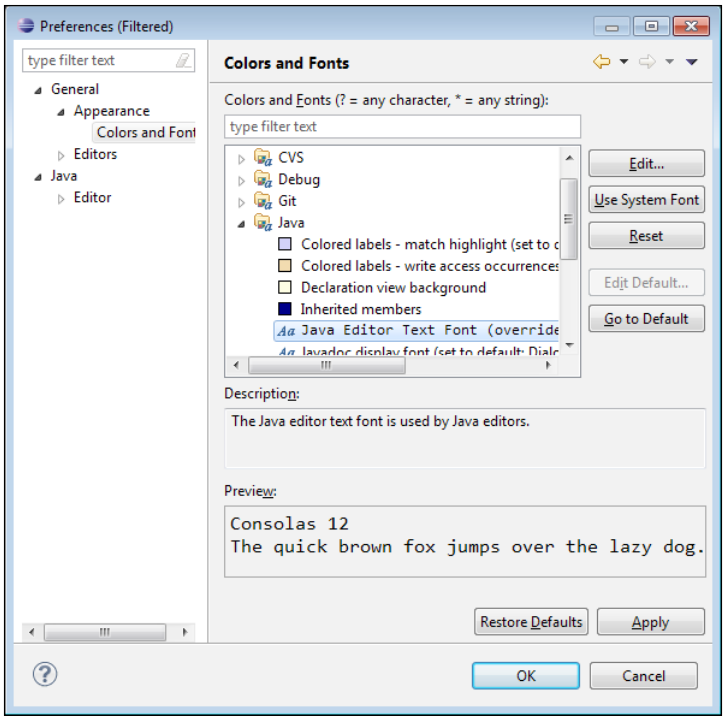
Code editor

This is the tool where the programming is cooked. Several important features of the Eclipse code editor (programmers love to have) are intelligence and the error marker (refer screenshot to follow). Code completion suggests objects, methods or variables available to be incorporated in our code, while the error marker will notify any syntax error immediately without having to compile the code. These features help a lot for faster programming:



The code editor

The code editor's appearance is customizable to suit your style and preference. To change the editor's environment, such as the background color or the code's font styles, right-click on the editor and choose **Preferences**, and then navigate to **General | Appearance | Colors and Fonts**. Then click on **Edit** to customize, refer the following screenshot:



Customizing the Code Editor's appearance

There are also several other XML code editors that help during design and development. They come in two flavors: GUI based; where things can be manipulated with a GUI interface, useful for someone who is uncomfortable editing the XML code manually; Source based: where XML codes can be manually edited. Some of the editors are listed as follows:

Graphical layout editor

Edit and design your XML layout files with a drag and drop interface. The layout editor renders your interface as well, offering you a preview as you design your layouts.

Android manifest editor

Edit Android manifests with a simple graphical interface. This editor is invoked when you open an `AndroidManifest.xml` file.

Menu editor

Edit menu groups and items with a simple graphical interface. This editor is invoked when you open an XML file with a `<menu>` declared (usually located in the `res/menu` folder).

Resources editor

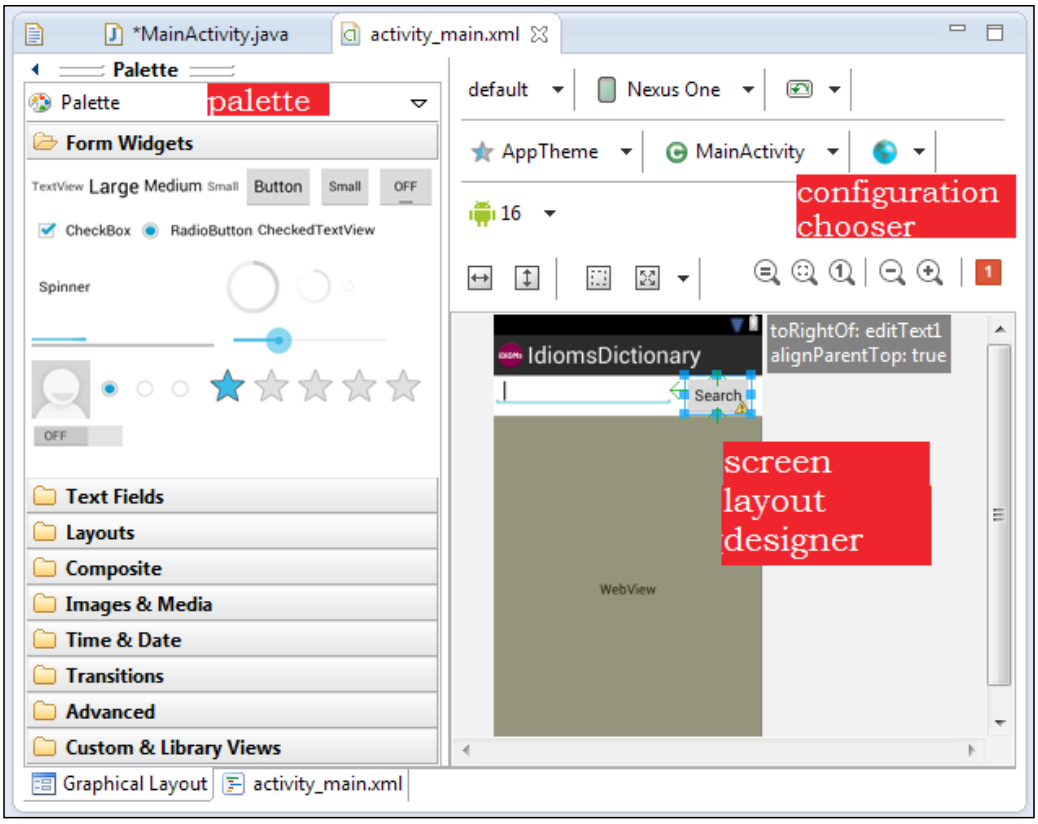
Edit resources with a simple graphical interface. This editor is invoked when you open an XML file with a `<resources>` tag declared.

XML resources editor

Edit XML resources with a simple graphical interface. This editor is invoked when you open an XML file.

Graphical user interface designer

This is the interface designer. It functions as the GUI editor for controls or a **widget** to the application screen. There are three sections of this GUI designer, the palette, configuration chooser and the screen layout preview, as shown in the following screenshot:



The ADT's GUI designer

The **Palette** contains all the GUI controls (widgets) that can help us design the interface. The available controls depend upon the API level we choose during creation of the project. Some of the common controls are: button, text field, radio button, check box, multimedia controls and so on.

The configuration chooser

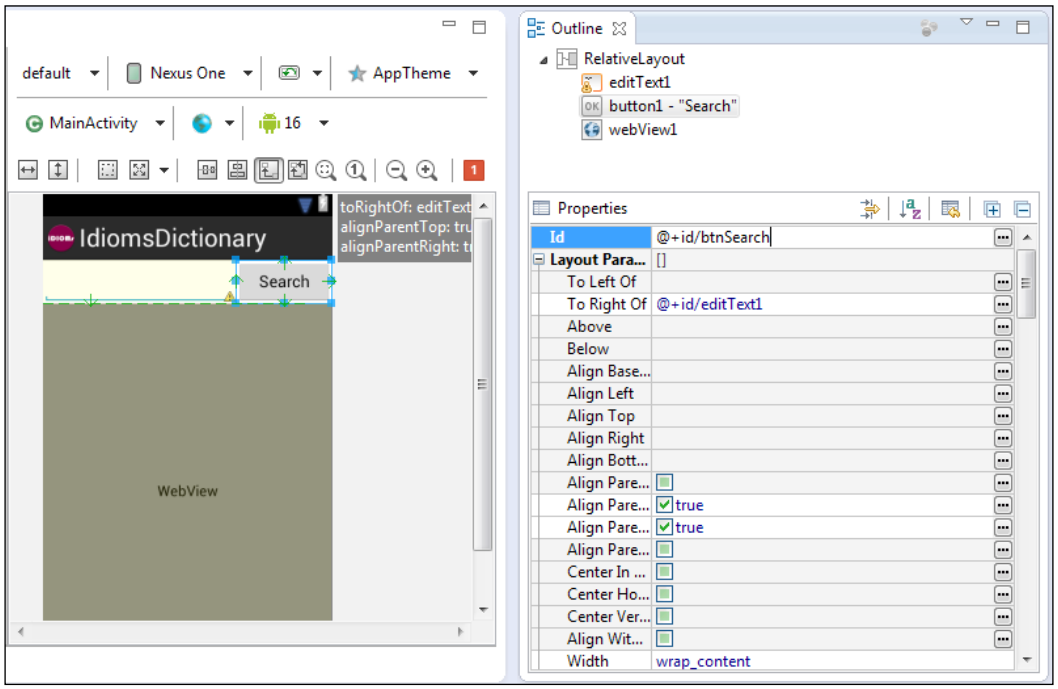
It lets you decide the appearance of your app view across different screen sizes, orientation, densities and themes.

The screen layout designer

It is a canvas to put things up and try out different designs. It is a designing workspace. Also, it provides a preview of how the screen may appear in a device.

Properties window

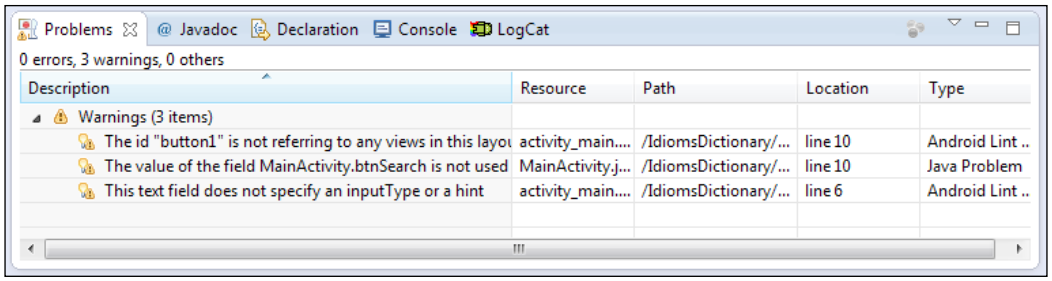
It helps in editing properties of the widgets. All the properties corresponding to widgets can be viewed and edited via this window visually. Though the properties can be edited directly by editing the XML file, this GUI interface eases it. All the changes made are persisted to XML file instantly and automatically. The following screenshot shows the **Properties** window:



The Properties window

Debugging pane

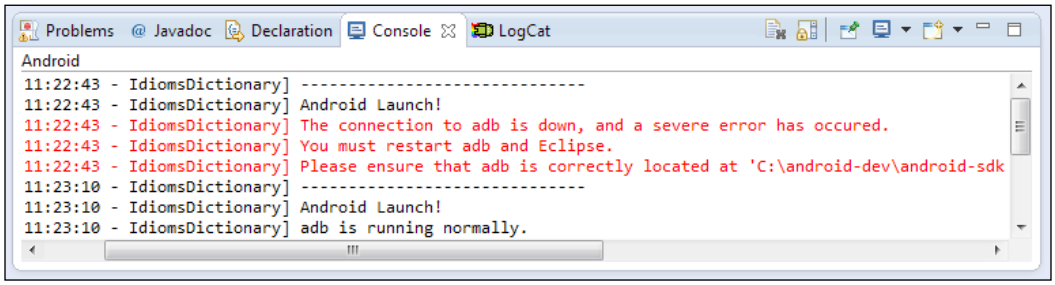
In the debugging perspective, we see the syntax errors, warning, console messages, run-time errors, variable transition (if breakpoint is used) and **LogCat. LogCat** is useful to trace any activity happening inside the device or emulator. The following screenshot shows the window to list all code problems, such as warnings or syntax errors:



Problems warnings or code syntax errors

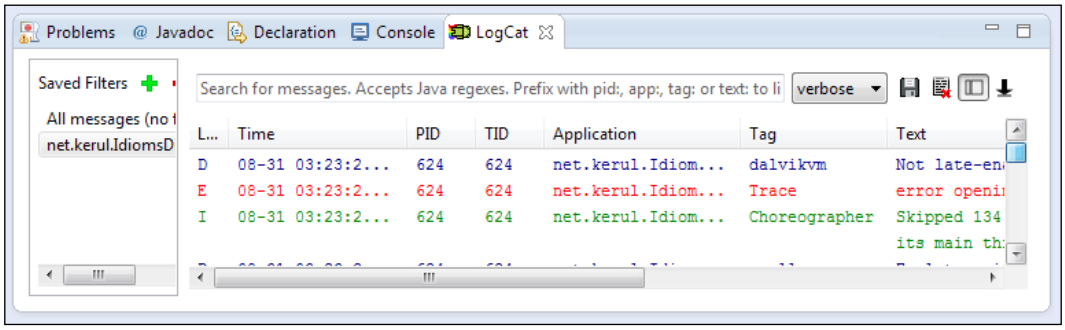
A sample of console messages from the ADB is listed in the following screenshot. As a java person, we would be tempted to use `System.out.println()` to split out message and objects' values; which are shown in the **LogCat** view, however it is advisable to use `Log` class for this purpose, reason being we can filter, print different colors and define log types. This could be one way of debugging your program, by displaying variables' values or parameters. To use `Log`, import `android.util.Log`, and use one of the following methods to print messages to **LogCat**:

```
v(String, String) (verbose)

d(String, String) (debug)

i(String, String) (information)

w(String, String) (warning)

e(String, String) (error)
```



The Android Debug Bridge console (displays ADB activities)

LogCat is used to view the internal log of the Android system, as shown in the following screenshot. It is useful to trace any activity happening inside the device or emulator through the ADB (Android Debug Bridge). ADB is a tool to connect your PC with the virtual device or actual device. Without it, the developer cannot directly transmit the APK file to an Android device/emulator:



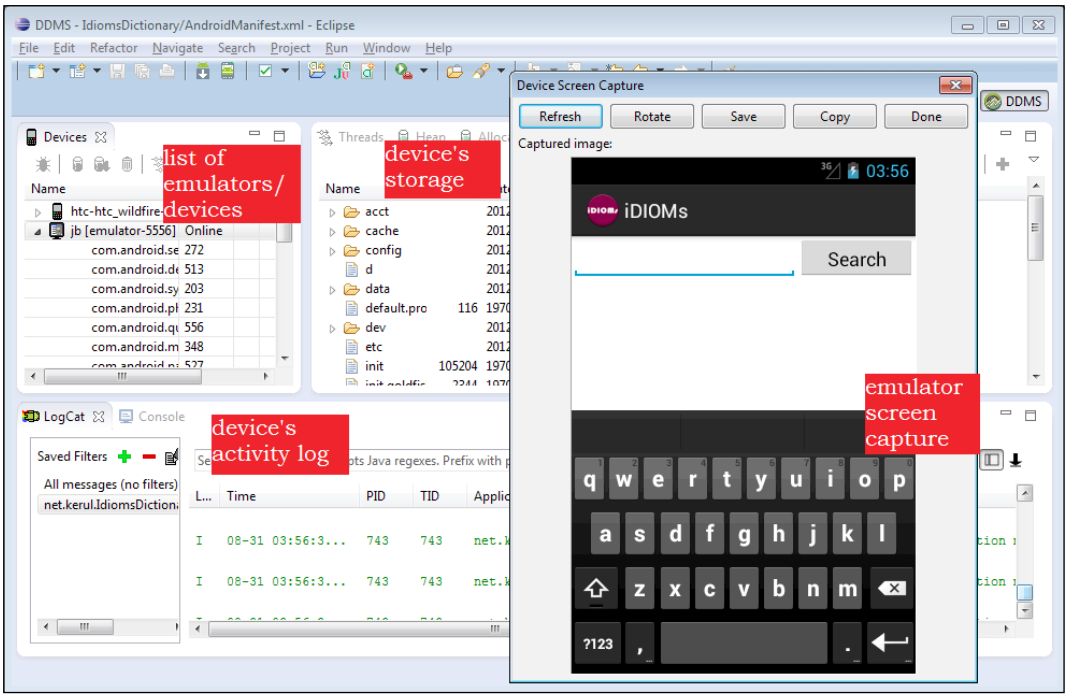
The LogCat (tracedump of all device/emulator activities)

Dalvik Debug Monitor Server (DDMS)

DDMS is a must have tool to view the emulator/device activities. To access DDMS in the Eclipse, navigate to **Windows** | **Open Perspective** | **Other** and then choose **DDMS**. By default it is available in the Android SDK (it's inside the folder `android-sdk/tools` by the file `ddms`). From this perspective the following aspects are available:

- **Devices**: The list of the devices and AVDs that are connected to ADB
- **Emulator Control**: It helps to carry out device functions
- **LogCat**: It views real time system log messages
- **Threads**: It gives an idea of currently running threads within a VM
- **Heap**: It shows heap usage by application
- **Allocation Tracker**: It provides information on memory allocation of objects
- **File Explorer**: It explores the device file system

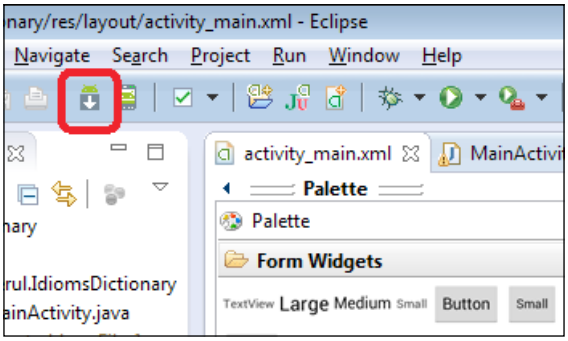
The following image shows important aspects of DDMS:



Dalvik Debug Monitor Server (DDMS)

SDK manager

SDK Manager is the tool to update Android SDK and manage the download of Android OS system images, documentations, and APIs. The icon appears, as shown in the following screenshot:



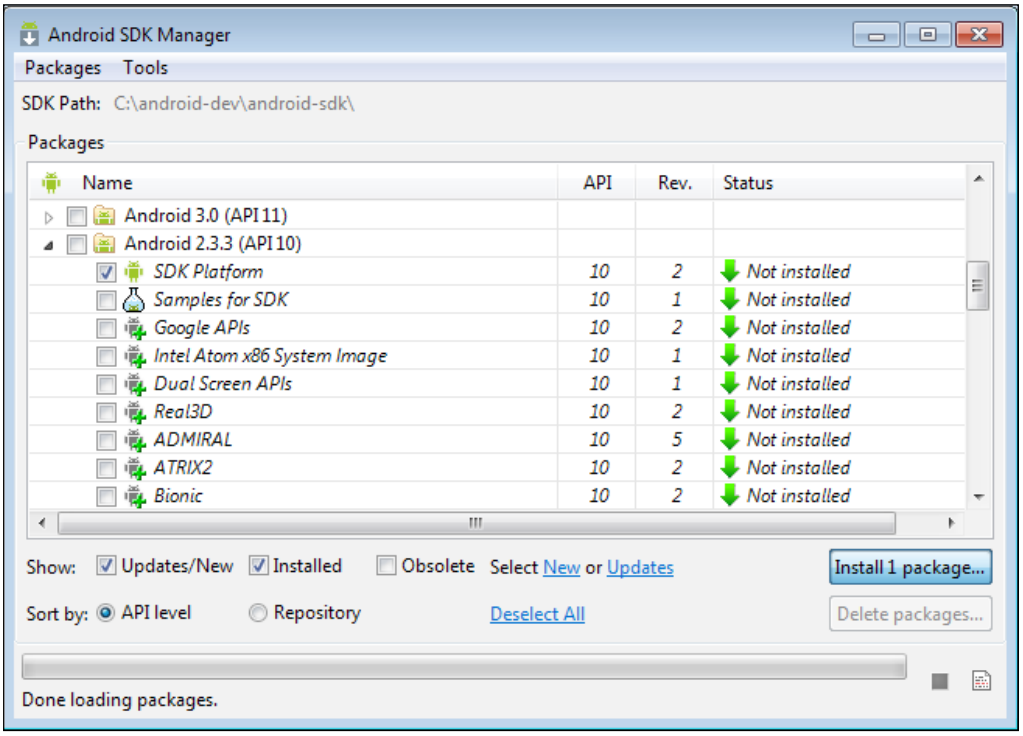
The SDK manager icon

The next screen to follow, as shown in the screenshot has a very long list. We need to be very decisive about what API level we need and select accordingly as the download may take significant time, depending upon the Internet speed. If not sure then choose the latest API level.

Expand the API level we want to issue and check the SDK platform. This download consists of the API for the corresponding level and the Android OS system image. By default, the system image is based on the ARM's architecture. However to run Android OS system image faster on an Intel architecture machine, just tick the **Intel Atom x86 System Image** option.

Tick the **Samples for SDK** if you need to learn from the samples. If your app needs to incorporate the Google special API (such as the Google Maps), then you might need to download the Google API. The rest of the list is about the device specific APIs. Unless you are planning to optimize your app for a certain device, then do not download.

Once you have finished selecting the necessary APIs, then click on the **Install package** button. Should you have any connection reset problem while downloading, navigate to **Tools | Options**. Uncheck the **Force https://...** sources to be fetched using **http://...** and try again:



The Android SDK Manager window

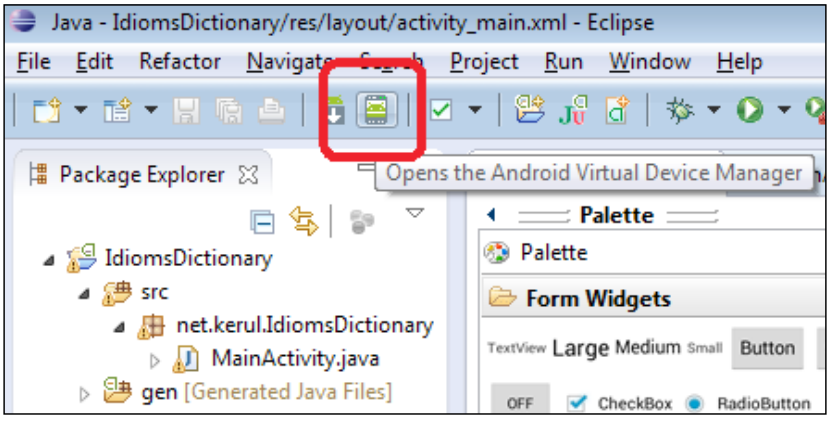
Android virtual device manager

Android virtual device is a virtual mobile device (emulator) that runs on your computer. The emulator lets you test an Android application without using a physical device. Although, it's not the best testing approach, as it just mimics the device, but at least you have something to test in case you cannot afford an actual Android device.

