Android 4.2 App Development



Essentials

Android 4.2 App Development Essentials

Android 4.2 App Development Essentials – First Edition

ISBN-13: 978-0-9860273-4-5

© 2013 Neil Smyth. All Rights Reserved.

This book is provided for personal use only. Unauthorized use, reproduction and/or distribution strictly prohibited. All rights reserved.

The content of this book is provided for informational purposes only. Neither the publisher nor the author offers any warranties or representation, express or implied, with regard to the accuracy of information contained in this book, nor do they accept any liability for any loss or damage arising from any errors or omissions.

This book contains trademarked terms that are used solely for editorial purposes and to the benefit of the respective trademark owner. The terms used within this book are not intended as infringement of any trademarks.

Rev 1.0

Table of Contents

1. Introduction	1
1.1 Downloading the Code Samples	1
1.2 Feedback	1
1.3 Errata	1
2. Setting up an Android Development Environment	3
2.1 System Requirements	3
2.2 Installing the Java Development Kit (JDK)	3
2.2.1 Windows JDK Installation	3
2.2.2 Mac OS X JDK Installation	4
2.2.3 Linux JDK Installation	4
2.3 Downloading the Android Developer Tools (ADT) Bundle	5
2.4 Installing the ADT Bundle	5
2.4.1 Installation on Windows	5
2.4.2 Installation on Mac OS X	6
2.4.3 Installation on Linux	6
2.5 Installing the Latest Android SDK Packages	7
2.6 Making the Android SDK Tools Command-line Accessible	8
2.6.1 Windows 7	9
2.6.2 Windows 8	9
2.6.3 Linux	
2.6.4 Mac OS X	
2.7 Updating the ADT	
2.8 Adding the ADT Plugin to an Existing Eclipse Integration	10
2.9 Summary	12
3. Creating an Android Virtual Device (AVD)	
3.1 About Android Virtual Devices	13
3.2 Creating a New AVD	14
3.3 Starting the Emulator	16
3.4 AVD Command-line Creation	17
3.5 Android Virtual Device Configuration Files	
3.6 Moving and Renaming an Android Virtual Device	19
3.7 Summary	19
4. Creating an Example Android Application	
4.1 Creating a New Android Project	21
4.2 Defining the Project Name and SDK Settings	21
4.3 Project Configuration Settings	22
4.4 Configuring the Launcher Icon	23
4.5 Creating an Activity	24
4.6 Running the Application in the AVD	25
4.7 Stopping a Running Application	27

4.8 Modifying the Example Application	29
4.9 Reviewing the Layout and Resource Files	
4.10 Summary	
5. Testing Android Applications on a Physical Android Device with ADB	
5.1 An Overview of the Android Debug Bridge (ADB)	
5.2 Enabling ADB on Android 4.2 based Devices	
5.2.1 Mac OS X ADB Configuration	
5.2.2 Windows ADB Configuration	40
5.2.3 Linux adb Configuration	
5.3 Testing the adb Connection	44
5.4 Manual Selection of the Application Run Target	44
5.5 Summary	46
6. An Overview of the Android Architecture	
6.1 The Android Software Stack	47
6.2 The Linux Kernel	48
6.3 Android Runtime - Dalvik Virtual Machine	49
6.4 Android Runtime – Core Libraries	49
6.4.1 Dalvik VM Specific Libraries	49
6.4.2 Java Interoperability Libraries	
6.4.3 Android Libraries	50
6.4.4 C/C++ Libraries	50
6.5 Application Framework	51
6.6 Applications	51
6.7 Summary	51
7. The Anatomy of an Android Application	53
7.1 Android Activities	53
7.2 Android Intents	54
7.3 Broadcast Intents	54
7.4 Broadcast Receivers	54
7.5 Android Services	54
7.6 Content Providers	55
7.7 The Application Manifest	55
7.8 Application Resources	55
7.9 Application Context	55
7.10 Summary	55
8. Understanding Android Application and Activity Lifecycles	
8.1 Android Applications and Resource Management	57
8.2 Android Process States	57
8.2.1 Foreground Process	58
8.2.2 Visible Process	58
8.2.3 Service Process	58
8.2.4 Background Process	58