When the emulator is running, you can interact with the emulated mobile device just as you would in an actual mobile device, except that you use your mouse pointer to touch the touchscreen and you are able to use some keyboard keys to invoke certain keys on the device.

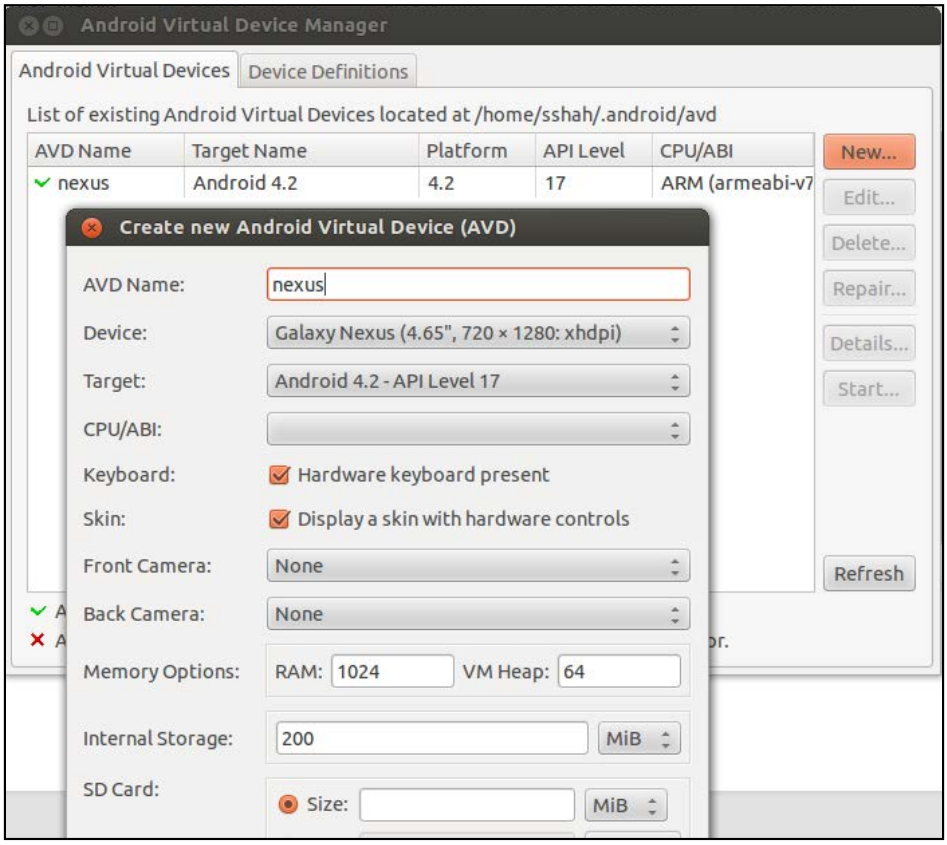
The Android emulator mimics all of the hardware and software features of a typical mobile device, except that it cannot place actual phone calls. It provides a variety of navigation and control keys, which you can "tap" using your mouse or keyboard to generate events for your application. It also provides a screen in which your application is displayed, together with any other running Android applications. For some features we may have to be aware of hot keys and details are at `http://developer.android.com/tools/help/emulator.html#KeyMapping`

Click on the button as shown in the following screenshot, to open the Android SDK and **AVD Manager** window. **AVD** is **Android Virtual Device**:



The AVD icon

The **AVD Manager** is shown in the following screenshot. First, click on **New…** to set a new emulator, as seen in the screenshot. Enter a name (for example, `nexus`), choose a target (make sure the Android OS system image has been downloaded for the selected target), and for simplicity choose the device, and all other fields will be auto-populated. We can also edit if you want something different. Also, choose **CPU** as **ARM (armeabi-v7a)** and click on **Create AVD**:



Creating a new AVD

Click on the new AVD that is already created, and start the AVD using the **Start** button. Use the default setting and click on the **Launch** button.

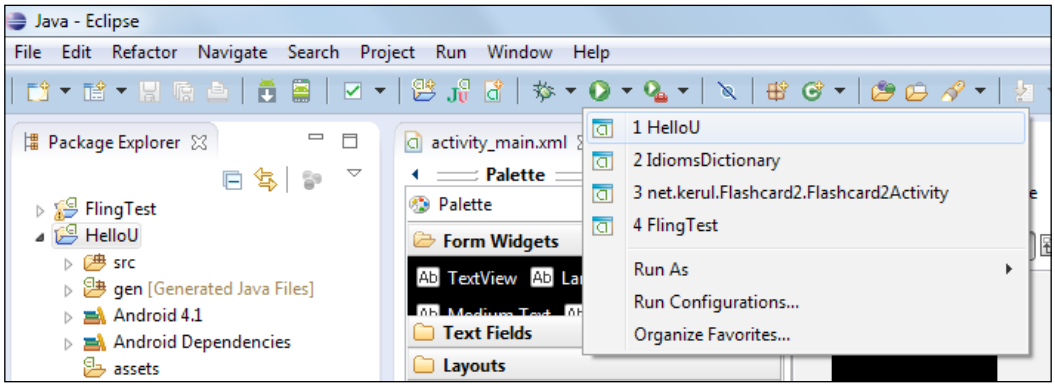
If we have a lower specification of processor and memory, you will notice that its emulator boot-up is really slow. I would like to advise you to have at least 3GB of RAM to make it faster.

Wait until the left screen displays a nice picture with icons, as shown in the following screenshot. The left component is your device (smartphone) screen and the right component is the physical smartphone keypad:

The Android emulator

Running the Application

The project with no error will be able to be executed and sent to the AVD. To run a project, click on the **Run** button as, shown on the following image. If your system is already running several emulators, Eclipse will ask which version of the emulator to use:



The Run application button

Getting help

- Go to **Help** in the menu, and choose **Search**
- Eclipse help: `http://help.eclipse.org/juno/index.jsp`
- ADT help: `http://developer.android.com/tools/help/adt.html`
- Android developer's official reference: `http://developer.android.com`
- ADT update: regularly check the ADT update from the menu, **Help | Check for Updates**
- More on DDMS: `http://developer.android.com/tools/debugging/ddms.html`

Summary

In this chapter, we discussed several important tools available in the Eclipse and the ADT, such as the project explorer, code editor, graphical user interface designer, properties window, debugging pane, Dalvik debug monitor, SDK manager, AVD manager, and the run application facilities. The next chapter will discuss how to create a new Android application project.

3
Creating a New Android Project

This chapter will demonstrate how to create a new Android app with a simple interaction using the button and text field. We will also write interactivity code, compile and run an app on the emulator/actual device. To illustrate this chapter, we will be creating a simple project named HelloU app.

- Creating new Android application project string resources
- Using the graphical layout designer
- String resources
- The XML layout editor
- Widgets' interactions through the source code editor
- Toast message
- Running the application on the emulator
- Running the application on an Android device
- Getting help

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Creating a new Android application project

To create a new Android project in the Eclipse, navigate to **File** | **New** | **Project**. A new project window will appear, then choose **Android** | **Android Application Project** from the list. Click on the **Next** button.

- **Application Name**: This is the name of your application, it will appear side-by-side to the launcher icon. Choose a project name that is relevant to your application.
- **Project Name**: This is typically similar to your application name. Avoid having the same name with existing projects in Eclipse, it is not permitted.
- **Package Name**: This is the package name of the application. It will act as an ID in the Google Play app store if we wish to publish. Typically it will be the reverse of your domain name if we have one (since this is unique) followed by the application name, and a valid Java package name, else we can have anything now and refactor it before publishing.

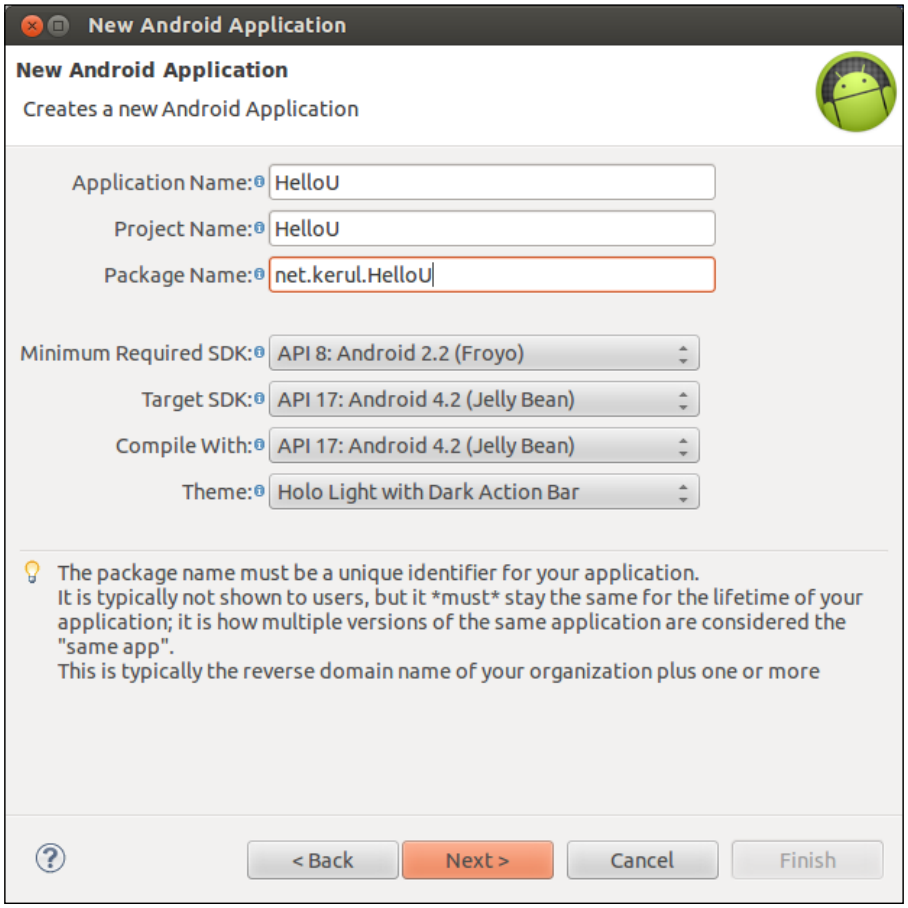
The `android:minSdkVersion` is an integer designating the minimum API Level required for the application to run. If not sure, leave it to whatever is selected.

For example, you might have your app set to `android:minSdkVersion="7"`. This setting will guarantee that your app works on devices with Android Éclair (2.1) or above, but not below.

The `targetSdkVersion` is the target devices you are focusing on. Let's say your app has `android:minSdkVersion="16"`, it means the apps could utilize all the features of Android Jelly Bean. However, bear in mind that features, such as the ability to move the app to an SD card and native Unicode are not supported in Android (2.1) Eclair. Though these features are available starting in the API level 8 (Android 2.2/Froyo) and level 11 (Android 3.0/Honeycomb), they cannot be utilized in the lower version of Android.

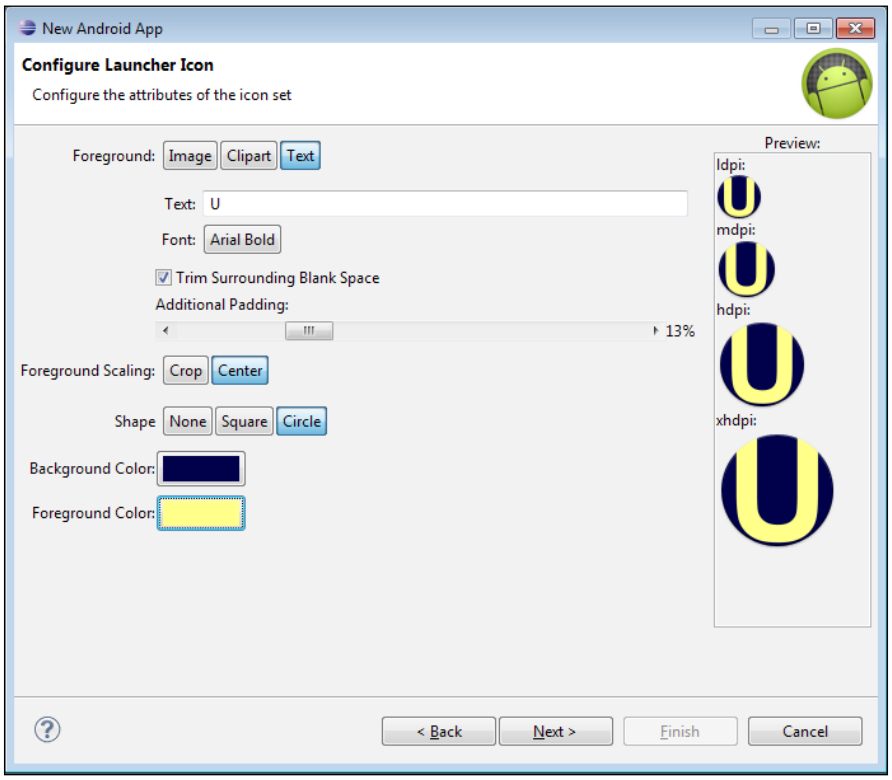
Do keep in mind that your `targetSdkVersion` has to be equal or more than the `minSdkVersion`. Otherwise, it doesn't really make much sense.

Click on **Next** to move to the next step:



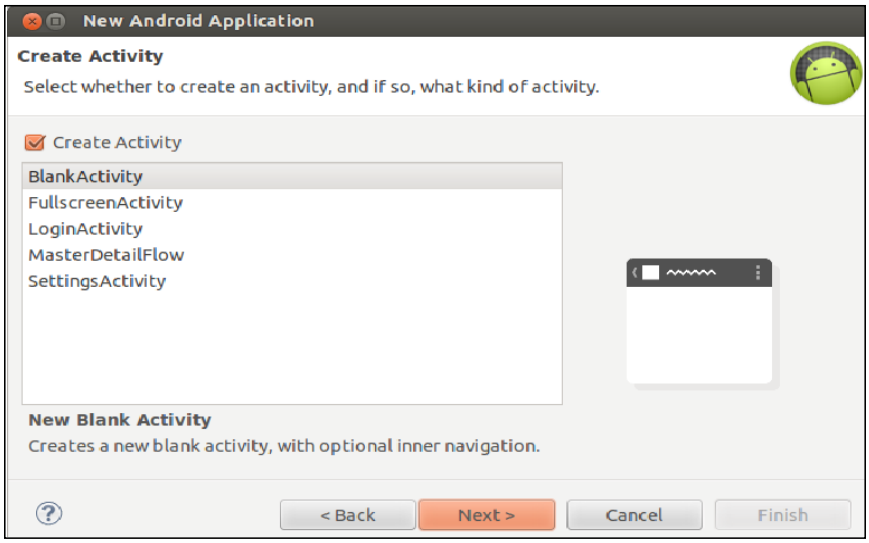
Create a new Android project

This is the window to configure your launcher icon. The launcher icon is the icon that will appear in the home screen or in the application drawer. This is an important aspect of your app as it will be representing the app. For this purpose, you may use the icon creator wizard using the available text and icon shape pre-customized in the ADT. Set the foreground as text, provide the letter `U` as the **Text**, pick the circle as the icon shape and adjust your color preference, as shown in the following screenshot. This wizard will create a simple icon and provides the ldpi (36x36 pixels), mdpi (48x48 pixels), hdpi (72x72 pixels) and xhdpi (96x96 pixels) of the launcher icon. Icons of different sizes are created to address various devices with different configuration of screen sizes and resolution. Click on **Next** to proceed:



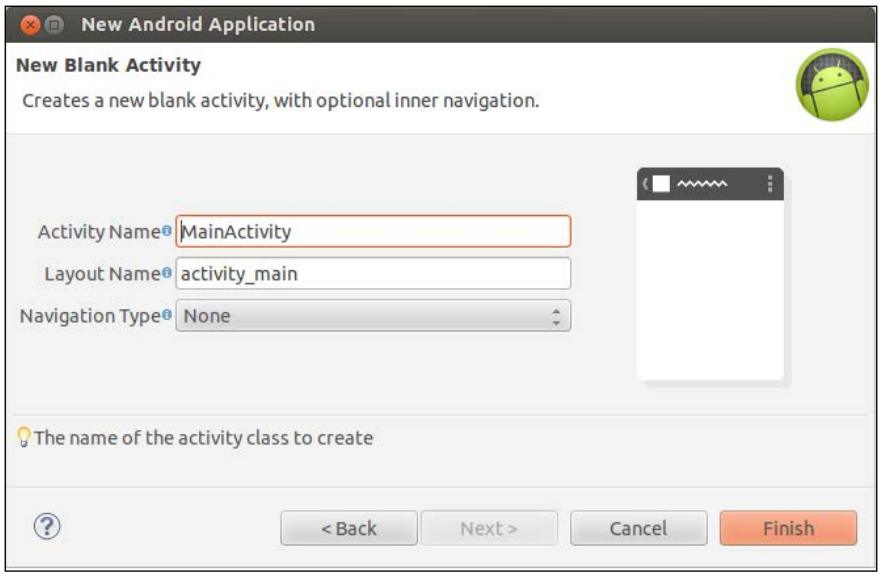
Launcher icon creator

Choose **BlankActivity** in the window, as shown in following screenshot, and click on **Next**:



Choose blank activity

The next window appears to input the `MainActivity` name, as shown in the following screenshot and click on the **Finish** button:

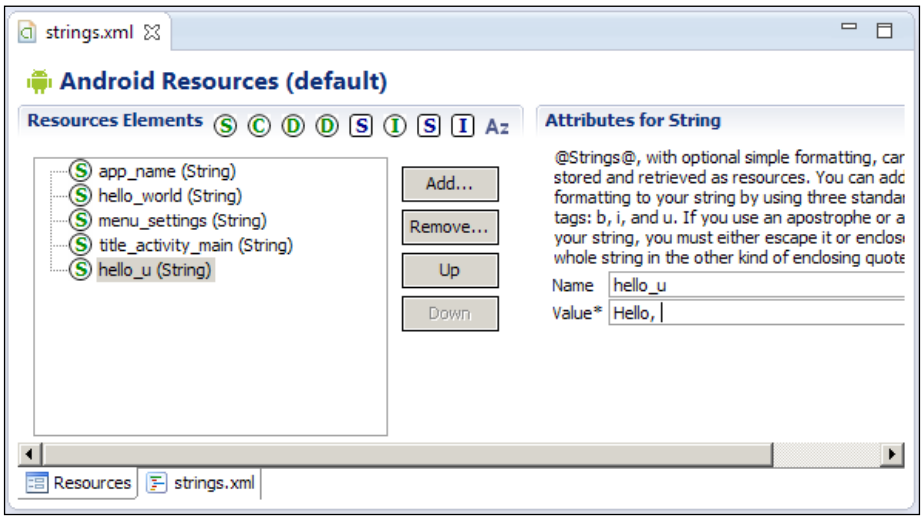


String resources

Usually, it is a practice for Android application to store the string values for user interface reference in the XML file due to the nature of mobile apps, which is distributed internationally. So it is best to provide multiple language options. However, this practice is optional, and you may use direct string assigning if you wish to do so.

The string resource file is in an XML form and available through the project tree in `res/values/strings.xml`. These string resources can also be used to store color information, integer arrays to name some.

Now, add a new string value by clicking on the **Add** button, provide the variable name in the **Name** box and the **Value** of the string. Press *Ctrl* + *S* to save the changes. For example, in the following screenshot, a new string variable is created as `hello_u` and the value is **Hello,**:



Adding a new string value

Add two more string values based on the table that follows. These strings will be used as the widgets' caption:

String variable	Value
`s_tvName`	Your name:
`s_btnDisplay`	Display name!

If you notice, we use `s_` to indicate it is a string variable from the resources, `tv` to indicate a `TextView`, and `btn` to indicate a button. Bear in mind that these conventions are not fixed, you may use your own preferences.

The new string values created will be saved in the `string.xml` file. The XML code is available by clicking on the tab on the red arrow, as shown in the following screenshot:

The string.xml code file

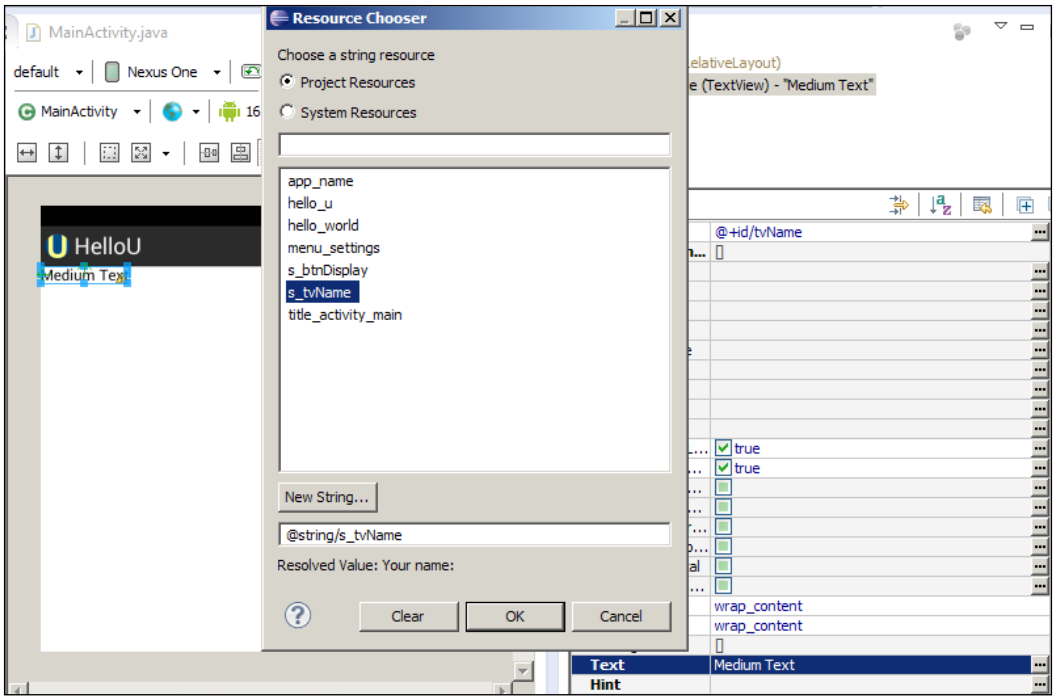
Using the graphical layout designer

The next exercise is to add a text label, a text box and a button. These elements are called widgets in Android which has the class name `TextView`, `EditText` and `Button` in the Android API. We will not go through the details of these classes; most importantly we could apply these widgets in our app.