8.2.5 Empty Process	59
8.3 Inter-Process Dependencies	59
8.4 The Activity Lifecycle	59
8.5 The Activity Stack	59
8.6 Activity States	60
8.7 Configuration Changes	61
8.8 Handling State Change	61
8.9 Summary	61
9. Handling Android Activity State Changes	63
9.1 The Activity Class	63
9.2 Dynamic State vs. Persistent State	64
9.3 The Android Activity Lifecycle Methods	65
9.4 Activity Lifetimes	66
9.5 Summary	67
10. Android Activity State Changes by Example	69
10.1 Creating the State Change Example Project	69
10.2 Designing the User Interface	70
10.3 Overriding the Activity Lifecycle Methods	71
10.4 Enabling and Filtering the LogCat Panel	73
10.5 Running the Application	74
10.6 Experimenting with the Activity	75
10.7 Saving Dynamic State	76
10.8 Summary	76
11. Saving and Restoring the User Interface State of an Android Activity	77
11.1 Saving Dynamic State	77
11.2 The Bundle Class	77
11.3 Saving the State	78
11.4 Restoring the State	79
11.5 Testing the Application	80
11.6 Summary	80
12. Understanding Android Views, View Groups and Layouts	
12.1 Designing for Different Android Devices	81
12.2 Views and View Groups	81
12.3 Android Layout Managers	82
12.4 The View Hierarchy	83
12.5 Creating User Interfaces	84
12.6 Summary	85
13. Designing an Android User Interface using the Graphical Layout Tool	
13.1 The Android Graphical Layout Tool	87
13.2 A Graphical Layout Tool Example	87
13.3 Adding an XML Resource File to the Project	88
13.4 Editing View Properties	90

13.5 Using the View Properties Sheet	
13.6 Creating a New Activity	
13.7 Adding the New Activity to the Manifest File	
13.8 Running the Application	
13.9 Manually Creating an XML Layout	
13.10 Using the Hierarchy Viewer	
13.11 Summary	
14. Creating an Android User Interface in Java Code	
14.1 Java Code vs. XML Layout Files	
14.2 Creating Views	
14.3 Properties and Layout Parameters	
14.4 Creating the Example Project	
14.5 Adding Views to an Activity	
14.6 Setting View Properties	
14.7 Adding Layout Parameters and Rules	
14.8 Using View IDs	
14.9 Converting Density Independent Pixels (dp) to Pixels (px)	
14.10 Summary	
15. Using the Android GridLayout Manager in the Graphical Layout Tool	
15.1 Introducing the Android GridLayout and Space Classes	
15.2 The GridLayout Example	
15.3 Creating the GridLayout Project	
15.4 Creating the GridLayout Instance	
15.5 An Overview of the GridLayout in the Graphical Layout Tool	
15.6 Adding Views to GridLayout Cells	
15.7 Implementing Cell Row and Column Spanning	
15.8 Changing the Gravity of a GridLayout Child	
15.9 Summary	
16. Working with the Android GridLayout in XML Layout Resources	
16.1 GridLayouts in XML Resource Files	
16.2 Adding Child Views to the GridLayout	
16.3 Declaring Cell Spanning, Gravity and Margins	
16.4 Summary	
17. An Overview and Example of Android Event Handling	
17.1 Understanding Android Events	
17.2 Using the android:onClick Resource	
17.3 Event Listeners and Callback Methods	
17.4 An Event Handling Example	
17.5 Designing the User Interface	
17.6 The Event Listener and Callback Method	131
17.7 Consuming Events	
17.8 Summary	

18. Android Touch and Multi-touch Event Handling	
18.1 Intercepting Touch Events	137
18.2 The MotionEvent Object	138
18.3 Understanding Touch Actions	138
18.4 Handling Multiple Touches	138
18.5 An Example Multi-Touch Application	139
18.6 Designing the Activity User Interface	139
18.7 Implementing the Touch Event Listener	140
18.8 Running the Example Application	143
18.9 Summary	144
19. Detecting Common Gestures using the Android Gesture Detector Class	
19.1 Implementing Common Gesture Detection	145
19.2 Creating an Example Gesture Detection Project	146
19.3 Implementing the Listener Class	146
19.4 Creating the GestureDetectorCompat Instance	149
19.5 Implementing the onTouchEvent() Method	150
19.6 Testing the Application	150
19.7 Summary	151
20. Implementing Custom Gesture and Pinch Recognition on Android	
20.1 The Android Gesture Builder Application	
20.2 The GestureOverlayView Class	
20.3 Detecting Gestures	153
20.4 Identifying Specific Gestures	153
20.5 Adding SD Card Support to an AVD	154
20.6 Building and Running the Gesture Builder Application	155
20.7 Creating a Gestures File	155
20.8 Extracting the Gestures File from the SD Card	157
20.9 Creating the Example Project	157
20.10 Designing the User Interface	158
20.11 Loading the Gestures File	159
20.12 Registering the Event Listener	160
20.13 Implementing the onGesturePerformed Method	160
20.14 Testing the Application	162
20.15 Configuring the GestureOverlayView	162
20.16 Intercepting Gestures	162
20.17 Detecting Pinch Gestures	163
20.18 A Pinch Gesture Example Project	163
20.19 Summary	165
21. An Introduction to Android Fragments	167
21.1 What is a Fragment?	167
21.2 Creating a Fragment	167
21.3 Adding a Fragment to an Activity using the Layout XML File	169
21.4 Adding and Managing Fragments in Code	

21.5 Handling Fragment Events	
21.6 Implementing Fragment Communication	
21.7 Summary	
22. Using Fragments in Android - A Worked Example	
22.1 About the Example Fragment Application	
22.2 Creating the Example Project	
22.3 Adding the Android Support Library	
22.4 Creating the First Fragment Layout	179
22.5 Creating the First Fragment Class	
22.6 Creating the Second Fragment Layout	
22.7 Adding the Fragments to the Activity	
22.8 Making the Toolbar Fragment Talk to the Activity	
22.9 Making the Activity Talk to the Text Fragment	
22.10 Testing the Application	
22.11 Summary	
23. An Android Master/Detail Flow Tutorial	
23.1 The Master/Detail Flow	
23.2 Creating a Master/Detail Flow Activity	
23.3 The Anatomy of the Master/Detail Flow Template	
23.4 Modifying the Master/Detail Flow Template	
23.5 Changing the Content Model	
23.6 Changing the Detail Pane	
23.7 Modifying the WebsiteDetailFragment Class	
23.8 Adding Manifest Permissions	
23.9 Running the Application	
23.10 Summary	
24. Creating and Managing Overflow Menus on Android	
24.1 The Overflow Menu	
24.2 Creating a Overflow Menu	
24.3 Displaying an Overflow Menu	
24.4 Responding to Menu Item Selections	
24.5 Creating Checkable Item Groups	
24.6 Creating the Example Project	
24.7 Modifying the Menu Description	
24.8 Implementing the onOptionsItemSelected() Method	209
24.9 Testing the Application	
24.10 Summary	210
25. An Overview of Android Intents	
25.1 An Overview of Intents	
25.2 Explicit Intents	211
25.3 Returning Data from an Activity	213
25.4 Implicit Intents	214

25.5 Using Intent Filters	
25.6 Checking Intent Availability	
25.7 Summary	215
26. Android Explicit Intents – A Worked Example	
26.1 Creating the Explicit Intent Example Application	217
26.2 Designing the User Interface Layout for ActivityA	
26.3 Creating the Second Activity Class	
26.4 Creating the User Interface for ActivityB	
26.5 Adding ActivityB to the Application Manifest File	
26.6 Creating the Intent	
26.7 Extracting Intent Data	
26.8 Launching ActivityB as a Sub-Activity	
26.9 Returning Data from a Sub-Activity	
26.10 Testing the Application	
26.11 Summary	
27. Android Implicit Intents – A Worked Example	
27.1 Creating the Implicit Intent Example Project	
27.2 Designing the User Interface	
27.3 Creating the Implicit Intent	
27.4 Adding a Second Matching Activity	
27.5 Adding the Web View to the UI	230
27.6 Obtaining the Intent URL	231
27.7 Modifying the MyWebView Project Manifest File	232
27.8 Installing the MyWebView Package on a Device	233
27.9 Testing the Application	234
27.10 Summary	234
28. Android Broadcast Intents and Broadcast Receivers	
28.1 An Overview of Broadcast Intents	235
28.2 An Overview of Broadcast Receivers	
28.3 Obtaining Results from a Broadcast	
28.4 Sticky Broadcast Intents	
28.5 The Broadcast Intent Example	
28.6 Creating the Example Application	
28.7 Creating and Sending the Broadcast Intent	
28.8 Creating the Broadcast Receiver	
28.9 Configuring a Broadcast Receiver in the Manifest File	
28.10 Testing the Broadcast Example	
28.11 Listening for System Broadcasts	
28.12 Summary	
29. A Basic Overview of Android Threads and Thread Handlers	
29.1 An Overview of Threads	245
29.2 The Application Main Thread	245