To open this layout, double-click the `res/layout/activity_main.xml` file from the project explorer.

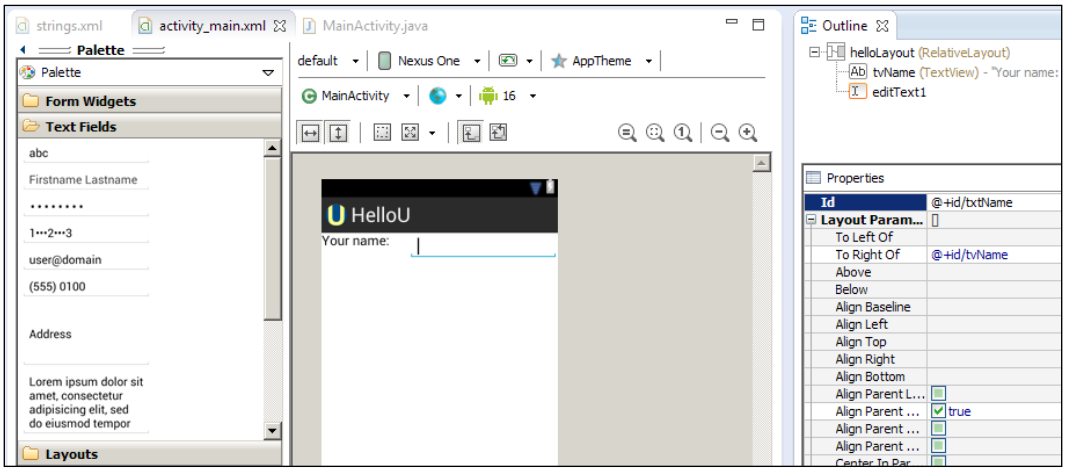
On the left of the app screen, you'll see the **Palette**. Browse the **Form Widgets**, there are several widgets including the `TextView`. Click and drag the `TextView` widgets to the app screen. Change the widget ID into `"@+id/tvName"`, and make sure to press *Enter* to confirm your changes and save them to the XML file. The `"@+id/"` is the ADT representation to say that the new ID has to be created and assigned to the widget.

After that, set the Text properties to point the value defined in the string resources, `s_tvName`. This could be done by clicking on the button with three dots, on the right side of each property. Press *Ctrl* + *S* to save the changes and to make sure the changes appear in the XML file:



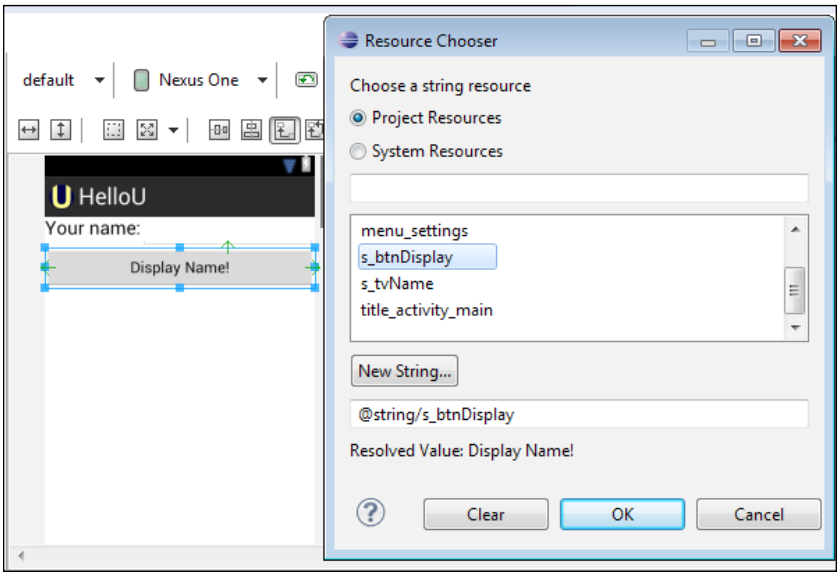
Changing the caption of a TextView

The next widget to add is the EditText with **Id** `txtName`, associate label as **Your name:** to accept user input, as shown in the following screenshot:



Adding an EditText

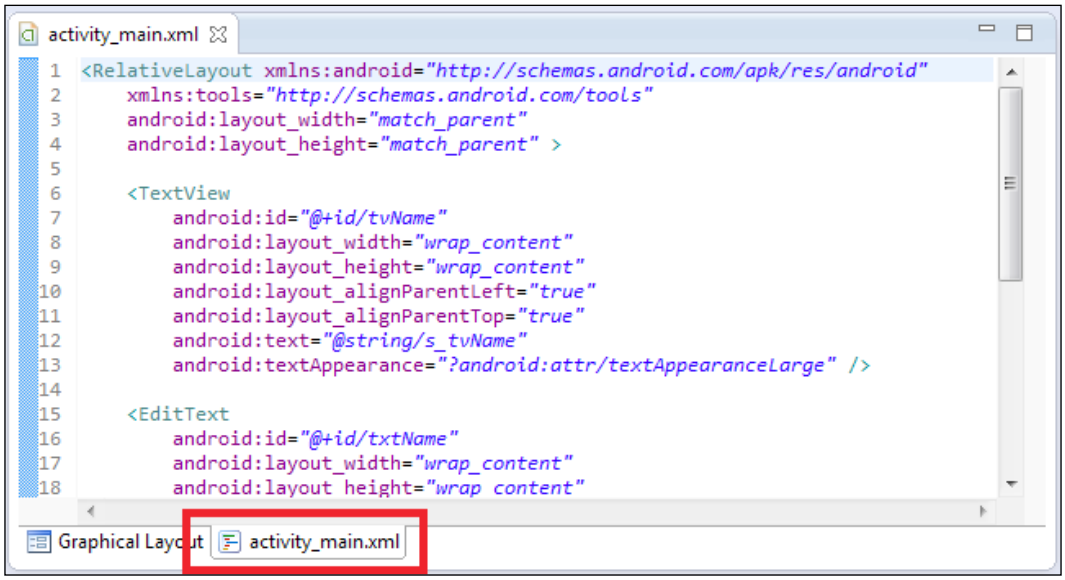
Add another widget, button, specify ID `btnDisplay` and associate label as **D isplay Name**, as shown, and expand it horizontally across the screen. You may use the resize feature by clicking and dragging the bluish resize mark on the edge of the widget:



Adding a Button

The XML layout code editor

The code editor is an alternative to change the layout properties. We recommend you change this code directly if you have prior knowledge of XML. It's a straight forward XML code actually. To access this code directly, just click the `activity_main.xml` on the bottom of the layout editor, as shown in following screenshot:



Accessing the XML layout code editor

Widget interactions through the source code editor

The layout we designed previously does not have to interact with each other automatically; let's make it happen. To put in simple words, when we execute the project, clicking on the button on the app will not trigger any action. We need to add the code for the interactions.

What we are trying to do is when the user taps on the button **Display Name!**, the app will capture any text inside the `TextView` and produce a simple popup to display your name.

To achieve our goal let's play around, go to `src`, double click on the `package` folder and double click again on the file `MainActivity.java`. This Java file will contain the code to load the layout of the XML file `main_activity.xml` in order to create a UI. The Java code as follows is the default code provided by ADT.

You will see the package name on line one and several classes imported to the project. The code in line six is the main class declaration which inherits the `Activity` class. method `onCreate` in line eight is the first method to be called when the apps start. The `setContentView(R.layout.activity_main)` is the command to initialize the screen layout based on the main screen designed previously. And the method in line 13, which is to create the screen menu, will be discussed later in *Chapter 5, Adding RadioButton, Checkbox, Menu, and Preferences*.

```
package net.kerul.HelloU;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity{
//First method called when App starts
 @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
      // loads Screen menu
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main,
         menu);
        return true;
    }
}
```

In order to provide button interaction, we need to add implements `OnClickListener` to the main class header.

```
public class MainActivity extends Activity implements OnClickListener
{
```

In the import section of the code, add this line:

```
  import android.view.View.OnClickListener;
```

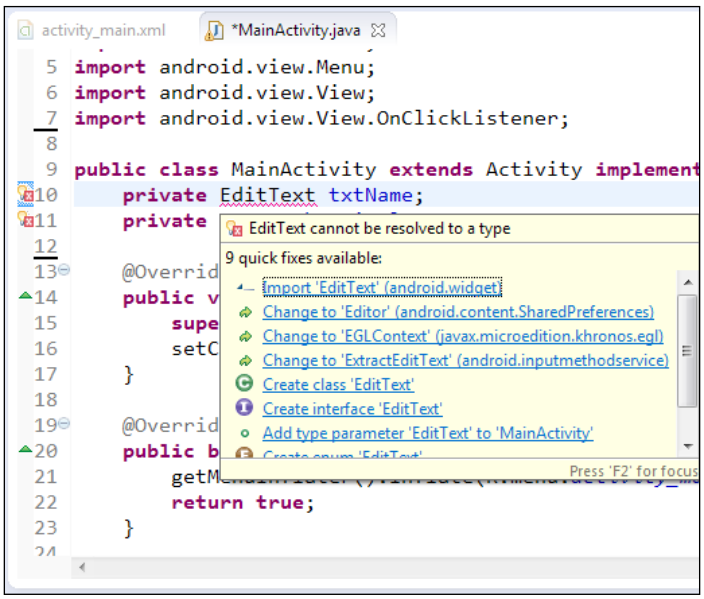
Now, initialize all the widgets that will get involved in the process. Immediately after the main class header, add the widgets' member declaration.

```
public class MainActivity extends Activity implements OnClickListener{
   private EditText txtName;
   private Button btnDisplay;
   …
```

Since `EditText` and `Button` are also another class which needs to be imported from the Android API, so add a couple of lines in the import section.

```
import android.widget.Button;
import android.widget.EditText;
```

In Eclipse you do not need to memorize all the classes and the packages' names that are needed to be imported. Just put the cursor (caret) to the class and press *Ctrl* + *Shift* + *O*. The IDE will help you to include the packages involved or point your mouse to the additional class, a menu will come out, and choose to import the class.



Menu to import class from the Android API

Next is to link the code and the layout design in the `MainActivity.xml file`. This is needed since the ADT is incorporating the MVC (Model-View-Controller) development method. It means that the screen layout is separated from the code to provide high project maintainability.

Basically after the layout has been loaded using `setContentView` you need to have access to these widgets that hide within that layout. This is where `findViewById ()` comes into play.

```
txtName=(EditText)findViewById(R.id.txtName);
btnDisplay=(Button)findViewById(R.id.btnDisplay);
```

The button is the action; we need to add the event listener to the button. The line to add is as follow:

```
btnDisplay.setOnClickListener(this);
```

Here we made the Activity itself implement `onClickListener`.

For any on-click event to be handled, Java needs a special method to be included. Inside the method is where the task will be executed. In our case, if the user clicks (or taps) the button (`btnDisplay`), the app will extract the content of the text field (`txtName`) and display the content on the screen. The action can be coded as follows:

```
public void onClick(View arg0) {
   if(arg0.getId()==R.id.btnDisplay){
      String hellomsg="Hello, "+txtName.getText().toString();
      Toast.makeText(this.getApplicationContext(), hellomsg,
         Toast.LENGTH_SHORT).show();
  }
   }
```

`View arg0` is the element that triggers the action. `arg0.getId()` is the method to get the ID of the widget triggering the action. If the widget ID is the `btnDisplay`, then do the action of capturing the input and display it to the screen.

To fetch the string of text field widgets, use the following code:

```
txtName.getText().toString();
```

`Toast.makeText()` is the method to display a short/brief message on the screen, we will discuss it in the next section.

The complete code would be:

```
package net.kerul.HelloU;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener{
   private EditText txtName;
   private Button btnDisplay;
```

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

       txtName=(EditText)findViewById(R.id.txtName);
       btnDisplay=(Button)findViewById(R.id.btnDisplay);
       btnDisplay.setOnClickListener(this);
    }

    public void onClick(View arg0) {
       if(arg0.getId()==R.id.btnDisplay){
           String hellomsg="Hello, "+txtName.getText().toString();
           Toast.makeText(this.getApplicationContext(), hellomsg,
               Toast.LENGTH_SHORT).show();
       }
    }
}
```

Toast message

This is one of the common practices to pop-up a message box for notifying the user. This kind of notification is a type of notification that does not require a user answer or feedback.