29.3 Thread Handlers	
29.4 A Basic Threading Example	
29.5 Creating a New Thread	
29.6 Implementing a Thread Handler	
29.7 Passing a Message to the Handler	
29.8 Summary	
30. An Overview of Android Started and Bound Services	253
30.1 Started Services	
30.2 Intent Service	
30.3 Bound Service	
30.4 The Anatomy of a Service	
30.5 Controlling Destroyed Service Restart Options	
30.6 Declaring a Service in the Manifest File	
30.7 Starting a Service Running on System Startup	
30.8 Summary	
31. Implementing an Android Started Service – A Worked Example	259
31.1 Creating the Example Project	259
31.2 Creating the Service Class	
31.3 Adding the Service to the Manifest File	
31.4 Starting the Service	
31.5 Testing the IntentService Example	
31.6 Using the Service Class	
31.7 Creating the New Service	
31.8 Modifying the User Interface	
31.9 Running the Application	
31.10 Creating a New Thread for Service Tasks	
31.11 Summary	
32. Android Local Bound Services – A Worked Example	
32.1 Understanding Bound Services	
32.2 Bound Service Interaction Options	
32.3 A Local Bound Service Example	
32.4 Adding a Bound Service to the Project	
32.5 Implementing the Binder	
32.6 Binding the Client to the Service	
32.7 Completing the Example	
32.8 Testing the Application	
32.9 Summary	
33. Android Remote Bound Services – A Worked Example	279
33.1 Client to Remote Service Communication	
33.2 Creating the Example Application	279
33.3 Designing the User Interface	
33.4 Implementing the Remote Bound Service	

33.5 Configuring a Remote Service in the Manifest File	
33.6 Launching and Binding to the Remote Service	
33.7 Sending a Message to the Remote Service	
33.8 Summary	
34. An Overview of Android SQLite Databases	
34.1 Understanding Database Tables	
34.2 Introducing Database Schema	
34.3 Columns and Data Types	
34.4 Database Rows	
34.5 Introducing Primary Keys	
34.6 What is SQLite?	
34.7 Structured Query Language (SQL)	
34.8 Trying SQLite on an Android Virtual Device (AVD)	
34.9 Android SQLite Java Classes	
34.9.1 Cursor	
34.9.2 SQLiteDatabase	
34.9.3 SQLiteOpenHelper	
34.9.4 ContentValues	
34.10 Summary	291
35. An Android TableLayout and TableRow Tutorial	
35.1 The TableLayout and TableRow Layout Views	293
35.2 Creating the Database Project	
35.3 Designing the User Interface Layout	
35.4 Summary	
36. An Android SQLite Database Tutorial	
36.1 About the Database Example	
36.2 Creating the Data Model	
36.3 Implementing the Data Handler	
36.3.1 The Add Handler Method	
36.3.2 The Query Handler Method	
36.3.3 The Delete Handler Method	
36.4 Implementing the Activity Event Methods	
36.5 Testing the Application	
36.6 Summary	
37. Understanding Android Content Providers	
37.1 What is a Content Provider?	
37.2 The Content Provider	
37.2.1 onCreate()	
37.2.2 query()	
37.2.3 insert()	
37.2.4 update()	
37.2.5 delete()	

37.2.6 getType()	314
37.3 The Content URI	
37.4 The Content Resolver	
37.5 The <provider> Manifest Element</provider>	
37.6 Summary	315
38. Implementing an Android Content Provider	
38.1 Copying the Database Project	
38.2 Adding the Content Provider Package	
38.3 Creating the Content Provider Class	318
38.4 Constructing the Authority and Content URI	319
38.5 Implementing URI Matching in the Content Provider	
38.6 Implementing the Content Provider onCreate() Method	
38.7 Implementing the Content Provider insert() Method	
38.8 Implementing the Content Provider query() Method	
38.9 Implementing the Content Provider update() Method	325
38.10 Implementing the Content Provider delete() Method	326
38.11 Declaring the Content Provider in the Manifest File	
38.12 Modifying the Database Handler	328
38.13 Summary	
39. Implementing Video Playback on Android using the VideoView and MediaController Classes	331
39.1 Introducing the Android VideoView Class	
39.2 Introducing the Android MediaController Class	332
39.3 Testing Video Playback	
39.4 Creating the Video Playback Example	332
39.5 Designing the VideoPlayer Layout	
39.6 Configuring the VideoView	
39.7 Adding Internet Permission	334
39.8 Adding the MediaController to the Video View	335
39.9 Setting up the onPreparedListener	
39.10 Summary	337
40. Video Recording and Image Capture on Android using Camera Intents	339
40.1 Checking for Camera Support	
40.2 Calling the Video Capture Intent	340
40.3 Calling the Image Capture Intent	
40.4 Creating an Example Video Recording Project	341
40.4 Creating an Example Video Recording Project 40.5 Designing the User Interface Layout	341 342
40.4 Creating an Example Video Recording Project 40.5 Designing the User Interface Layout 40.6 Checking for the Camera	
40.4 Creating an Example Video Recording Project 40.5 Designing the User Interface Layout 40.6 Checking for the Camera 40.7 Launching the Video Capture Intent	
 40.4 Creating an Example Video Recording Project	341 342 343 344 345
 40.4 Creating an Example Video Recording Project	
 40.4 Creating an Example Video Recording Project	

41.1 Playing Audio	347
41.2 Recording Audio and Video using the MediaRecorder Class	
41.3 About the Example Project	349
41.4 Creating the AudioApp Project	349
41.5 Designing the User Interface	
41.6 Checking for Microphone Availability	350
41.7 Performing the Activity Initialization	
41.8 Implementing the recordAudio() Method	353
41.9 Implementing the stopClicked() Method	353
41.10 Implementing the playAudio() method	354
41.11 Configuring Permissions in the Manifest File	354
41.12 Testing the Application	355
41.13 Summary	355
42. Working with the Google Maps Android API	
42.1 The Elements of the Google Maps Android API	357
42.2 Getting Ready to use the Google Maps Android API	358
42.2.1 Installing the Google APIs	358
42.2.2 Downloading the Google Play Services SDK	358
42.2.3 Adding the Google Play Services Library Project to the Eclipse Workspace	359
42.2.4 Adding the Google Play Services Library to a Project Build Path	359
42.2.5 Obtaining Your Developer Signature	361
42.2.6 Registering the Project in the Google APIs Console	
42.3 Adding Map Support to the AndroidManifest.xml File	364
42.4 Checking for Google Play Services Support	
42.5 Understanding Geocoding and Reverse Geocoding	
42.6 Adding a Map to an Application	
42.7 Displaying the User's Current Location	369
42.8 Changing the Map Type	369
42.9 Displaying Map Controls to the User	370
42.10 Handling Map Gesture Interaction	370
42.10.1 Map Zooming Gestures	
42.10.2 Map Scrolling/Panning Gestures	
42.10.3 Map Tilt Gestures	
42.10.4 Map Rotation Gestures	
42.11 Creating Map Markers	
42.12 Controlling the Map Camera	372
42.13 Summary	
43. Handling Different Android Devices and Displays	
43.1 Handling Different Device Displays	375
43.2 Creating a Layout for each Display Size	375
43.3 Providing Different Images	
43.4 Checking for Hardware Support	377
43.5 Providing Device Specific Application Binaries	377
43.6 Summary	378

44. Signing and Preparing an Android Application for Release	379
44.1 The Release Preparation Process	379
44.2 Accessing the Export Wizard	379
44.3 Creating a Keystore File	
44.4 Generating a Private Key	
44.5 Creating the Application APK File	
44.6 Register for a Google Play Developer Console Account	
44.7 Summary	
45. Integrating Google Play In-app Billing into an Android Application – A Tutorial	385
45.1 Installing the Google Play Billing Library	
45.2 Creating the Example In-app Billing Project	
45.3 Adding Billing Permission to the Manifest File	
45.4 Adding the IInAppBillingService.aidl File to the Project	387
45.5 Adding the Utility Classes to the Project	
45.6 Designing the User Interface	
45.7 Implementing the "Click Me" Button	391
45.8 Google Play Developer Console and Google Wallet Accounts	392
45.9 Obtaining the Public License Key for the Application	392
45.10 Setting Up Google Play Billing in the Application	393
45.11 Initiating a Google Play In-app Billing Purchase	395
45.12 Implementing the onActivityResult Method	
45.13 Implementing the Purchase Finished Listener	396
45.14 Consuming the Purchased Item	397
45.15 Releasing the labHelper Instance	398
45.16 Testing the In-app Billing Application	398
45.17 Creating a New In-app Product	399
45.18 Adding In-app Billing Test Accounts	401
45.19 Summary	402
Index	403

1. Introduction

The goal of this book is to teach the skills necessary to develop Android based applications using the Eclipse Integrated Development Environment (IDE) and the Android 4.2 Software Development Kit (SDK).