```
Toast.makeText(this.getApplicationContext(), hellomsg,
         Toast.LENGTH_SHORT).show();
```

`Toast.makeText()` contains three parameters which are the application context, the message and the time length.

- The application context is the current screen to display the message
- The message is the string to be displayed
- The time length is consisting of a short or longer duration of the message display and has to be one of `Toast.LENGTH_*` constants

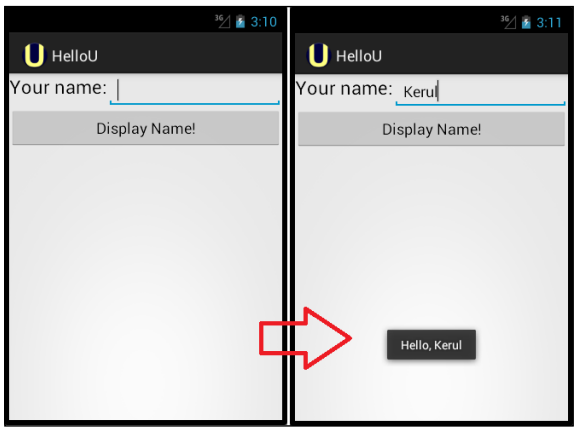
The arrow in the following screenshot is pointing to a Toast:

Example of a Toast

Running the application on the emulator

Running the **HelloU** app in the emulator would need you to start the emulator first. Start the emulator that has the Android version that suits your target platform. Once the emulator is fully loaded, we can compile and run the app.

Click on the **HelloU** project on the project explorer (this is to activate the project). Navigate to **Run** in the Eclipse menu, and choose **Run** or press *Ctrl + F11* for a shortcut. Select run as Android Application, and *Enter*. Wait for a couple of seconds and view your emulator. The **HelloU** app will appear shortly, as in shown in the following screenshot. Enter your name and tap on the **Display Name!** button, the Toast message will appear with the name entered on the bottom of the screen:



The HelloU app running in the Emulator

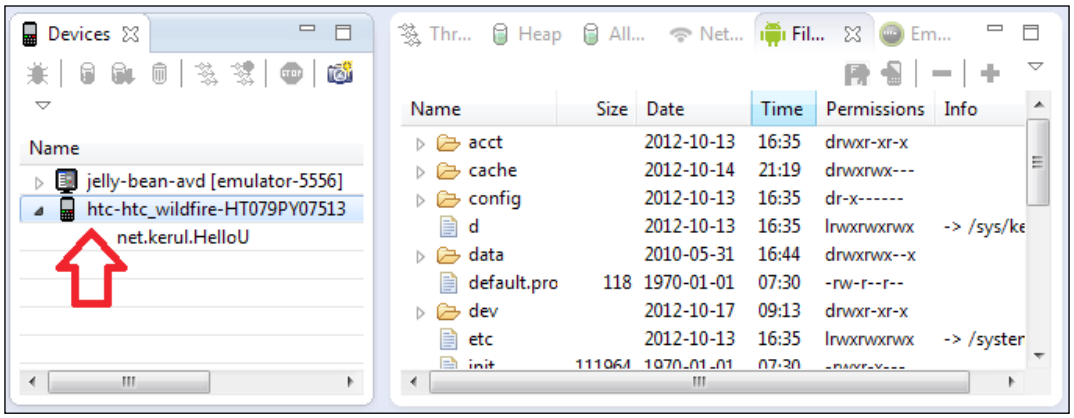
Running the application on an Android device

To run and deploy on a real device, first install the driver of the device. This varies as per device model and manufacturer.

These are some links you could refer:

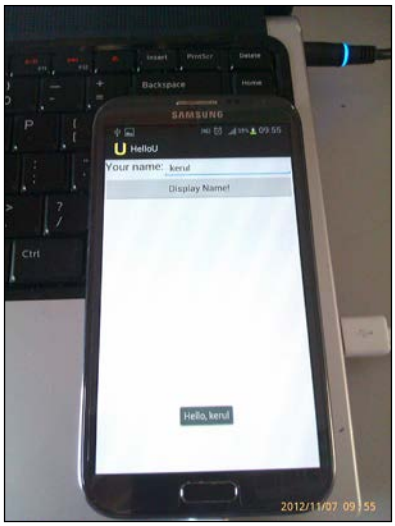
- For Google Android devices only `http://developer.android.com/sdk/win-usb.html`.
- Others: `http://www.teamandroid.com/download-android-usb-drivers/`.

Make sure the Android phone is connected to the computer through the USB cable. To check whether the phone is properly connected to your PC and in debug mode, please switch to the DDMS perspective.



The Android phone as appear in the DDMS.

If everything goes well, then run the app. Notice that a window appears asking you to select between the emulator and a real Android device; select the Android device. A few seconds later, the app will be running in the Android phone.



HelloU app in the actual Android device. Getting help

The following are some references to guide you on using the Eclipse and ADT. You can spend some time going through the documentation and tutorial to get updated. Reading the tutorials and discussions at `stackoverflow.com` are among the convenient way of learning these tools.

- Go to **Help** in the menu, and choose **Search**.
- Eclipse help: `http://help.eclipse.org/juno/index.jsp`
- ADT help: `http://developer.android.com/tools/help/adt.html`
- Android Developer's official reference: `http://developer.android.com`
- ADT Update: regularly check the ADT update from the menu, **Help | Check for Updates**.
- More on DDMS: `http://developer.android.com/tools/debugging/ddms.html`

Summary

Congratulations! You now have an Android app of your own. You have designed the screen layout, added a label, text field and a button. The simple interactivity exposed you to how to develop an android mobile app. In the next chapter, we will add more widgets and learn to develop more complex apps involving multiple screens.

4
Incorporating Multimedia Elements

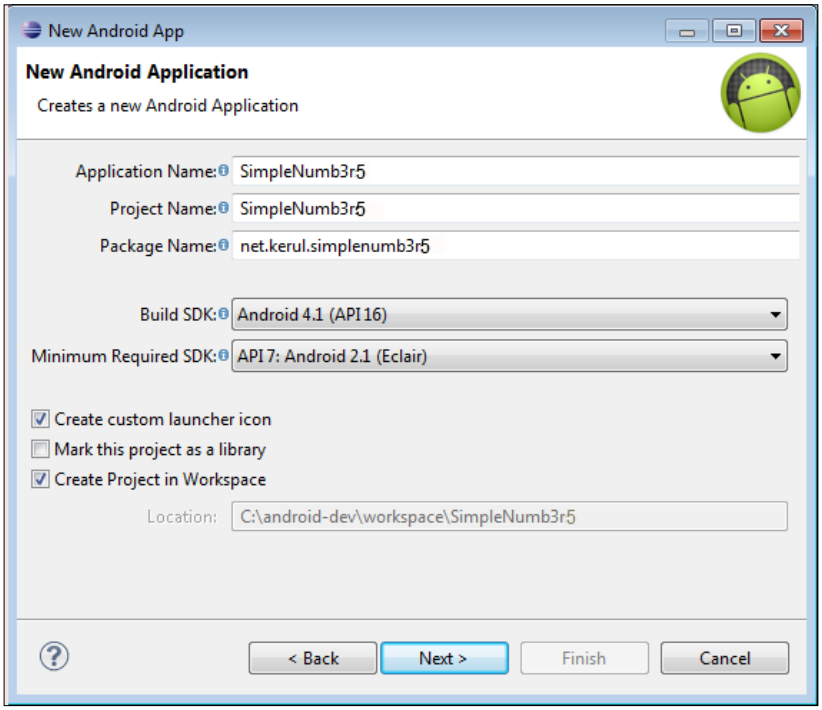
This chapter will discuss how to incorporate multimedia elements inside a project and handle several screens in an app. The readers will be shown how to add images, sounds and an HTML page in the project. We will discuss the following topics with the help of a project called `SimpleNumb3r5`:

- Forming the layout
- Adding the image resources
- Inserting ImageView
- Inserting ImageButtons
- ImageButton and handling events
- Adding audio and multiple screen support
- Inserting HTML in a WebView
- Using Intent and Activity
- Adding a new activity in the manifest file
- The final product – run, deploy, and test app

For this chapter, we need a new project that will cover the Android devices from Version 2.1 (API level 7) to the latest version. So set `android:minSdkVersion` to 7, and `android:targetSdkVersion` to 16.

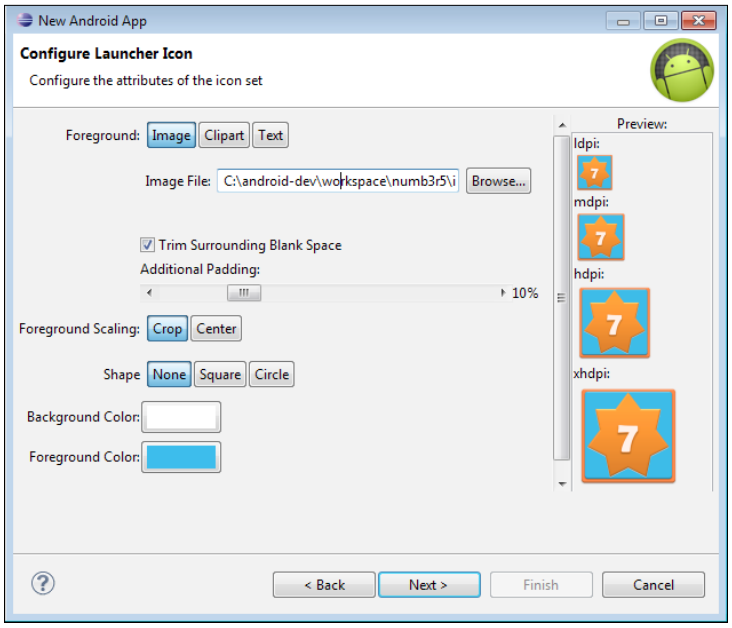
The icon and other resources are available in a downloadable source code (refer to the download tip mentioned in the *Preface* of this book). Download these materials prior to developing this app. We do not want to make your life miserable doing the graphic design.

The selected name for the new app is `SimpleNumb3r5`, as shown in the following screenshot. If you are wondering why we chose Android 2.1, this is to widen the device coverage:



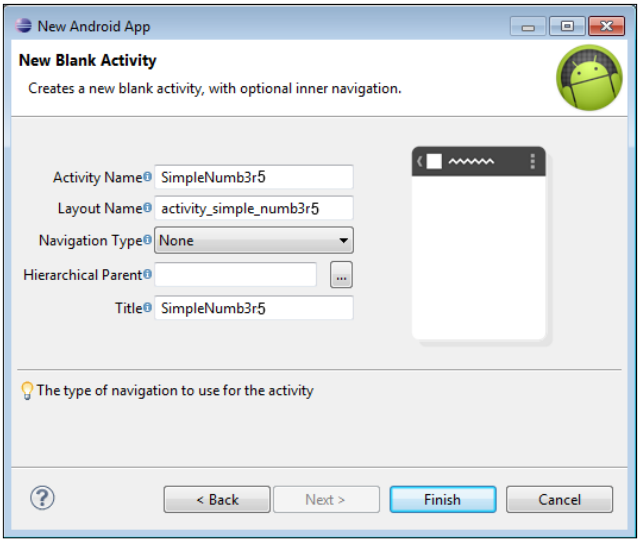
Create a new Android project named SimpleNumb3r5

We provide the launcher icon in the resource materials and the image named `ic_launcher-web.png` in the **Image File** field, as shown in the following screenshot. This is the dedicated logo of this app. Should you prefer a different logo to suit your app, you are welcome to design it personally. By using this wizard, the icon launcher will be prepared to suit the `xhdpi`, `hdpi`, `mdpi`, and `ldpi` formats in the respective drawable folder.



Creating the launcher icon

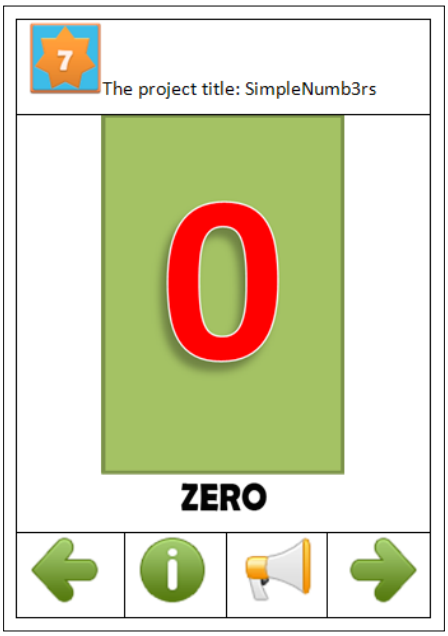
The next screen, as shown in the following screenshot, is to provide a name of the application. This can be any string that has the right meaning suitable to the app. The layout name will be created automatically for you, and could be changed to your preference. Choose the navigation type as **None** as it has no concern with respect to our application development.



Choose the blank activity

The following screenshot is the mock-up of the app being developed. We have a major section of the screen dedicated to display the image of the numbers zero to nine and the spelling. The bottom row of the screen is the navigation bar where the user may navigate to the previous and next screen. The button with the speaker is for the user to listen to the number spoken to them. The button with the lower case, **i**, is the icon to show the information screen.

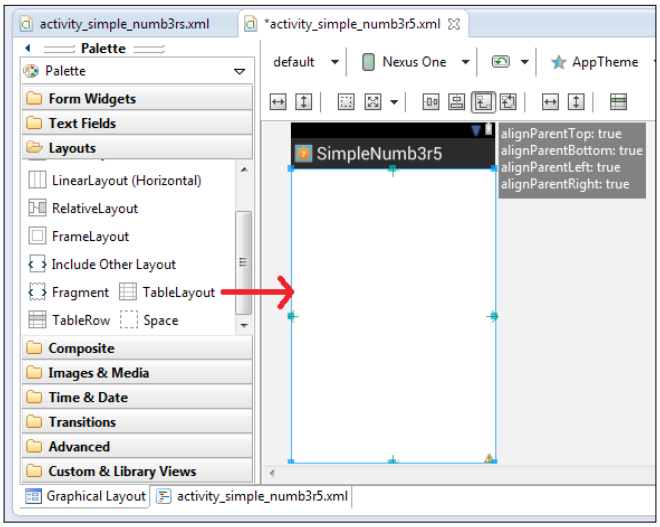
- The project title (appears by default).
- The image number location. This row consists of the three cells merged together.
- The bottom row consists of previous, info, play sound, and next buttons.



The main screen mock-up

Adding a TableLayout

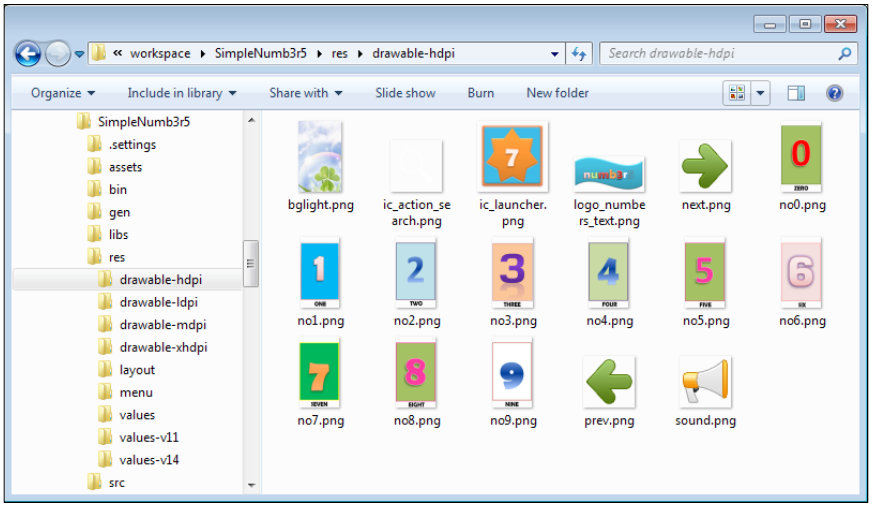
Our project will consist of one TableLayout and inside it there are two TableRows. By default, when you add a TableLayout, the IDE will include four sets of TableRows. Remove two rows by using the XML code editor, the previous app mock-up can provide some guidelines to remove the rows not in use. Adjust the TableLayout, so that it utilizes all the space of the screen layout, as shown in the following screenshot:



Inserting a TableLayout

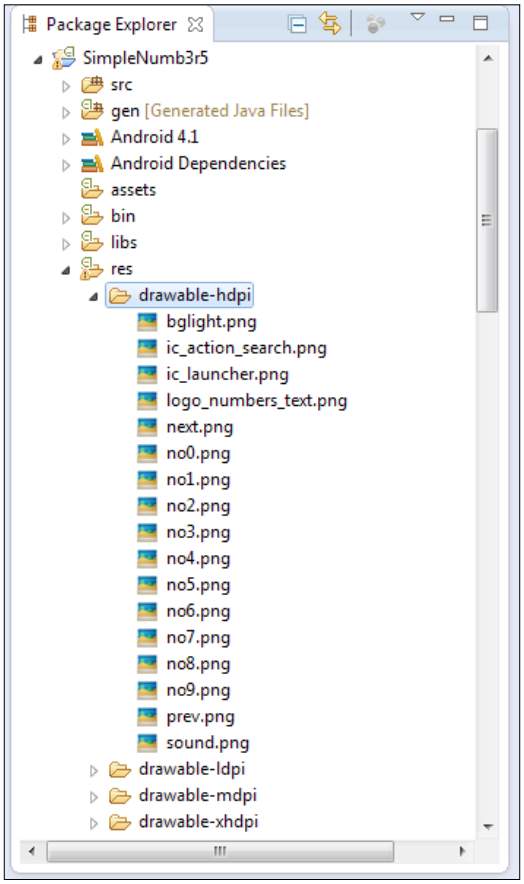
Adding the image resources

Copy the images provided in the supplement files for *Chapter 4* to the `res/drawable-hdpi` folder through the Windows file manager, as shown in the following screenshot. In this exercise, we just provide the image resources for `hdpi` drawable. It's always a good practice to prepare all the suitable resources for `xhdpi`, `mdpi`, and `ldpi` accordingly. Do not forget we have a lot of screen size variant in the Android devices. Currently we also do not consider resources for the tablet size devices.



The resources for the drawable

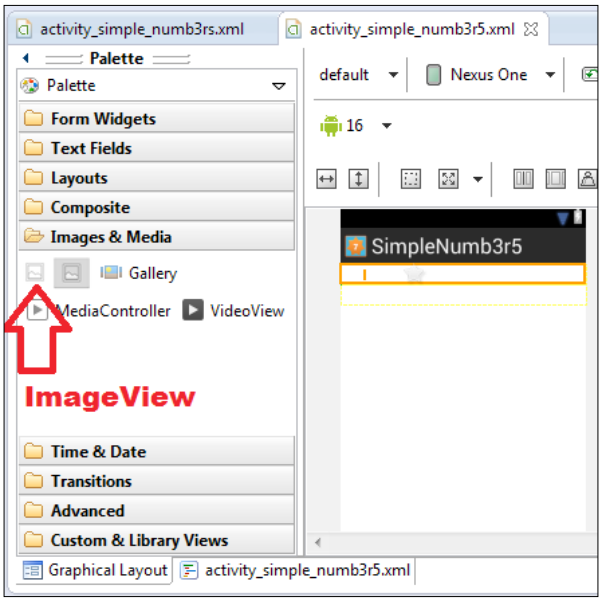
Then go to your project explorer (in Eclipse), right-click on `res/drawable-hdpi` and click on **Refresh**. The following screenshot shows the appearance of the `drawable-hdpi` folder after the image resources have been copied:



The resources for the drawable

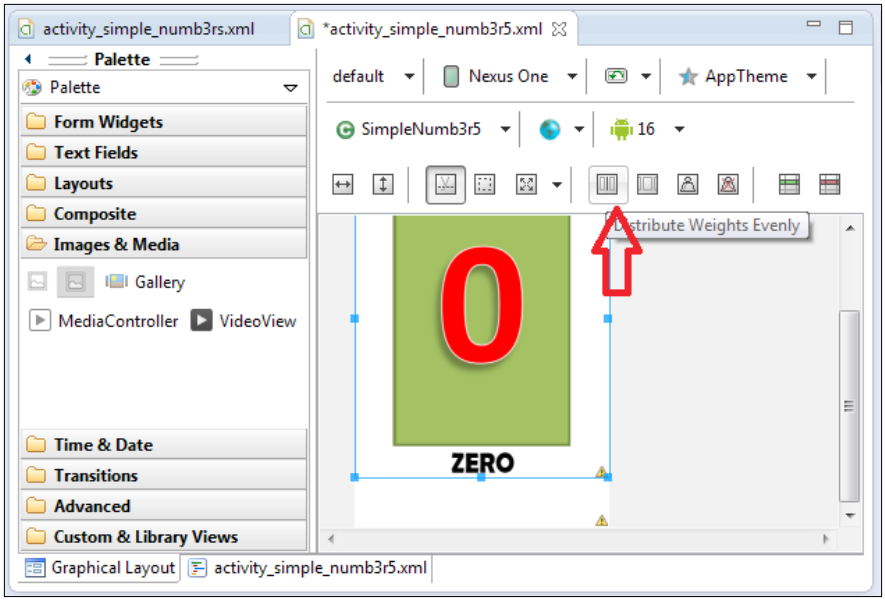
Adding ImageView

As shown on the previous screenshot, our app has an image 0 that fills the entire screen and to achieve that let's add an ImageView to the first row of the TableLayout. Use the no0 image in the drawable folder as the initial image (zero is the first number to be displayed). Adjust the width and height of the ImageView to populate the screen.



Adding an ImageView to the app screen

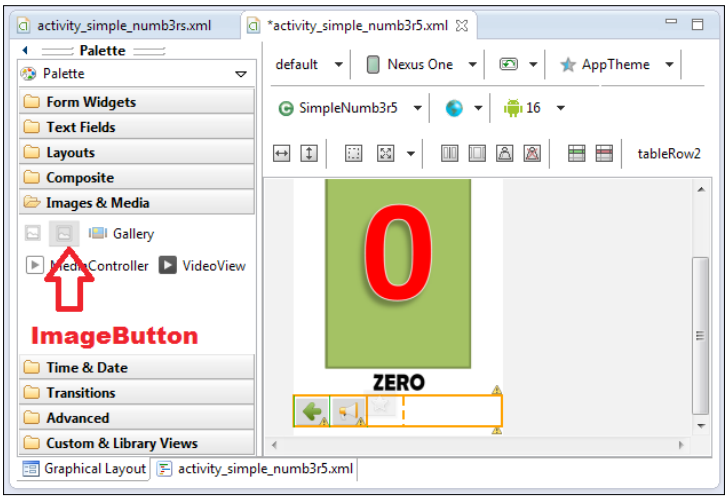
Distribute weight (specifies how much of the extra space in the layout to be allocated to the View) evenly to center the widget. Use the button shown in the following screenshot to adjust the ImageView to the center of the screen. Do this while the ImageView is active (selected):



Distribute weight evenly

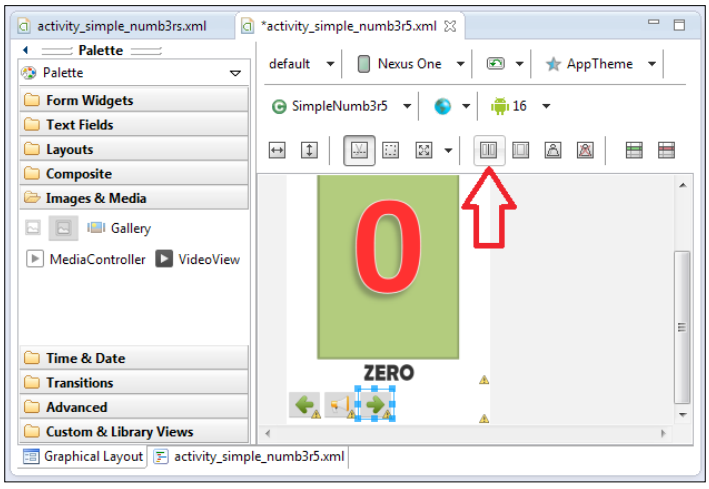
Adding ImageButtons

The second row in TableLayout is for the navigation buttons (previous and next) and the play sound button. ImageButton is more attractive for this kind of app. When you create an ImageButton, you will be asked to choose the image. For the first button use the image **prev** from the drawables. The second is **sound** and the last one is **next**. These buttons need to be added one at a time, as shown in the following screenshot:



Adding ImageButtons

Activate (select) one of the buttons and distribute evenly, as shown in the following screenshot. This is to make sure all the buttons are spread evenly across the screen's width.

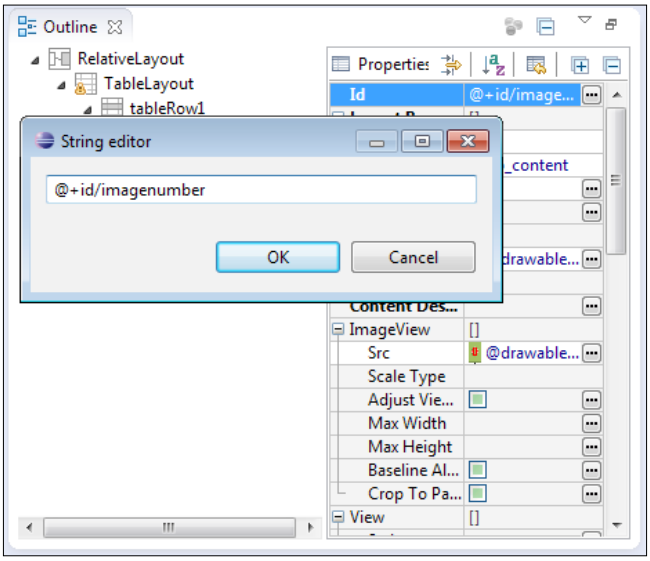


Distribute weight of the ImageButtons evenly

And if you prefer, change the background of your screen. A background image has been prepared for you; it is in the drawables and is named **bglight**. Activate the main layout by clicking on the app title/logo on the app screen. Change the background properties by clicking on the three dots button on the right-side of the attributes. Later, you may add the **btninfo** button to display the app's information.

Assigning the widget's ID

There are basically one ImageView and three ImageButtons. To change the ImageView ID, select it and go to the widget properties on the right-side. Click on the three dots button on the **Id** attribute. Change the ID of the ImageView to **imagenumber**, as shown in the following screenshot:



Changing the widget's ID through the Property window

After that, change all the IDs of all the buttons to **btnprevious**, **btninfo**, **btnsound**, and **btnnext**. Use the following table as a guide:

Widget	ID
ImageView	imagenumber
Left most button	btnprevious
Display app info	btninfo
Play sound button	btnsound
Right most button	btnnext

Finally, you will get the screen, as shown here:

The whole layout design of the main activity

The following XML code is available through the XML editor in the tab `activity_simple_numb3rs.xml` across the **Graphical Layout** tab:

```
<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:background="@drawable/bglight" >

    <TableLayout
        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" >

        <TableRow
            android:id="@+id/tableRow1"
```

```
            android:layout_width="wrap_content"
            android:layout_height="0dp"
            android:layout_weight="1" >

            <ImageView
                android:id="@+id/imagenumber"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:src="@drawable/no0" />
        </TableRow>

        <TableRow
            android:id="@+id/tableRow2"
            android:layout_width="wrap_content"
            android:layout_height="0dp"
            android:layout_weight="1" >

            <ImageButton
                android:id="@+id/btnprevious"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:src="@drawable/prev" />
            <ImageButton
                android:id="@+id/btninfo"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:src="@drawable/info" />
            <ImageButton
                android:id="@+id/btnsound"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:src="@drawable/sound" />
            <ImageButton
                android:id="@+id/btnnext"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:src="@drawable/next" />
        </TableRow>
    </TableLayout>
</RelativeLayout>
```

ImageButtons and handling event

This is where we code the ImageButtons click events. Open the Java source code from `src/net.kerul.simplenumb3r5/SimpleNumb3r5.java`. Here, we will be discussing the main activity class that will provide the screen navigation with the following code:

```
public class SimpleNumb3r5 extends Activity implements OnClickListener
```

The main class, as usual, will inherit the `Activity` class, and implement `OnClickListener` to enable the widget interaction.

The main variable declarations are as follows:

```
//initialize all widgets
  private ImageView imagenumber;
  private ImageButton btnprevious, btninfo, btnsound, btnnext;
  //define variables to track screen number, start from 0
  private int screennumber=0;
  //define a sound controller
  private MediaPlayer mp;
  //define an array for the sound files
  private String[] soundfile={"0.mp3","1.mp3","2.mp3","3.mp3",
    "4.mp3","5.mp3","6.mp3","7.mp3","8.mp3","9.mp3"};
```

Widget objects are `imagenumber` as for the container to display the number of images, and we have `btnprevious`, `btnsound`, and `btnnext` for the buttons.

The `screennumber` is the variable to keep a track of the current screen position; initially it is given the value 0 because we have a list of numbers that start from zero (0).

The sound controller object is named `mp`, and the string array named `soundfile` is the list of all the recordings of the spoken numbers from zero to nine.

The `onCreate` method is the place where all the widgets are initialized and linked together in a view, as follows:

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_numb3r5);
        imagenumber=(ImageView)findViewById(R.id.imagenumber);

        //create the object for the button
        btnprevious=(ImageButton)findViewById(R.id.btnprevious);
          //this button will initially be disabled
```

```
            btnprevious.setEnabled(false);
            //add listener to the button
            btnprevious.setOnClickListener(this);
          btninfo=(ImageButton)findViewById(R.id. btninfo);
            btninfo.setOnClickListener(this);
            btnsound=(ImageButton)findViewById(R.id.btnsound);
            btnsound.setOnClickListener(this);
            btnnext=(ImageButton)findViewById(R.id.btnnext);
            btnnext.setOnClickListener(this);

  }//end onCreate
```

Next, we have the `onClick` method to handle the navigation interactions. What we do here is basically disabling the `btnprevious` button if the `screennumber` is 0, and enabling it on for `screennumber` more than 0. `btnnext` will also be disabled if the `screennumber` value is 9, on when less than 9. These are to prevent runtime errors when the user trying to access that is less than 0 or more than 9. The `btnsound` value is currently ignored; it will be discussed later when we deal with sounds (that is, playing of sound/audio).

```
  //this method is to handle button click
    public void onClick(View arg0) {
      //when btnprevious is clicked
      if(arg0.getId()==R.id.btnprevious){
        screennumber--;// Decrement  1 to the screennumber
        changeNumber(screennumber);
        if(screennumber==0){
          // Disable previous Button
          btnprevious.setEnabled(false);
        }else{
          // Enable back disabled Button.
          btnprevious.setEnabled(true);
        }
        changeNumber(screennumber);
        btnnext.setEnabled(true);
      }

      //when btnnext is clicked
      else if(arg0.getId()==R.id.btnnext){
        screennumber++;//add 1 to the screennumber
        changeNumber(screennumber);
        if(screennumber==9){
          Disable  no screen available  next
          btnnext.setEnabled(false);
```

```
        }else{
          / Only prevoius screen  available
          btnnext.setEnabled(true);
        }
        changeNumber(screennumber);
        btnprevious.setEnabled(true);

      }
      //when btnplay is clicked
      else if(arg0.getId()==R.id.btnsound){
        //playSound -  will implement later
      }
      else if(arg0.getId()==R.id.btninfo){
        //display info  will implement later
      }

    }//end onClick
```

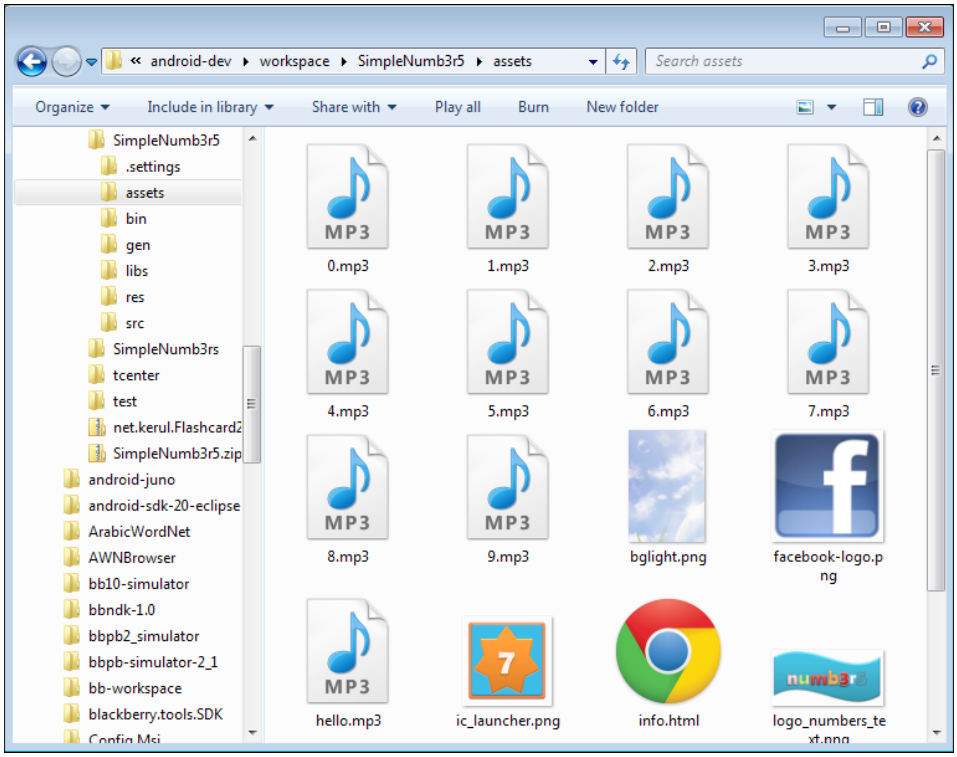
There is an additional method to switch the image of the numbers. The `R.id.imagefile` is the representation of the actual drawable image resources. Since we have 10 images altogether, and `R.id` returns `int`, so we can use the switch case 10 times as follows:

```
//this method is to change the number that appears on the screen
//  after the navigation button is clicked
//    as R.id retuns int so we  use switch
private void changeNumber(int screen){
    switch (screen){
      case 0:  imagenumber.setImageResource(R.drawable.no0);
      break;
      case 1:  imagenumber.setImageResource(R.drawable.no1);
      break;
      case 2:  imagenumber.setImageResource(R.drawable.no2);
      break;
      case 3:  imagenumber.setImageResource(R.drawable.no3);
      break;
      case 4:  imagenumber.setImageResource(R.drawable.no4);
      break;
      case 5:  imagenumber.setImageResource(R.drawable.no5);
      break;
      case 6:  imagenumber.setImageResource(R.drawable.no6);
      break;
      case 7:  imagenumber.setImageResource(R.drawable.no7);
      break;
```

```
        case 8:  imagenumber.setImageResource(R.drawable.no8);
        break;
        case 9:  imagenumber.setImageResource(R.drawable.no9);
        break;

        }
    }//end changeNumber
```

Adding audio

Before doing this exercise, copy all the sound resources to the `assets` folder. You may do this by copying all the mp3 files to the `assets` folder through the `File Manager`, as shown in the following screenshot:



Copy the MP3 files to the folder assets

Add code for `btnsound` in the `onClick` method. Add the following lines so that when the `btnplay` button is clicked, it will execute the method named `playSound()`. This method will receive a string argument as the value of the sound file name to be played.

The `soundfile` array variable is the variable that stores the list of the mp3 filenames, while the `screennumber` indicates the current number on the screen.

```
//when btnplay is clicked
    else if(arg0.getId()==R.id.btnsound){
      //call the method playSound
      playSound(soundfile[screennumber].toString());
    }//end btnsound clicked
```

The next important method is the `playsound` method. This method will play an mp3 sound file. The `soundname` parameter is the string that contains the sound file name which resides in the `assets` folder of the Android project.

```
public void playSound(String soundName){
  Boolean mpPlayingStatus;

  try{//try to check MediaPlayer status
    mpPlayingStatus=mp.isPlaying();
  }
  catch (Exception e){
    mpPlayingStatus=false;
  }
//if the MediaPlayer is playing a sound, stop it to play new voice
  if(mp.isPlaying()){
       mp.stop(); //stop the sound
       mp.release(); //remove sound from the memory
  }
  else{
    try{
      mp = new MediaPlayer();
      AssetFileDescriptor afd = getAssets().openFd(soundName);
      //set the sound source file

      FileDescriptor fd = afd.getFileDescriptor();
      mp.setDataSource(fd);

      mp.prepare(); // prepare for playback
      mp.start(); //play the sound

    }//try block
    catch(IOException e) {
      //display the error message in debug
      Log.i("Error playing sound: ", e.toString());
    }
  }
}//end playSound
```

The following is the explanation of the variables and processes involved:

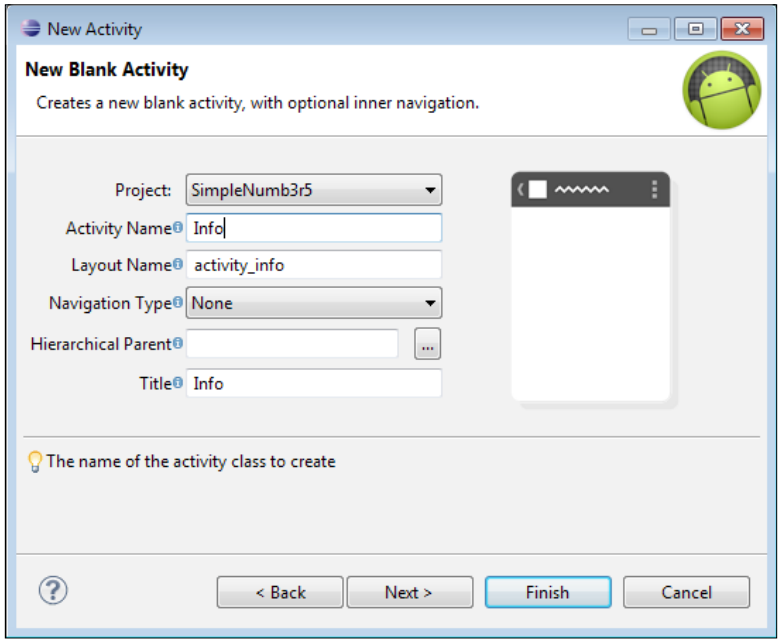
- `try...catch block`: This is an exception handler, whose purpose is to enclose the code that might throw an exception. In this case the exception is to try to catch any problem while trying to play the sound file using `MediaPlayer`. If you notice, the catch block is the statement that will be executed if a certain code execution causes an exception.
- `mp` is the object instantiated from the class `MediaPlayer`.
    - `isPlaying()`: Checks whether the `MediaPlayer` is playing, `True` is playing whereas `false` is otherwise
    - `setDataSource()`: Sets the data source to be used. In this case, the data source is `FileDescriptor`
    - `prepare()`: Prepares the player for playback, synchronously
    - `play()`: Plays the sound file
    - `stop()`: Stops the current sound playing
    - `release()`: Releases the sound from the memory
- `afd` is a variable instantiated from the class `AssetFileDescriptor`.
    - `getAssets()`: Retrieves the underlying resources (from the `assets` folder) via the `AssetManager` API
    - `openFD()`: Opens the file specified in the `String` argument
    - `getFileDescriptor()`: Returns the `FileDescriptor` data source that can be used to read the data in the file
    - `getStartOffSet()`: Returns the byte offset where this asset entry's data starts
    - `getLength()`: Returns the total number of bytes of this asset entry's data

Adding another screen in the app

This exercise is to add an information screen on the `SimpleNumb3r5` app. The information regarding the developer, email, Facebook fan page, and other information is displayed in the next screen. Since the screen contains a lot of text information including several pictures, so we make use of an HTML page as our approach here:

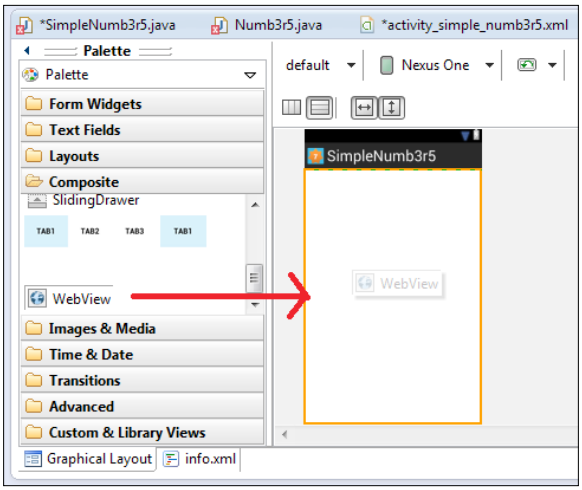
1. Now, create an activity class to handle the new screen. Open the `src` folder, right-click on the package name (`net.kerul.SimpleNumb3r5`), and choose **New** | **Other...** From the selections, choose to add a new Android activity, and click on the **Next** button. Then, choose a blank activity and click on **Next**.

2. Set the activity name as **Info**, as shown in the following screenshot and the wizard will suggest the screen layout as **info_activity**. Click on the **Finish** button.



Creating a new activity named Info

3. A blank new screen layout will appear. Remove the `HelloWorld` TextView (that comes with default). On the **Palette** panel, open the folder named `Composite`.
4. Click and drag the WebView widget. Change the ID of WebView to `webinfo`. This layout will be saved in the file `info_activity.xml`.



Adding a WebView widget

Adding HTML to WebView

Create an HTML page using your favorite web editor, or you may just reuse the HTML page in the resources provided (in the `assets` folder, file name `info.html`). The HTML page, as shown in the following screenshot, is a simple HTML page that contains the app information. If you find that the HTML is too simple, do add your own information. In this exercise, we will put the HTML pages and the resources inside the `assets` folder, hence before proceeding, copy all the related materials of the HTML page into the `assets` folder.

The HTML page in info.html

Next is to edit the source code for `Info.java` that resides in the folder `src/net.kerul.simplenumb3r5`. Add the following code to the existing template:

```
package net.kerul.simplenumb3r5;
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;

public class Info extends Activity {
    private WebView webinfo;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_info);

        webinfo=(WebView)findViewById(R.id.webinfo);
        //provide the URL path pointing to info.html
        webinfo.loadUrl("file:///android_asset/info.html");
    }
}
```

Let's understand the following lines of code added to the template above:

- `setContentView(R.layout.activity_info)`: `R.layout.activity_info` is referring to the layout created previously.
- `webinfo.loadUrl("file:///android_asset/info.html")`: This is the method to load an HTML page from a specific URL. The path to point to an HTML file inside the `assets` folder is `file:///android_asset/info.html`. This path cannot be found on a real device, however it provides access to the `app asset` files.

Intent and Activity

Intent is an abstract description of an operation to be performed. To be more specific, it is an asynchronous call which allows the application to request functionality from other Android components, for example, services/activities. It can be used with the `startActivity()` command to launch an activity. The previous code in `SimpleNumb3r5.java` is the main activity (or class) for this application. We've just created the second activity (class) in the file `Info.java`. In order for the second activity to appear, it has to be started using an intent.

We have decided to use the button `btninfo` as the trigger to invoke the second activity. Again, open the file `SimpleNumb3r5.java` and add the following lines to invoke another activity. These lines must be added to the `btninfo` button's `onClick` method. Notice that an instance of `Intent` is created as info. The main class is able to call the second class using the `startActivity()` method. The `Info.class` argument is referring to the second class.

```
  else if(arg0.getId()==R.id.btninfo){
    //invoke the Info activity
    Intent info = new Intent(this, Info.class);
    startActivity(info);
  }Adding Activity in Manifest file
```

In order to call the second class through `Intent`, the `Manifest.xml` files need to be modified. However, you will notice that this has been done automatically by the **Android Development Toolkits** since Version 20. In case the following lines are missing in `AndroidManifest.xml` please add it manually:

```
<activity
    android:name=".Info"
    android:label="@string/title_activity_info" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Should you need to display a webpage from the Internet in the WebView, you must declare the user permission by adding this line in `AndroidManifest.xml` above the `<application>` tag as follows:

```
<uses-permission android:name="android.permission.INTERNET" />
```

The final product run and test

After all the processes we have gone through, run the app in the emulator and you'll get the following screen:

SimpleNumb3r5 in action

Summary

In this chapter, we have explored a simple approach to incorporate several multimedia elements, such as image, an HTML page, and voice. The latest SDK is much more user friendly than any of the previous versions.

In the next chapter, we will learn more about the different widgets, such as menu, checkbox, radio button, and also about adding the preference screen.

5
Adding RadioButton, CheckBox, Menu, and Preferences

Are you excited enough? If not, you should be; we are half way through and ready to explore some more of the widgets that are commonly used and have a lot of significance in any application. The things to be covered in this chapter are adding a menu, check box, radio button, and preference to the application. We will make use of these widgets and create the `DistanceConverter` application .The main objective of this application is to convert distance entered in km/m to mile/foot and yards. The following are the steps that we cover in this chapter to successfully create the `DistanceConverter` application:

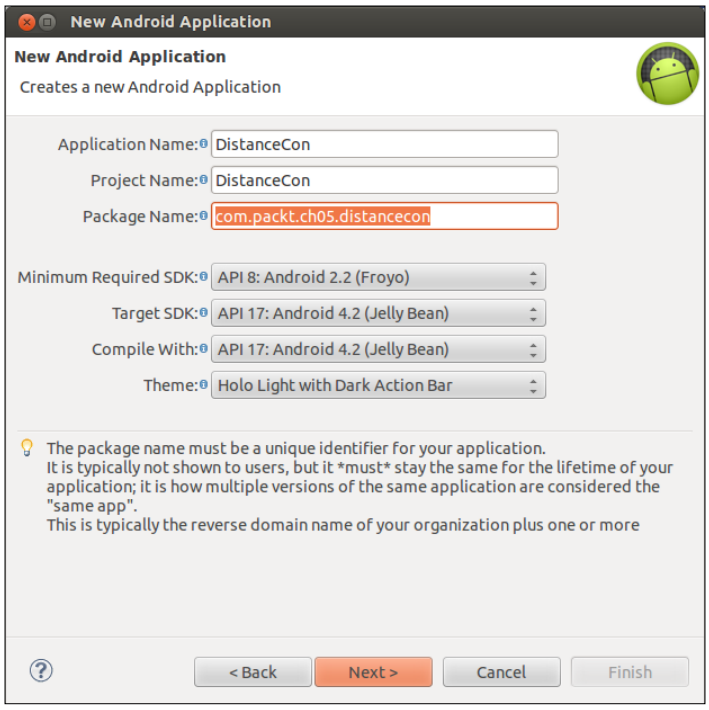
- Creating a project: `DistanceConverter`
- Adding a RadioGroup.RadioButton
- Adding a CheckBox
- Adding a menu
- Defining the Strings
- Defining the Preferences screen
- Hook up
- Binding menu and Preference
- Getting values from Preference
- Running the application

Creating a new project

The `DistanceConverter` application will allow users to input distance in km/m and convert them to miles, feet, and yards simultaneously. We have already covered creating a new project in the earlier chapters, hence we will keep it very short here. Let's create a new project by navigating to **File** | **New** | **Others** | **Android Application Project**. Enter the fitting data from the following table in the corresponding wizards:

Property	Value
Application name	`DistanceCon`
Project Name	`DistanceCon`
Package Name	`com.packt.ch05.distancecon`
Template	`BlankActivity`
Activity	`MainActivity`
Layout	`activity_main`

The following screen shows some data being filled in the wizard as per the preceeding table:



Adding a RadioGroup, RadioButton, and a TextField

Android SDK provides two types of radio controls to be used in conjunction, where only one control can be chosen at a given time. RadioGroup (`android.widget.RadioGroup`) is used to encapsulate a set of RadioButton controls for this purpose.

Before we add the RadioGroup and RadioButton control, let's add the label `Distance` and the TextField to allow users to provide inputs. Open the `activity_main.xml` file, and add following entries:

```
<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="44dp"
      android:text="@string/distance "
      android:textAppearance="?android:attr/textAppearanceMedium" />
  <EditText
      android:id="@+id/distText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignBaseline="@+id/textView1"
      android:layout_alignBottom="@+id/textView1"
      android:layout_toRightOf="@+id/textView1"
      android:ems="10"
      android:inputType="numberDecimal|numberSigned" />
```

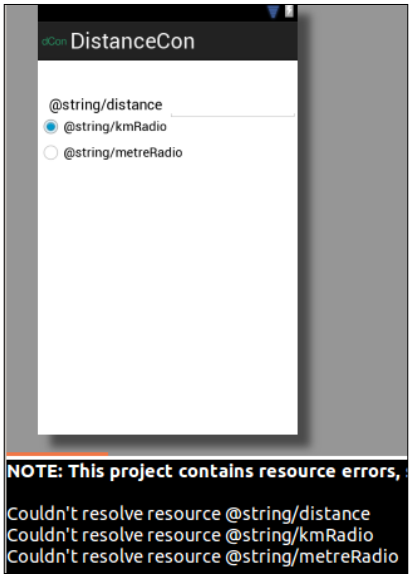
Let's get back and add the RadioGroup and RadioButtons in it. Add the following entries to the same file:

```
<RadioGroup android:id="@+id/distanceRadioGp"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_below="@+id/distText">
      <RadioButton android:id="@+id/kmRadiobutton"
          android:layout_height="wrap_content"
          android:layout_width="wrap_content"
          android:checked="true"
          android:text="@string/kmRadio">
```

```
            </RadioButton>
            <RadioButton  android:id="@+id/metreRadioButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/metreRadio">
            </RadioButton>
    </RadioGroup>
```

We have made `android:checked="true"` to be checked by default. After this step we would see some errors, don't worry about them as we are yet to define these strings.

The following screenshot is what we may see after adding the preceeding code in the XML file:



Adding a CheckBox

We will use CheckBox to allow users to have a conversion facility available for multiple types of conversions, at once. To add a CheckBox, add the following code in `activity_main.xml`. We will have three checkboxes for each: Mile, Foot, and Yard; the same can be achieved using:

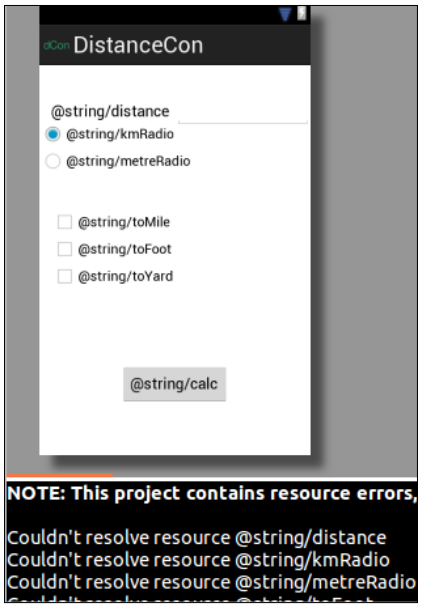
```
    <CheckBox
            android:id="@+id/checkBoxFoot"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/checkBoxMile"
        android:text="@string/toFoot">
</CheckBox>
<CheckBox
        android:id="@+id/checkBoxYard"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/checkBoxFoot"
        android:layout_below="@+id/checkBoxFoot"
        android:text="@string/toYard">
</CheckBox>
<CheckBox
        android:id="@+id/checkBoxMile"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/checkBoxFoot"
        android:layout_below="@+id/distanceRadioGp"
        android:layout_marginTop="40dp"
        android:text="@string/toMile">
</CheckBox>
```

Also add a button, such that upon clicking on it the conversion kicks off:

```
<Button
        android:id="@+id/calButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="60dp"
        android:onClick="onClick"
        android:text="@string/calc">
</Button>
```

The resulting screen should appear as follows:



Adding a menu

We will invoke the `Preference` screen from the menu. There are essentially three different types of menus available: Options menu, Context menu, and Pop up Menu. Here, we will use the Options menu for our purpose. To add the menu under `res/menu` create a new file named `prefsetting.xml`. Add the menu item, using the `<item></item>` element by adding the following code:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    <item android:id="@+id/menusettings"
        android:showAsAction="never"
        android:title="Preferences"
        android:orderInCategory="100">
    </item>
</menu>
```

The name of the menu item is set as `android:title="Preferences"`. The `android:showAsAction` keyword indicates how an item should appear in the action bar. For more menu options and attributes please refer to the following URLs:

- `http://developer.android.com/guide/topics/ui/menus.html`
- `http://developer.android.com/guide/topics/resources/menu-resource.html`

Defining the Strings

Under the **res/values** tab, open `strings.xml` and add the following entries:

```
<string name="menu_settings">Settings</string
    <string name="distance ">Distance</string>
    <string name="kmRadio">Km</string>
    <string name="metreRadio">Metre</string>
    <string name="calc">Calculate</string>
    <string name="toMile">Mile</string>
    <string name="toFoot">Feet</string>
    <string name="toYard">Yard</string>
```

After this step all the previous verbose errors should disappear.

Defining the Preference screen

Preferences are an important aspect of the android applications. It allows users to have the choice to modify and personalize it. Preferences can be set two ways: the first method is to create the `preferences.xml` file in the `res/xml` directory and the second method is to set the preferences from the code. We will use the former, also the easier one, by creating the `preferences.xml` file as follows:

Create the `xml` directory, if it does not exit, and add the `preferences.xml` file. Every preference needs the following attributes, as shown in the table:

Property	Description
`android:key`	Used to get the preference value
`android:title`	To specify the android title
`android:summary`	Summary about preferences
`android:defaultValue`	Optional, used to set the default values

Usually, there are five different preference views, as listed in the following table:

Views	Description
`CheckBoxPreference`	Simple checkbox returns true/false
`ListPreference`	Shows RadioGroup, only 1 item selected
`EditTextPreference`	Shows dialog box edit TextView, returns String
`RingTonePreference`	RadioGroup that shows ringtone
`PreferenceCategory`	Is a category with preferences

We will make use of `CheckBoxPreference`, `ListPreference`, and `PreferenceCategory` in our application. Let's add these preferences view in the `preferences.xml` file we have created. Add the following entries:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/
android" >
  <PreferenceCategory  android:title="Set Default Converison ">
      <CheckBoxPreference  android:title="@string/convertToMile"
          android:key="inputUserMile"
          android:summary="@string/summaryMile"
          android:defaultValue="false">
      </CheckBoxPreference>
      <CheckBoxPreference  android:title="@string/convertToYard"
          android:key="inputUserYard"
          android:summary="@string/summaryYard"
          android:defaultValue="false">
      </CheckBoxPreference>
      </PreferenceCategory>
      <CheckBoxPreference  android:title="@string/convertToFeet"
          android:key="inputUserFt"
          android:summary="@string/summaryFt"
          android:defaultValue="false">
      </CheckBoxPreference>
      <PreferenceCategory android:title="@string/prefInputType">
      <ListPreference android:title="@string/inputTypeList"
          android:key="inputTypeKey"
          android:summary="@string/userInputSummary"
          android:entries="@array/inputEntry"
          android:entryValues="@array/inputValues">
      </ListPreference>
      </PreferenceCategory
</PreferenceScreen>
```

This will result in spitting a lot of errors, however we will now solve this by defining strings. `ListPreference` provides a list and allows the selection of only one item, and hence, contains `android:entries`, and `android:entryValues` takes array. Now we will provide an array declaration for the same, to do that under `res/values`, if it does not exist, create the file `arrays.xml` and add the following entries:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="inputEntry">
        <item >Distance in Km</item>
        <item >Distance in Metre</item>
```

```
        </string-array>
        <string-array name="inputValues">
            <item >1</item>
            <item >2</item>
        </string-array>
    </resources>
```

Define the following strings that are used in the `preferences.xml` file in the `strings.xml` file.

```
    <string name="prefInputType">Set Default Input Type</string>
        <string name="userInputSummary">Distance  provided for
          calculation</string>
        <string name="convertedSummary">Summary of Conversion</string>
        <string name="convertToMile">Mile</string>
        <string name="convertToYard">Yard</string>
        <string name="convertToFeet">Foot</string>
        <string name="summaryMile">Convert to  Mile</string>
        <string name="summaryYard">Convert to Yard</string>
        <string name="summaryFt">Convert to Feet</string>
        <string name="inputTypeList">Choose default  distance supplied
          </string>
```

Now that we are done defining the Preference screen, let's do some work to show it. The Preference framework comes with the activity class `android.preference.PreferenceActivity` needs to be overridden with our class. Create a class `UserSettings.java` under the `com.packt.ch05.distnacecon` package and write the following code:

```
    package com.packt.ch05.distancecon;
    import android.os.Bundle;
    import android.preference.PreferenceActivity;

    public class UserSettings extends PreferenceActivity {

    @Override
      public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }
    }
```

`addPreferencesFromResources()` loads the Preference screen from the `preferences.xml` file.

Hook up

After doing all the hard work of defining and putting things in place, let's get in to do some action by hooking up everything with the main screen (Main Activity). Open the `MainActivity.java` file and let's binds things in now.

Initialize the widgets as follows:

```
private EditText text;
private RadioButton rBtnKm;
private RadioButton rBtnMtr;
private CheckBox cBoxMile;
private CheckBox cBoxFt;
private CheckBox cBoxYd;
```

The `onCreate` method is first called to fetch the instances of widgets as follows:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    text= (EditText)findViewById(R.id.distText);

rBtnKm=(RadioButton)findViewById(R.id.kmRadiobutton);
rBtnMtr= (RadioButton )findViewById(R.id.metreRadioButton);
cBoxMile = (CheckBox) findViewById(R.id.checkBoxMile);
cBoxFt = (CheckBox) findViewById(R.id.checkBoxFoot);
cBoxYd = (CheckBox) findViewById(R.id.checkBoxYard);
}
```

Binding the menu and Preference

We specify our earlier defined menu from the resources file `prefesetting.xml`, by `getMenuInflater().inflate(R.menu.prefsetting, menu)` command as follows:

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

On the menu item select the `override` method as follows:

```
public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.menusettings:
          //Get  the intent Preference Activity
          Intent i = new Intent(this, UserSettings.class);
            //Start the intent  and return the result
            startActivityForResult(i, 1);
            break;
        }
        return true;
    }
```

`onActivityResult` is called receiving the result from the following code, so perform the operation needed here:

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
          super.onActivityResult(requestCode, resultCode, data);

          switch (requestCode) {
          case  1:
              showPreferenceSettings();
              break;
          }
      }
```

Getting values from Preferences

Now, we want to reflect the value set in the Preference screen onto the main screen to show personalization.

We get the values from the Preference screen and set it back to the main screen in `showPreferenceSettings()`. We get the preferences values via `PreferenceManager`.

```
private void showPreferenceSettings(){
    SharedPreferences sharedPrefs =
      PreferenceManager.getDefaultSharedPreferences(this);

    if(sharedPrefs.getBoolean("inputUserMile", false))
        cBoxMile.setChecked(true);
    if(sharedPrefs.getBoolean("inputUserYard", false))
```

```
            cBoxYd.setChecked(true);
        if(sharedPrefs.getBoolean("inputUserFt", false))
            cBoxFt.setChecked(true);
    }
```

On clicking the **Calculate** button, the conversion should happen and the result should be shown. To show the result we make use of the ToastView command here.

The `onClick` function is called when the button is clicked, we then get the RadioButton values and the checked CheckBox values and call the corresponding convert functions which is then shown via ToastView with the following code:

```
public void onClick(View view ){
    StringBuffer dist =new StringBuffer();
    switch (view.getId()){
    case R.id.calButton:
      if(text.getText().length()==0){
        Toast.makeText(this, "Please enter the valid number ",
          Toast.LENGTH_LONG).show();
        return ;
      }

        double distValue=Double.parseDouble
          ((text.getText().toString()));
      //Find  RadioButton is checked
      if(rBtnKm.isChecked()){
      //Find checkBox is checked
        if(cBoxMile.isChecked()){
          double km=convertKmToMile(distValue);
          dist.append(km+"Mile.");
        }
        if(cBoxYd.isChecked()){
          double yd=convertkmToYard(distValue);
          dist.append("   "+yd+"yard.");
        }
        if(cBoxFt.isChecked()){
          double ft=convertkmToFoot(distValue);
          dist.append("   "+ft+"ft.");
        }

        Toast.makeText(this,dist,Toast.LENGTH_SHORT).show();
      }
```

```
        if(rBtnMtr.isChecked()){
          if(cBoxMile.isChecked()){
            double km=convertMToMile(distValue);
            dist.append(km+"Mile.");
          }
          if(cBoxYd.isChecked()){
            double yd=convertMtoYard(distValue);
             dist.append("   "+yd+"yard.");
           }
           if(cBoxFt.isChecked()){
             double ft=convertMtoFoot(distValue);
             dist.append("   "+ft+"ft.");
           }
           Toast.makeText(this,dist,Toast.LENGTH_SHORT).show();
         }
         return;
       }

    }
```

Let's add the conversion method for each type as follows:

```
    private double convertKmToMile(double distance ){
      return (distance*0.62137);
    }

    private double convertkmToYard(double distance){
      return distance*1093.6;
    }
```

Add the other conversion method for the others as well.

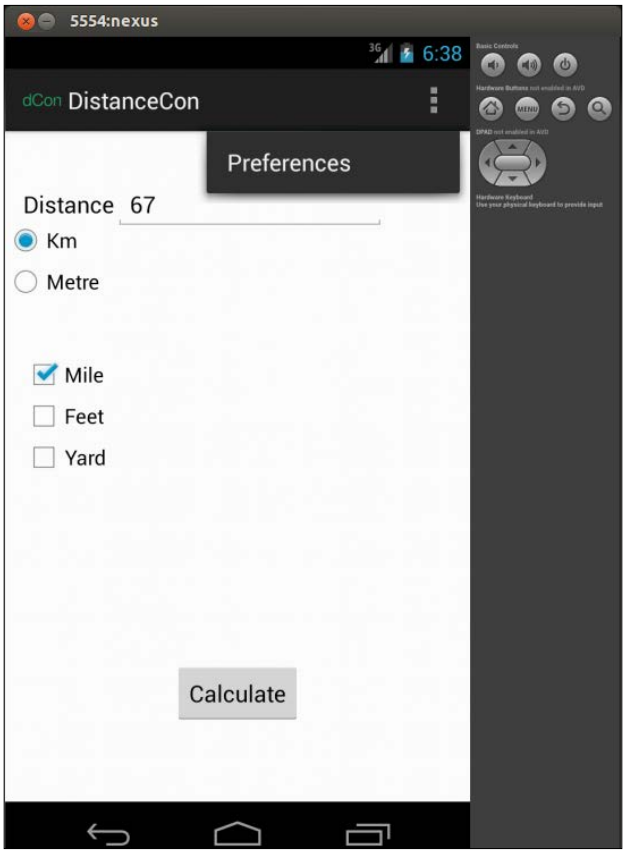
Finally, add the following tag which denotes an activity in the `AndroidManifest.xml` file.

```
<activity android:name=".UserSettings" />
```

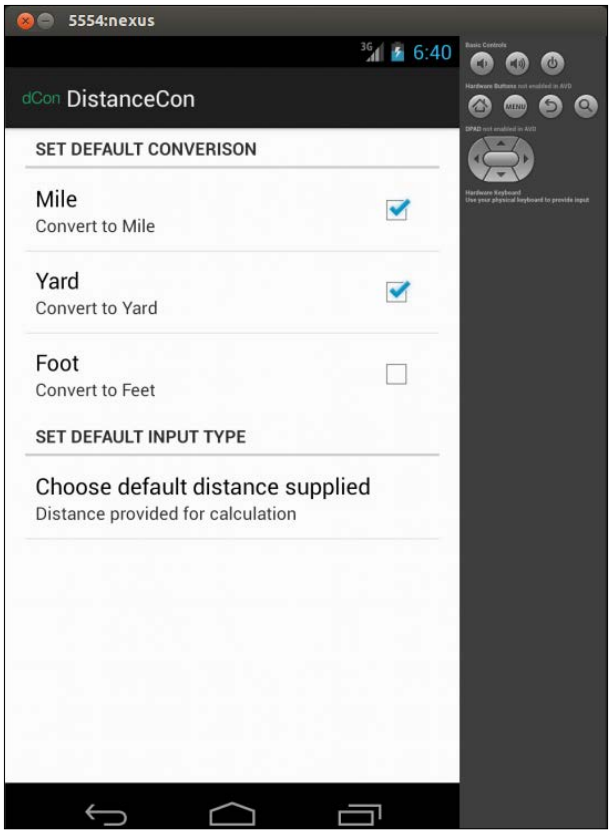
The complete code and resources are available in a downloadable source code.

Run the application

When we run the application, the following screen should appear where the first screen accepts the input and the output appears as ToastView popup on clicking the **Calculate** button:



The following screenshot shows the Preference screen:



Summary

In this chapter we have learned about how to get going with widgets, such as CheckBox, RadioButton together with RadioButton, menu, and creating custom Preferences view and getting values from it. Also, using these concepts we have created the `DistanceConverter` application.

In the next chapter, we will learn how to handle the various screen types and orientations for this application.

6
Handling Multiple Screen Types

Android devices are available in different shapes and sizes. For a wider audience, handling multiple screen types across different devices is the key. In this chapter we will learn about catering to different screen orientation changes and different screen types. We will make use of the DistanceConverter application discussed earlier, and make changes to cater to different concepts needed to achieve this:

- Adapting to different screens using `wrap_content` and `match_parent`
- Introducing Fragment
- Defining Fragment and Landscape layout
- Hook up in the Main Layout file
- Running the application
- Optimizing for tablet
- Persisting the state information during the state transition

We will use the DistanceConverter application from a previous chapter and use fragment to define layouts for landscape, and adapt to different screen orientations and types.

Using wrap_content and match_parent

In order to cater to the need of a variety of android devices available in the market, the application needs to be compatible to different screen sizes. For example, a layout should adapt to different screen sizes, and the corresponding views should also resize accordingly. To ensure that we make use of `wrap_content` and `match_parent` for width and height of view components refer to the following:

- `wrap_content`: It ensures that the width and height of the view is set to the minimum size required to fit the content
- `match_parent`: Before API level 8, it was known as `fill_parent` and it ensures the component expands to match the size of its parent view

Therefore, use of these attributes affirms our views to use the space required and expands to fill the available space. We have made use of these in the DistanceConverter application for components in layout file. Following is a small code snippet from `activity_main.xml`, our previous application to demonstrate its usage:

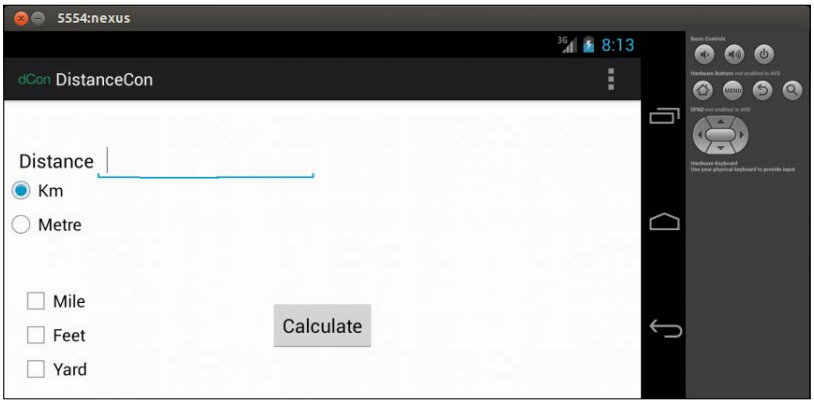
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >
  <RadioGroup android:id="@+id/distanceRadioGp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/distText">
</RelativeLayout>
```

Fragment

A **Fragment** is an independent component that can be connected to an Activity or simply is a subactivity. Typically it defines a part of UI but can also exist with no user interface, that is, headless. An instance of fragment must exist within an activity.

Fragments ease the reuse of components for different layouts. Fragments are the way to support UI variances across different types of screens. The most popular use is for building single pane layouts for phones and multipane layouts for tablets (large screens). Fragment was introduced in Android 3.0 API 11. Fragment can also be used for supporting different layouts for portrait and landscape orientations.

A fragment stops as activity stops, and is destroyed as activity is destroyed. The `OnCreateView()` method is where the view UI is created via the `inflate()` method call. Following is the screenshot of our application in landscape orientation from our previous code:



We will make use of fragment to define a landscape layout for our DistanceConverter application in the proceeding chapter.

Defining Fragment and Landscape layout

Let's make changes in the layout for Landscape mode. To support different layouts for landscape mode, create a folder `layout-land` in the `res` folder. Create a file `activity_main.xml` under it and add following code:

```
<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"
  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="14dp"
      android:layout_marginTop="44dp"
      android:text="@string/distance "
      android:textAppearance="?android:attr/textAppearanceMedium"
/>
```

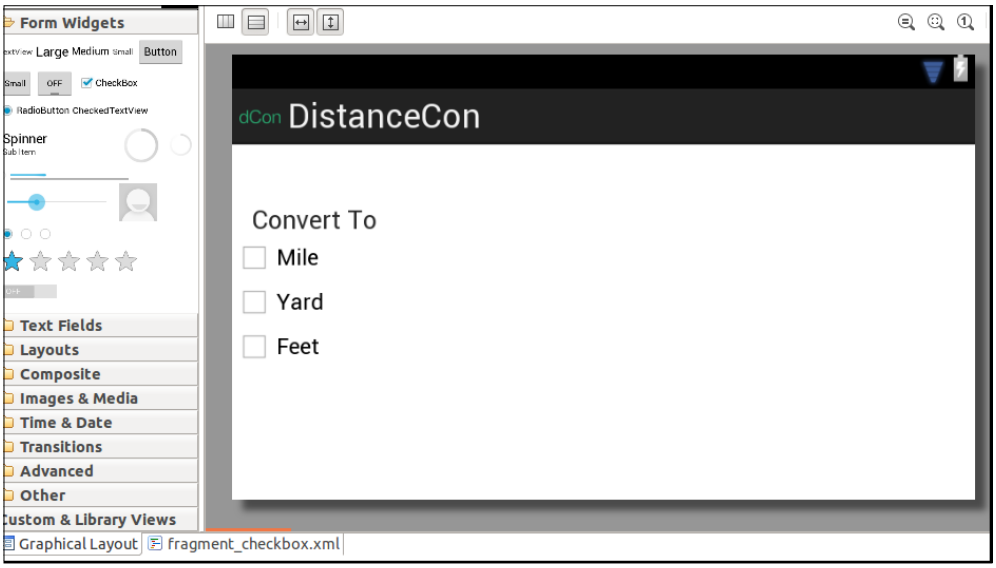
```
    <EditText
      android:id="@+id/distText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignBaseline="@+id/textView1"
      android:layout_alignBottom="@+id/textView1"
      android:layout_toRightOf="@+id/textView1"
      android:ems="10"
      android:inputType="numberDecimal|numberSigned" />
    <RadioGroup android:id="@+id/distanceRadioGp"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_below="@+id/distText"
    <RadioButton android:id="@+id/kmRadiobutton"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:checked="true"
      android:text="@string/kmRadio">
    </RadioButton>
    <RadioButton android:id="@+id/metreRadioButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/metreRadio">
    </RadioButton>
    </RadioGroup>
    <Button
      android:id="@+id/calButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="60dp"
      android:onClick="onClick"
      android:text="@string/calc"
    </Button>
</RelativeLayout>
```

Create a file `fragment_checkbox.xml` under the same folder to define the UI for fragment. Add the following code in it:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
  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:layout_marginLeft="14dp"
      android:layout_marginTop="44dp"
      android:text="@string/convertTo"
      android:textAppearance="?android:attr/textAppearanceMedium" />
    <CheckBox
      android:id="@+id/checkBoxMile"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/toMile" />
    <CheckBox
      android:id="@+id/checkBoxYard"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/toYard" />
    <CheckBox
      android:id="@+id/checkBoxFoot"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/toFoot" />
</LinearLayout>
```

Fragment layout from the preceding code is as shown in the following screenshot:



After putting down layout of fragments let's define fragment by extending the `android.app.Fragment` class. Let's create a fragment class `ConvertToFragment` with the following code:

```
@TargetApi(Build.VERSION_CODES.HONEYCOMB)
public class ConvertToFragment extends Fragment{
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup
container,Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_checkbox,
      container, false);
    return view;
  }
}
```

As fragment is available in the Android 3.0 (also known as API 11), we have put `@TargetApi(Build.VERSION_CODES.HONEYCOMB)` at the top. For devices at lower API level, fragments will not be available, in that case we have to define and arrange views in `activity-main.xml` under the `res/layout-land` folder.

For the compulsive use of fragments in lower API level, use Support Libraries which is a JAR file that allows us to use the most recent Android APIs. For more information, refer to the `http://developer.android.com/training/basics/fragments/support-lib.html`.

In the `onCreateView()` method we inflate the view from XML via the `inflate()` method.

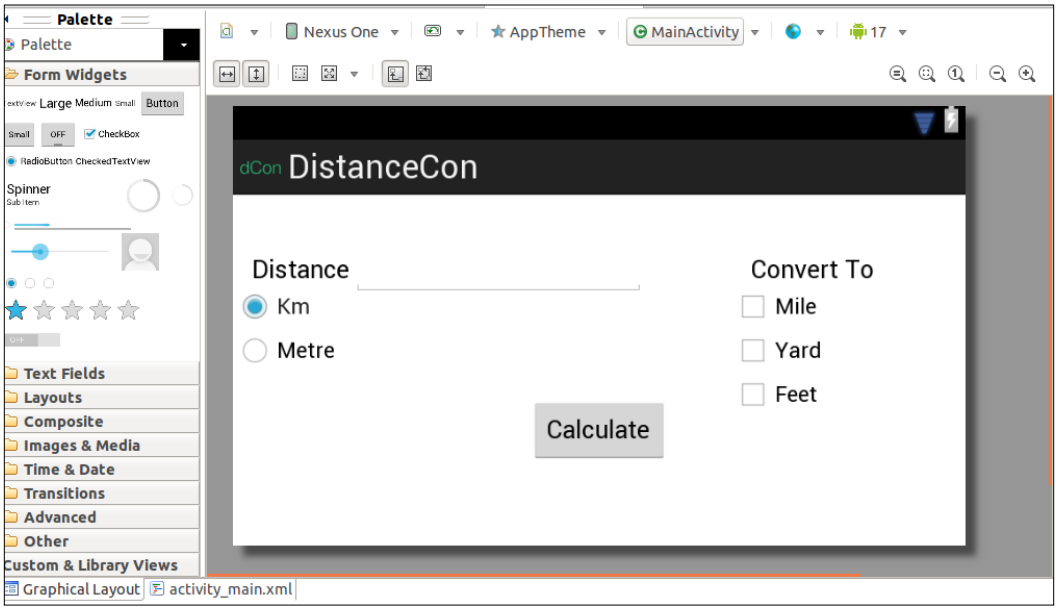
Hook up in the Main Layout file

Open the `activity_main.xml` file in `res/layout-land` and append following code:

```
<fragment
  android:id="@+id/convertToCheckBox"
  android:layout_width="wrap_content"
  android:layout_height="match_parent"
  android:layout_alignParentTop="true"
  android:layout_marginLeft="45dp"
  android:layout_toRightOf="@+id/calButton"
  class="com.packt.ch05.distancecon.ConvertToFragment"
  tools:layout="@layout/fragment_checkbox" />
```

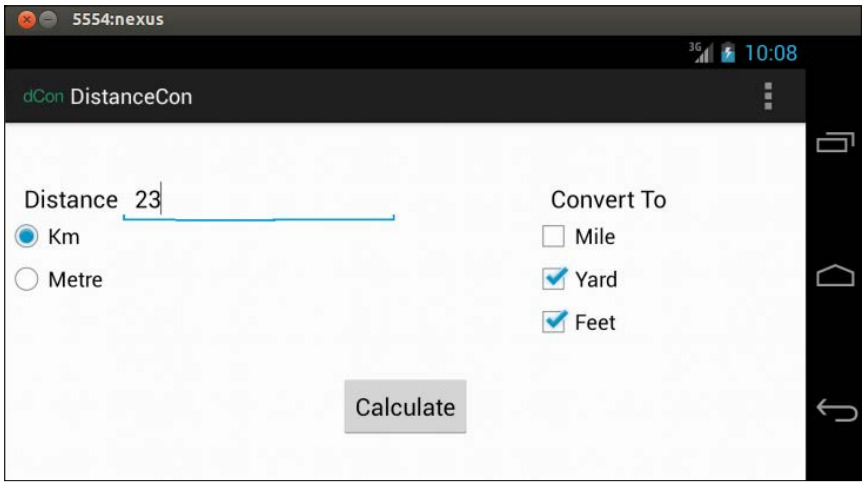
The `class` points to the corresponding fragment class. The `tools:layout` points to the layout for the corresponding fragment.

After the preceding step, the graphical layout screen should look like the following screenshot:

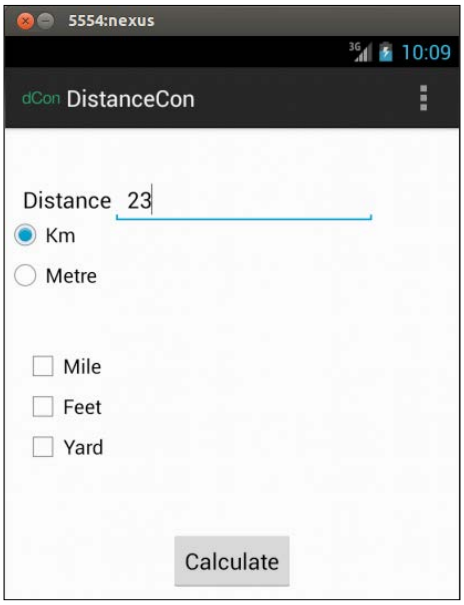


Running the application

Now that we are done with all of the programming, let's check out how our final application will look. The application in landscape mode is depicted in the following screenshot:



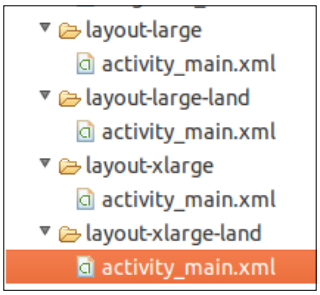
The application in the portrait mode is depicted in the following screenshot:



Use *Ctrl+F11* to change screen mode from portrait to landscape and vice versa in the emulator.

Optimizing for tablet

Tablet is another emerging Android device in the present context. We should also define layouts to support tablet devices. To cater to tablet devices, or so called large devices, we need to have another set of layouts defined under the folder `res/layout-xlarge` (for the portrait mode) and `layout-xlarge-land` (for landscape mode).The following snapshot shows the folders and files for defining the layouts for larger devices (tablets):



Once we have created the corresponding folder, we can make use of fragments as demonstrated previously, to create different layouts and achieve the goal of supporting tablets.

Persisting the state information during the state transition

You must have observed that the state of checkboxes are not persisted after screen mode changes from landscape to portrait and vice versa. This is a very important concept that we should be aware of. For every screen orientation change, the activity is destroyed, and then recreated. The `onCreate()` method is called and hence, the current state of the activity is lost. We need to save the state using the `onSaveInstanceSate` method and get it back with the `onRestoreInstanceState` method. So let's override these methods to achieve this with the following code:

```
@Override
  public void onSaveInstanceState(Bundle outState)
  {
    //---save whatever you need to persist—
    outState.putBoolean("mileChecked",cBoxMile.isChecked());
    outState.putBoolean("ydChecked",cBoxYd.isChecked());
    outState.putBoolean("ftchecked",cBoxFt.isChecked());
    super.onSaveInstanceState(outState);
  }
@Override
  public void onRestoreInstanceState(Bundle savedInstanceState)
  {
    super.onRestoreInstanceState(savedInstanceState);
    //---retrieve the information persisted earlier---
    cBoxFt.setChecked(savedInstanceState.getBoolean("ftchecked"));
    cBoxMile.setChecked(savedInstanceState.getBoolean
      ("mileChecked"));
    cBoxYd.setChecked(savedInstanceState.getBoolean("ydChecked"));
  }
```

For the complete source, go to `http://www.packtpub.com/support`. For more information on handling different screen types, refer to the following URLs:

- `http://developer.android.com/training/multiscreen/screensizes.html`
- `http://developer.android.com/distribute/googleplay/quality/tablet.html`

Summary

In this chapter, we learned about fragment and its usage, and used it to have different layouts for landscape mode for our application DistanceConverter. We also learned about handling different screen types and persisting state during screen mode changes. In the next chapter, we will learn about adding an external library, for example, `AdMob`, and incorporate advertisements in the application.

7
Adding an External Library

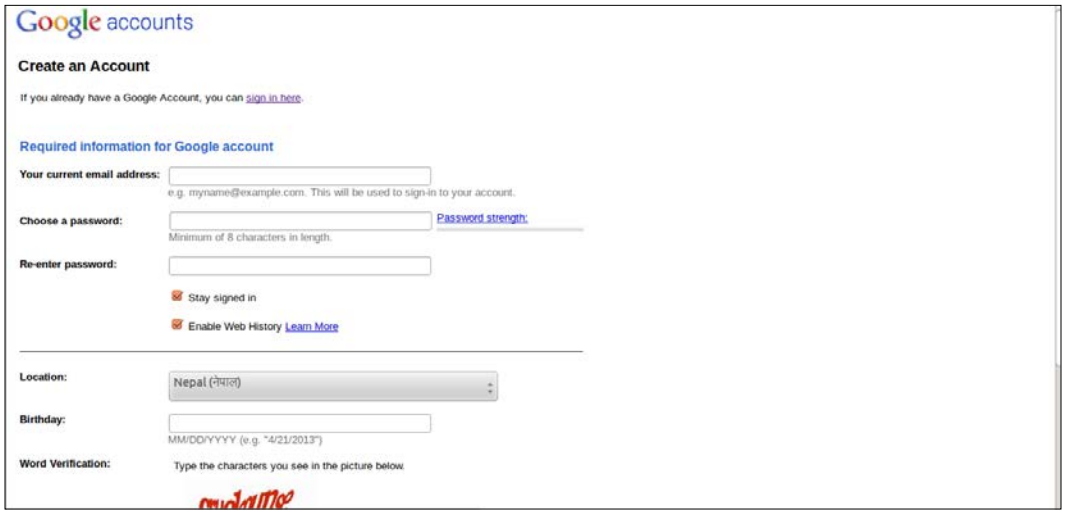
An Android application cannot achieve everything on its own, it will always need the company of external jars/libraries to achieve different goals and serve various purposes. Almost every free Android application published on store has advertisements embedded in it, which makes use of external components to achieve it. Incorporating advertisements in the Android application is a vital aspect of today's application development. In this chapter, we will continue on our DistanceConverter application developed from the previous chapters, and make use of an external library, `AdMob`, to incorporate advertisements in our application. The coverage will include the following:

- Creating an account at the `AdMob` site
- Adding Site/Application
- Adding the Advertisement Meditation Network
- Adding `AdMob` in the application
- Making changes in the manifest file
- Adding the `AdMob` widget/view in the layout file
- Running the application

Creating an account at the AdMob website

`AdMob` is one way to incorporate advertisements in our Android application. To make use of `AdMob`, the first thing we need to do is to register and get an account for ourselves. To register, visit the `http://www.admob.com` website and register on it. On the right-hand side, click on **Sign up with AdMob**, and then fill up the form and register.

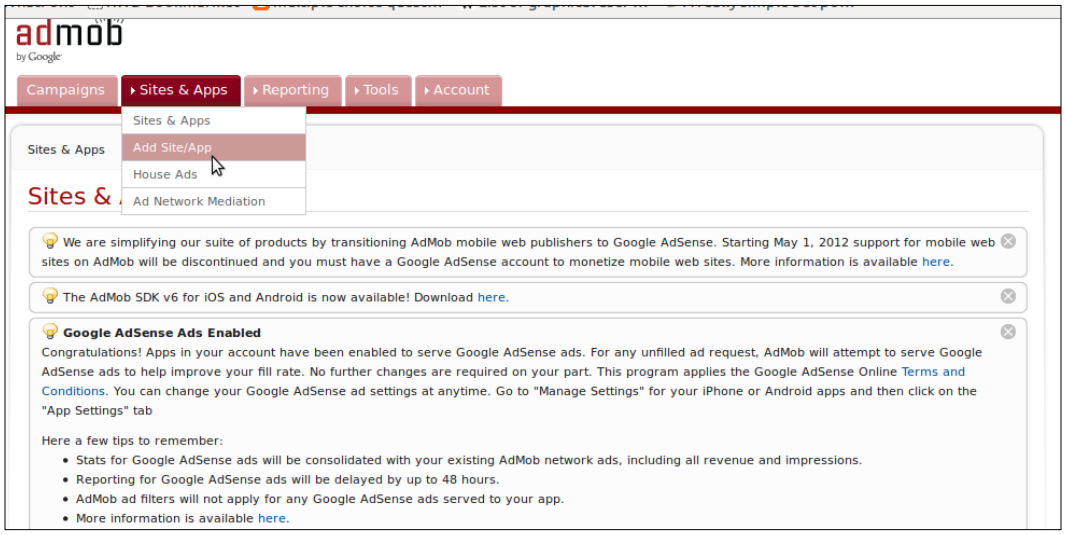
The following screenshot shows the sign up form:



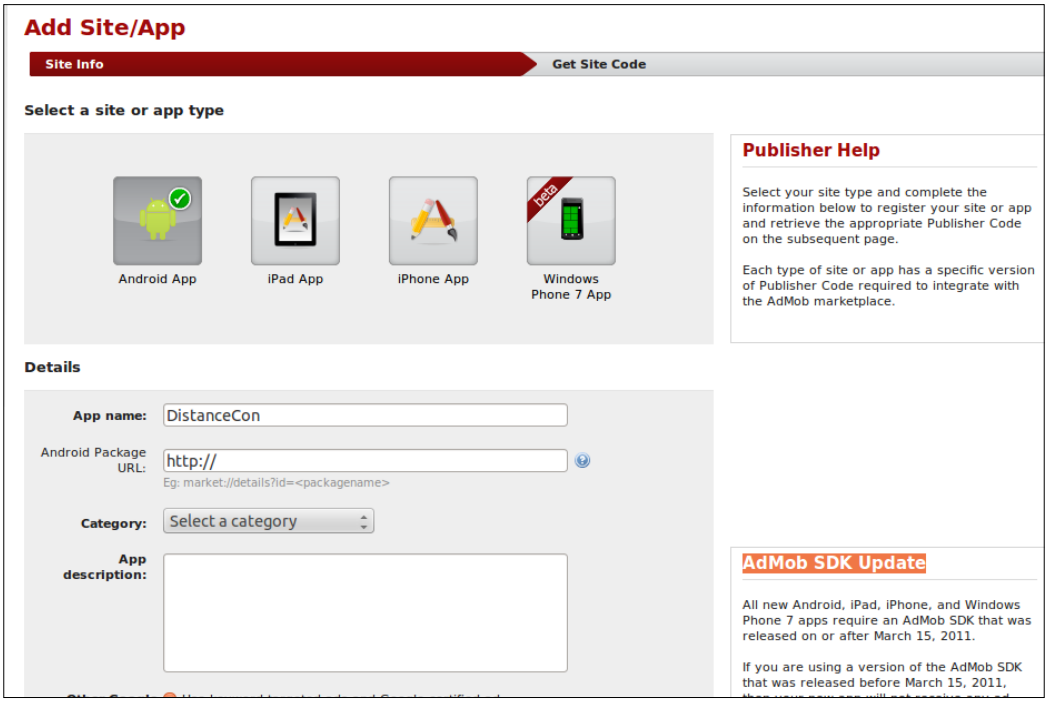
We can use our existing Google ID if we have, else the preceding steps will create one and link it with the `AdMob` account.

Adding Site/Application

Once we have created our account, we need to add a Site/Application (basically, it identifies or acts as unique handle for ads networks for the ads they place). To add Site/Application we perform the following steps:

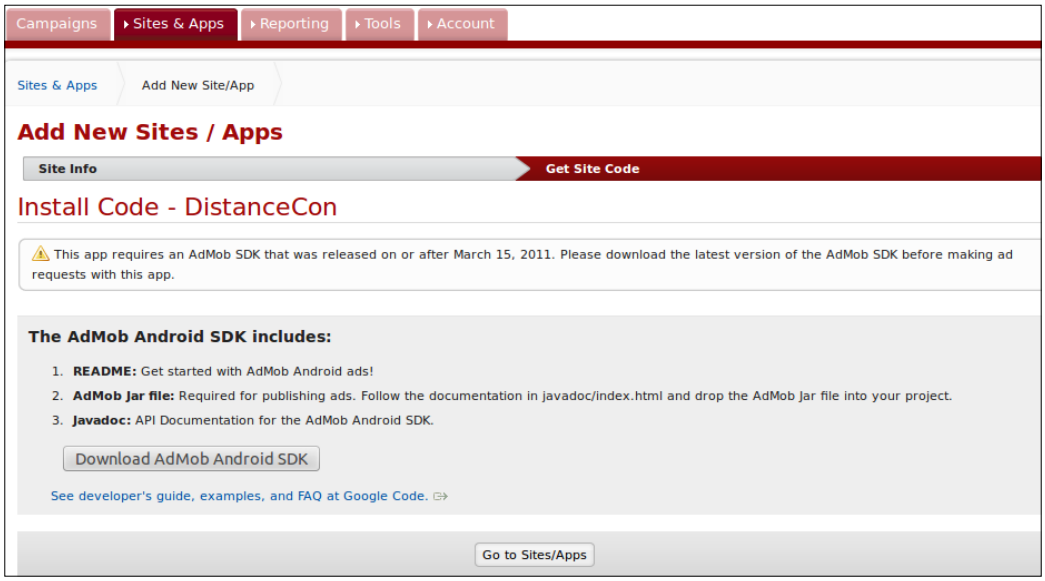


1. Navigate to **Add Site/App** from the **Sites & Apps** menu, as shown in the preceding screenshot. The **Add Site/App** screen will appear, as shown in the following screenshot:

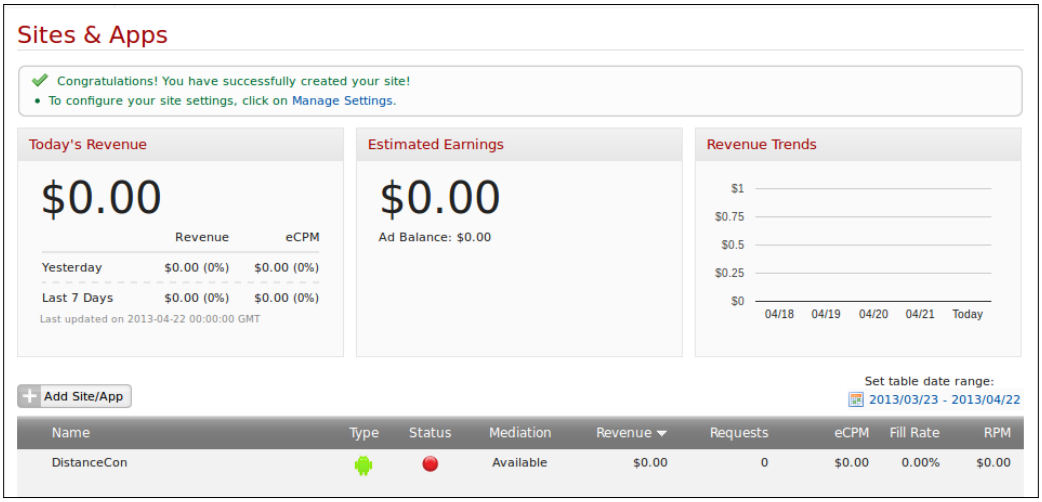


2. Select **Android App**, as shown in the preceding screenshot and fill in the other details. Because our application is not in the market place, use **http://** for **Android Package URL**, as shown in the preceding screenshot.

3. Select the corresponding category, in this case we used **Tools**, and add some description in the **App description** textarea. Also, leave the other fields to their default, and enter the captcha and create site. After this the following screen will appear:



4. Next, click on the **Download AdMob Android SDK** button to download the `AdMob` SDK. Once the SDK is downloaded, click on the **Go to Sites/App** button and our site should have been added, and will appear in the sites list as shown in the following screenshot:



5. The **Status** appears to be red as it has not received any ad request for this site. It will automatically turn green once it starts getting ad requests for this site.

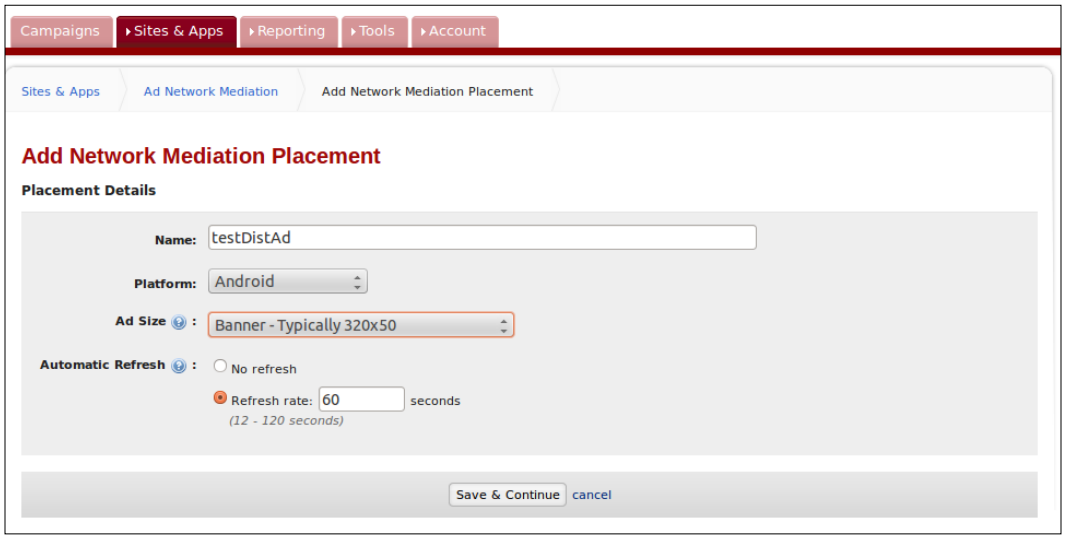
Choosing the Ad Network Mediation

Once we are done with adding the Site/Application and downloading the SDK lets get into adding **Ad Network Mediation** (**AdMob Mediation**). It coordinates with the different ad networks to help us maximize fill rate (represents the percentage of ad requests that satisfy the ad requests sent by the app) and increase monetization. It ensures that a proper network is selected to serve the ads at any time. For more information on AdMob Mediation, please refer to the following URL:

`https://support.google.com/admob/topic/2403413?hl=en&ref_topic=1307209`

To add the Ad Network Mediation, follow the given steps:

1. Navigate to the **Ad Network Mediation** under the **Sites & Apps** menu, and follow the steps, as shown in the following screenshot:

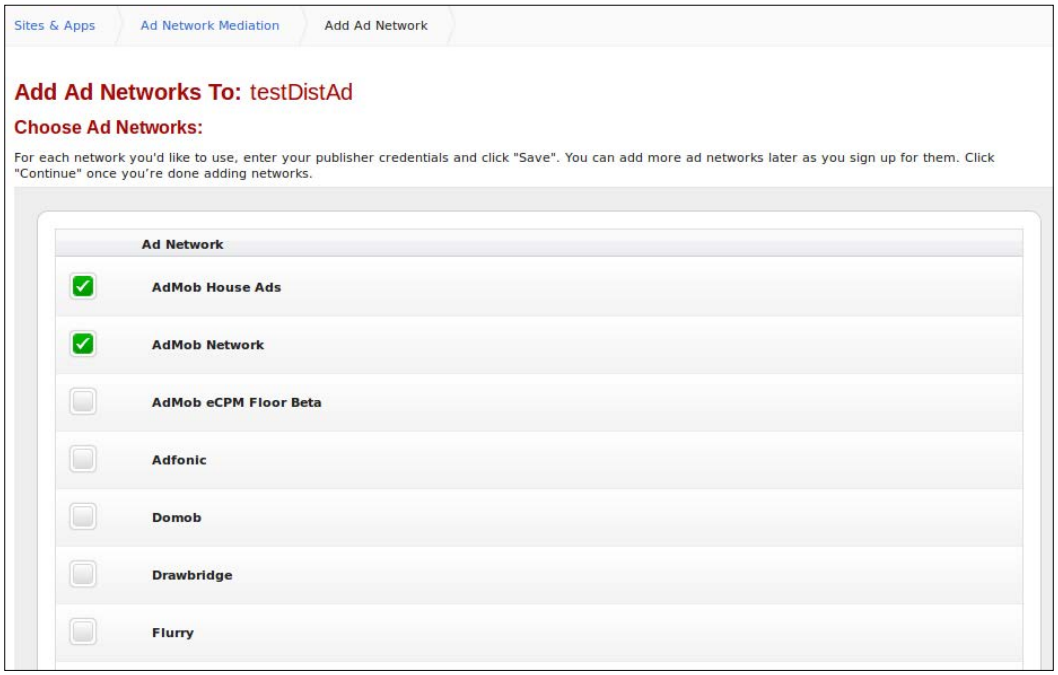


2. Select the **Ad Size** as **Banner - Typical 320x50** for support on most of the iPhones and Android phones in portrait, and **Platform** as **Android**.

   For more information on banner sizes and decision, refer to the following URL:

   `https://developers.google.com/mobile-ads-sdk/docs/admob/smart-banners`

3. Next, select **Automatic Refresh**, and then specify the **Refresh rate,** and then click on the **Save & Continue** button. The following screen will appear. Select **Ad Network** from it, and then click on **Continue** as depicted in the following screenshot:

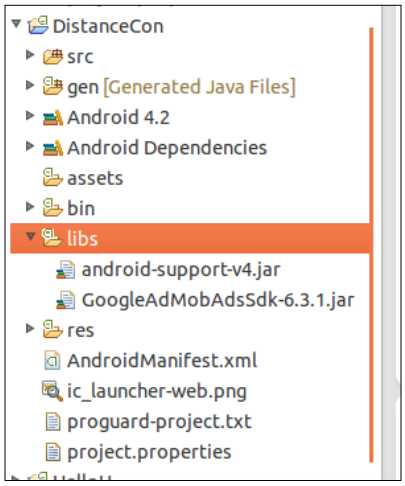


4. Choose the network you wish from the options in the preceding screenshot.

Publisher credentials are to be provided for the network we select at the bottom of the same screen. In this case, we have credentials for `AdMob` as we just signed up and we only chose **AdMob Network**, as shown in the preceding screenshot. However, we are free to add any number of networks, provided we have credential details. Also, we can always add any network at any point of time.

Adding AdMob SDK to the project

Let's extract the previously downloaded `AdMob SDK zip` file, and we should get the folder `GoogleAdMobAdsSdkAndroid-6.*.*`. Under that folder there is `GoogleAdMobAdsSdk-6.x.x.jar` file. Copy this JAR file in the `libs` folder of the project, as shown in the following screenshot:



Other Java libraries can be added in the same way for use in our project, and to reference Android libraries in the project, information is available at the following URL:

`http://developer.android.com/tools/projects/projects-eclipse.html`

Making changes in the manifest file

The `AdMob` needs to make request across the internet to fetch ads. Therefore, that permission needs to be added in the `AndroidManifest.xml` file as shown in the following code:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
  android:name="android.permission.ACCESS_NETWORK_STATE" />
```

In other words, it also helps the AdMob SDK to figure out a currently working Internet connection before it places requests.

Also, add the `AdView` activity which is responsible for getting and showing ads in the file, as shown in the following code:

```
<activity
  android:name="com.google.ads.AdActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenLayou
t|uiMode|screenSize|smallestScreenSize" />
```

For more information on integration, refer to the following URL:

`https://developers.google.com/mobile-ads-sdk/docs/`

Adding the AdMob widget/view in the layout file

To add the `AdMob` view, add the following code in the `layout/activity_main.xml` file for the portrait mode:

```
<com.google.ads.AdView
  android:id="@+id/adView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_alignParentBottom="true"
  ads:adSize="SMART_BANNER"
  ads:testDevices="TEST_EMULATOR"
  ads:adUnitId="a1516e8871e5b38"
  ads:loadAdOnCreate="true"/>
```

Similarly, add the same piece of code in the `layout-land/activity_main.xml` file for the landscape mode. After this addition, an error will be shown, and that is because we have not defined the namespace for `AdView`. We will do that next and the error will disappear.

Add the `meta` tag in the namespace at the top of the XML along with other namespaces:

```
xmlns:ads="http://schemas.android.com/apk/lib/com.google.ads"
```

Let's look at some of the important tags and the values of `AdView` that were used previously:

Item	Value
`ads:adSize`	`SMART_BANNER`: the banner adjusts according to the screen types and orientation using the width of screen.
`ads:testDevices`	It is used for testing whether the code is fine. `TEST_EMULATOR` is used for Emulator. Devices ID can also be specified if used for testing. It should be removed if moving to production from dev. The easiest way to find the device ID is from the `AdMob` SDK log output.
`ads:adUnitId`	Publisher ID. Replace with the corresponding ID.
`ads:loadAdOnCreate`	To create the view by inflating, and send ad request to `AdMob`.

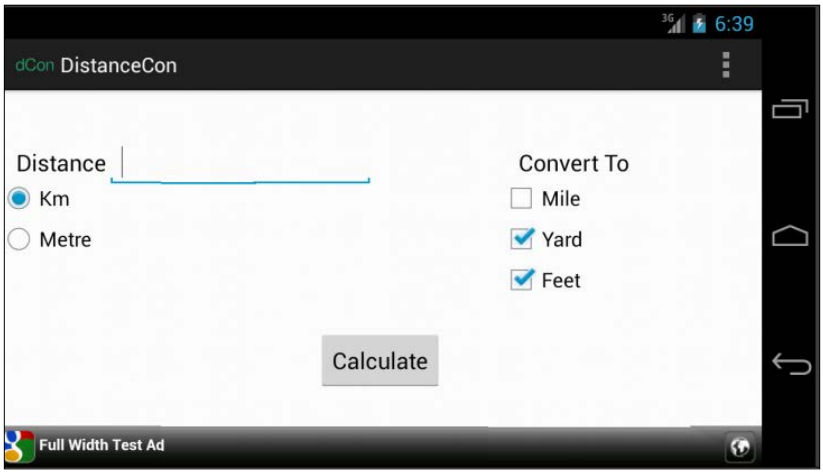
In the previous case we are loading `Adview` and making request via XML. There is another way to achieve this by placing the following code in the `MainActivity.java` file in the `onCreate()` method, as shown in the following code snippet:

```
adView = (AdView)findViewById(R.id.adView);
AdRequest re = new AdRequest();
re.setTesting(true);
adView.loadAd(re)
```

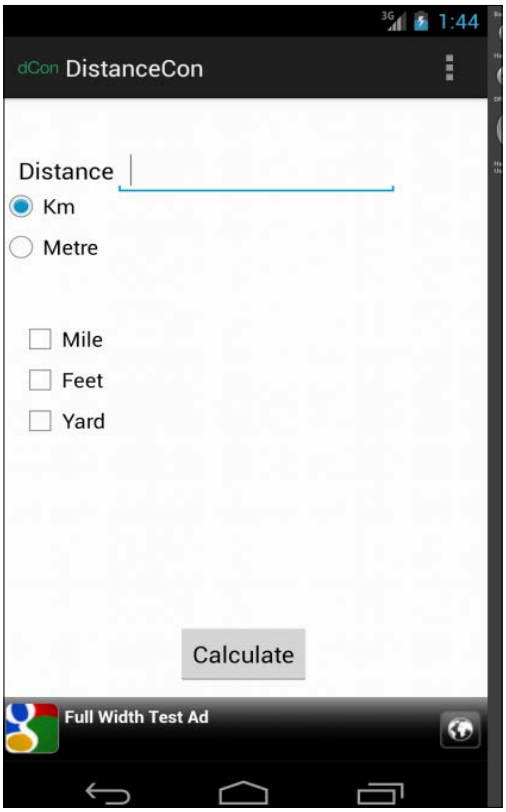
Make sure the testing mode is removed before the Android app gets ready to be published to the store.

Running the application

After all the hard work, let's run the application to check out how it looks. In the landscape mode, the advertisement would appear as shown in the following screenshot:



In the portrait mode, the ad will appear as shown in the following screenshot:



For the first time the `AdMob` ads may take 1 or 2 minutes to show, so have patience.

Summary

In this chapter, we learned how to add an external library by means of incorporating `AdMob` mobile advertisements in our DistanceConverter application.

In the next chapter, we will learn about what it takes to sign and get ready to publish the application.

8
Signing and Distributing APK

All the hard work done so far is not going to pay off unless we distribute our application for others to use. An Android application has to be signed before it goes on the radar for distribution. Any Android application, be it used in the emulator or distributed to friends, relative for testing, or published to Google Play store, needs to be signed electronically. In this chapter, we will learn about how to sign it and publish it for use by others. This chapter will cover the following:

- APK (Android package)
- Preparing for release
- Compilation for release
- Generating a private key
- Using the Eclipse ADT for release
- Publish to Google Play

APK – Android package

The **Android package** (**APK**), in simple terms, is similar to the runnable JAR or executable file (on Windows OS) which consists of everything that is needed to run the application.

The Android ecosystem uses a virtual machine, that is, **Dalvik virtual machine** (**DVM**) to run the Java applications. Dalvik uses its own bytecode, which is quite different from the Java bytecode.

A tool `dx` under Android SDK converts our Java classes to `.dex` (**Dalvik executable**).

The `.dex` files and resources of application (XML and images) are packaged by the tool **aapt** (**Android asset packing tool**) into the `.apk` file.

Preparing for release

After the hard work of coding and testing the application needs to be packaged for release. Packaging involves the following steps.

Compilation for release

This is the very first step towards release and distribution. It comprises of setting a package name in the application's manifest file, configuring application attributes, and compilation before release. They involve the following steps:

- **Choosing appropriate package name**: Once the application is released it cannot be undone hence, the need to dwell upon and choose a suitable package name. The package name can be set in the application's manifest file.
- **Disabling debugging**: We need to make sure we disable debugging before we release it. To disable debugging, comment or remove the `Log()` method call in the code. Also, debugging can be disabled by removing the `android:debuggable` attribute from the `<application>` tag.
- **Pointing out the application icon**: Every application needs to have an icon of itself. Please make sure that the icon follows the icon guidelines a: `http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html`. Icons can be specified by using the icon attributes of the `<application>` tag.

Versioning: This is the most important aspect of release and also maintenance. The version identifies the application's release build and determines how it should be updated. To put it in the simple terms, the version number must be incremented with each published release. With no version in place, it is rather impossible for future updates. The versioning information is provided by the following two attributes:

`android:versionCode`	It is the integer represents version of application.
`android:versionName`	It is the string that is displayed to users to identify what is installed in the device.

Both these attributes can be specified under the `<manifest>` element.

- **Review the manifest file for permissions**: It should only specify relevant permissions in the manifest file using the `<uses-permission>` tag.

Generating a private key

An android application must be signed with our own private key. It identifies a person, corporation, or entity associated with the application. This can be generated using the program `keytool` from the Java SDK. The following command is used for generating the key:

```
keytool -genkey -v -keystore <filename>.keystore -alias <key-name>
-keyalg RSA -keysize 2048 -validity 10000
```

We can use a different key for each published application, and specify a different name to identify it. Also, Google expects validity of at least 25 years or more. A very important thing to consider is to keep a back up and securely store the key, because once it is compromised it impossible to update an already published application.

Signing

After obtaining the private key we need to sign the application. This is done using a program `jarsigner` from the Java SDK. The following command is used:

```
jarsigner -verbose -sigalg MD5withRSA -digestalg SHA1 -keystore my-
release-key.keystore my_application.apk alias_name
```

Alignment

Once the APK is signed it needs to be optimized, to do that we use the `zipalign` tool available with the Android SDK under the `tools/` directory. The usage is as follows:

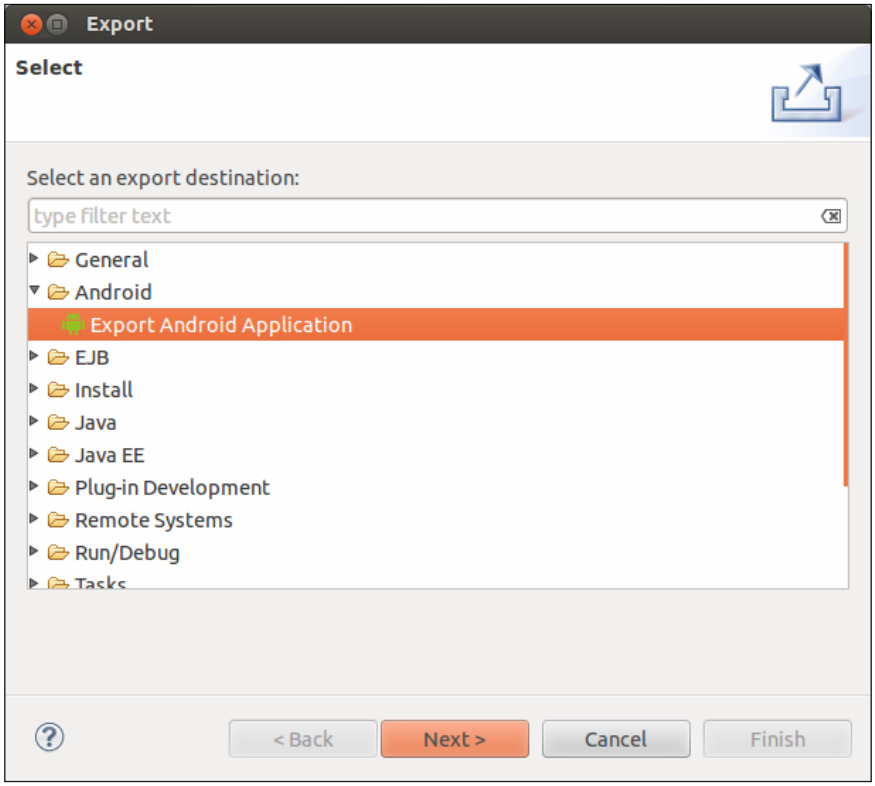
```
zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

Using the Eclipse ADT for release

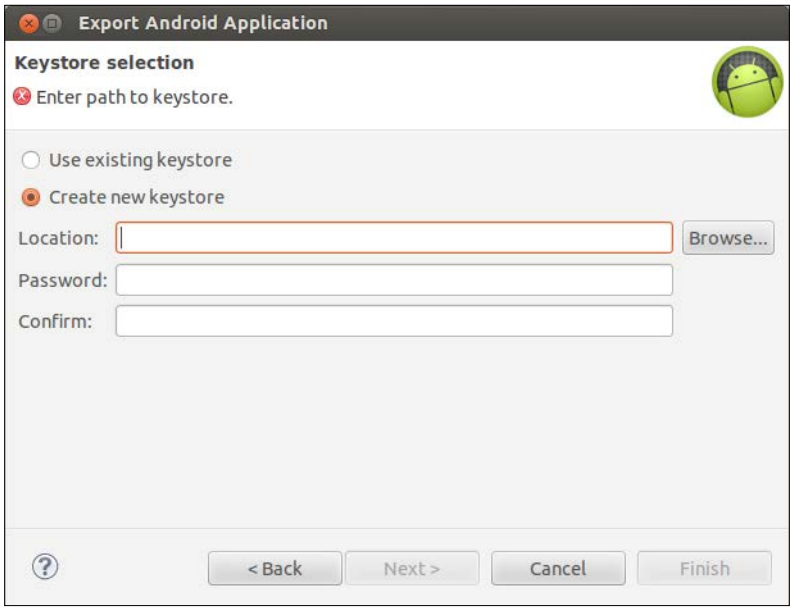
Using the Eclipse **Android Development Tool** (**ADT**), all the aforementioned steps in the *Preparing for release* section can be done with ease. Let's prepare our DistanceConverter from the earlier chapter for release using the Eclipse ADT.

Follow the given steps:

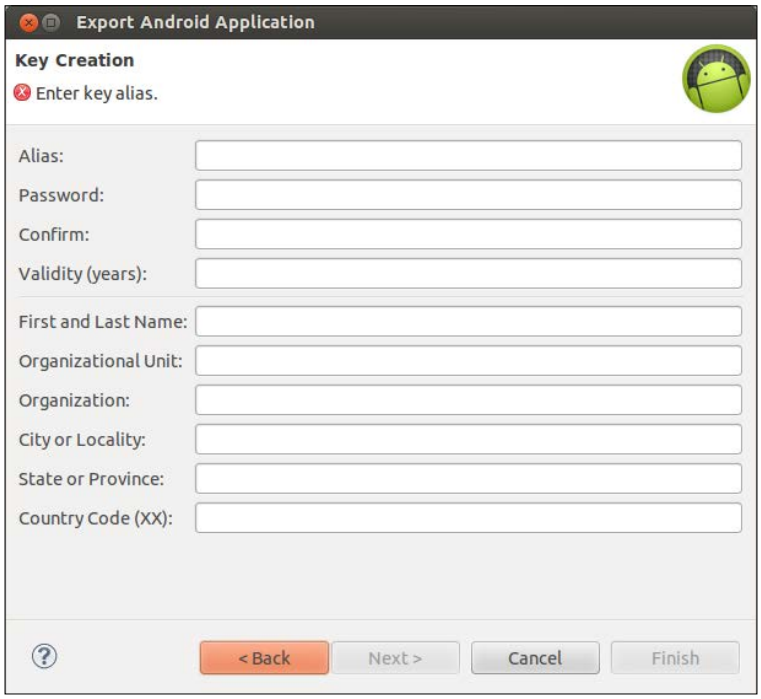
1. Right-click on the project **DistanceConverter** and then select **Export** from the context menu. Select **Export Android Application**, as shown in the following screenshot:



2. The **Export** wizard will now guide you through the process of signing, including the steps for selecting the private key (if already generated using the tool), or creating a new keystore and private key. Some of the following screens are captured, with the first screenshot being the creation of a keystore.
3. Now select **Create new keystore** and provide the **Location** and **Password** values:



4. In the following screen we can enter other details about the key creation as specified in the next table:



5. In the **Export Android Application** wizard, fill in the respective details:

Field	Value
Alias	`DIS` – It is the key alias name
Password	`<password>`
Validity	`25` – for publishing in Google Play, a period ending 22 October 2033 is a requirement
First and Last Name	`<NAME>`
Organizational Unit	Personal
Organization	Personal
City or Locality	`<CITY NAME>`
State or Province	`<STATE NAME>`
Country Code(xx)	Two letter code (for example, US)

6. Click on **Finish**, and the result is compiled, signed, aligned, and ready for distribution.

Publishing to Google Play

Publishing at Google Play is very simple and involves the following:

- **Register for Google Play**: Visit and register it at `https://play.google.com/`. It requires $25 USD to register, and is fairly straightforward and can take a few days until you get the final access.
- **Uploading APK**: Once the registration is over, the users have to log in and upload the APK file using the **Upload Application** link. Also, they have to upload the required assets, and edit the listing details, the one users will see when they browse the application in store.
- Finish up the task by using the **publish** button.

Getting help

For more information and help on signing and publishing, refer to following links:

- `http://developer.android.com/tools/publishing/app-signing.html`
- `http://developer.android.com/tools/publishing/versioning.html`
- `http://developer.android.com/tools/publishing/preparing.html`

Summary

In this chapter, we learned about the steps involved in signing and distribution of APK, and how it can be achieved using the Eclipse ADT easily.

Index

Symbols

A

C

D

E

F

G

H

I

J

L

M

N

X

Thank you for buying

Android Development Tools for Eclipse

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

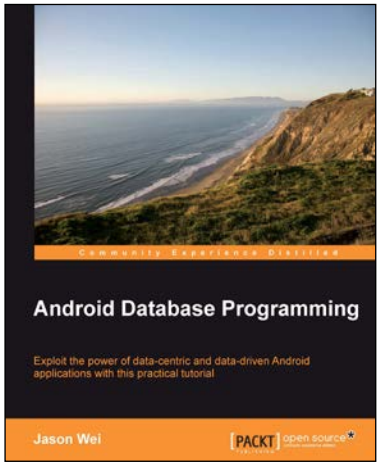
Android 3.0 Application Development Cookbook

ISBN: 978-1-84951-294-7 Paperback: 272 pages

Over 70 working recipes covering every aspect of Android development

1. Written for Android 3.0 but also applicable to lower versions
2. Quickly develop applications that take advantage of the very latest mobile technologies, including web apps, sensors, and touch screens
3. Part of Packt's Cookbook series: Discover tips and tricks for varied and imaginative uses of the latest Android features

---



Android Database Programming

ISBN: 978-1-84951-812-3 Paperback: 212 pages

Exploit the power of data-centric and data-driven Android applications with this practical tutorial

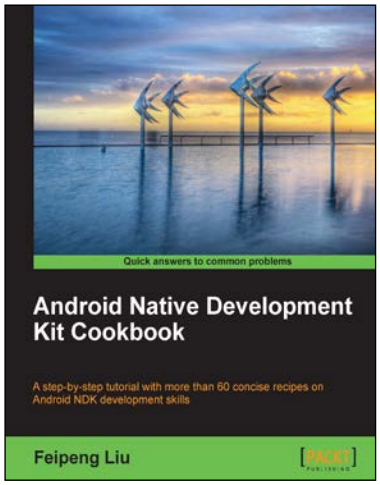
1. Master the skills to build data-centric Android applications
2. Go beyond just code by challenging yourself to think about practical use-cases with SQLite and others
3. Focus on flushing out high level design concepts, before drilling down into different code examples

Android 4: New Features for Application Development

ISBN: 978-1-84951-952-6 Paperback: 166 pages

Develop Android applications using the new features of Android Ice Cream Sandwich

1. Learn new APIs in Android 4
2. Get familiar with the best practices in developing Android applications
3. Step-by-step approach with clearly explained sample codes



Android Native Development Kit Cookbook

ISBN: 978-1-84969-150-5 Paperback: 346 pages

A step-by-step tutorial with more than 60 concise recipes on Android NDK development skills

1. Build, debug, and profile Android NDK apps
2. Implement part of Android apps in native C/C++ code
3. Optimize code performance in assembly with Android NDK