Beginning with the basics, this book provides an outline of the steps necessary to set up an Android development and testing environment. An introduction to the architecture of Android is followed by an indepth look at the design of Android applications and user interfaces. More advanced topics such as database management, content providers and intents are also covered, as are touch screen handling, gesture recognition, camera access and the playback and recording of both video and audio.

In addition to covering general Android development techniques, the book also includes Google Play specific topics such as using the Google Play In-App Billing API, implementing maps using the Google Maps Android API and submitting apps to the Google Play Developer Console.

Assuming you already have some Java programming experience, are ready to download Eclipse and the Android SDK, have access to a Windows, Mac or Linux system and ideas for some apps to develop, you are ready to get started.

1.1 Downloading the Code Samples

The source code and Eclipse project files for the examples contained in this book are available for download at:

http://www.ebookfrenzy.com/direct/android42/index.php

Once the file has been downloaded and unzipped, the samples may be imported into an existing Eclipse workspace by selecting the Eclipse *File -> Import...* menu option and choosing the *Android -> Existing Android Code Into Workspace* category. When prompted, select the folder containing the sample project folders as the *Root Directory* before choosing the sample projects to be imported from the resulting list.

1.2 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book, or have any comments, questions or concerns please contact us at *feedback@ebookfrenzy.com*.

1.3 **Errata**

Whilst we make every effort to ensure the accuracy of the content of this book, it is inevitable that a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined, together with solutions, at the following URL:

http://www.ebookfrenzy.com/errata/android42.html

In the event that you find an error not listed in the errata, please let us know by emailing our technical support team at *feedback@ebookfrenzy.com*. They are there to help you and will work to resolve any problems you may encounter.

2. Setting up an Android Development Environment

Before any work can begin on the development of an Android application, the first step is to configure a computer system to act as the development platform. This involves a number of steps consisting of installing the Java Development Kit (JDK), the Eclipse Integrated Development Environment (IDE) and the appropriate Android Software Development Kit (SDK). In addition to these steps, it will also be necessary to install the Eclipse Android Development Tool (ADT) Plug-in.

This chapter will cover the steps necessary to install the requisite components for Android application development on Windows, Mac OS X and Linux based systems.

2.1 System Requirements

Android application development may be performed on any of the following system types:

- Windows XP (32-bit)
- Windows Vista (32-bit or 64-bit)
- Windows 7 (32-bit or 64-bit)
- Windows 8
- Mac OS X 10.5.8 or later (Intel based systems only)
- Linux systems with version 2.7 or later of GNU C Library (glibc)

2.2 Installing the Java Development Kit (JDK)

Both the Eclipse IDE and Android SDK were developed using the Java programming language. Similarly, Android applications are also developed using Java. As a result, the Java Development Kit (JDK) is the first component that must be installed.

Android development requires the installation of the Standard Edition of the Java Platform Development Kit version 6 or later. Java is provided in both development (JDK) and runtime (JRE) packages. For the purposes of Android development, the JDK must be installed.

2.2.1 Windows JDK Installation

For Windows systems, the JDK may be obtained from Oracle Corporation's website using the following URL:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Assuming that a suitable JDK is not already installed on your system, download the latest JDK package that matches the destination computer system. Once downloaded, launch the installation executable and follow the on screen instructions to complete the installation process.

2.2.2 Mac OS X JDK Installation

The Java SE 6 environment or a more recent version should already be installed on the latest Mac OS X versions. To confirm the version that is installed, open a Terminal window and enter the following command:

java -version

Assuming that Java is currently installed, output similar to the following will appear in the terminal window:

```
java version "1.6.0_37"
Java(TM) SE Runtime Environment (build 1.6.0_37-b06-434-11M3909)
Java HotSpot(TM) 64-Bit Server VM (build 20.12-b01-434, mixed mode)
```

In the event that Java is not installed, issuing the "java" command in the terminal window should initiate the JDK installation process.

2.2.3 Linux JDK Installation

Firstly, if the chosen development system is running the 64-bit version of Ubuntu then it is essential that the 32-bit library support package be installed:

sudo apt-get install ia32-libs

As with Windows based JDK installation, it is possible to install the JDK on Linux by downloading the appropriate package from the Oracle web site, the URL for which is as follows:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Packages are provided by Oracle in RPM format (for installation on Red Hat Linux based systems such as Red Hat Enterprise Linux, Fedora and CentOS) and as a tar archive for other Linux distributions such as Ubuntu.

On Red Hat based Linux systems, download the .rpm JDK file from the Oracle web site and perform the installation using the *rpm* command in a terminal window. Assuming, for example, that the downloaded JDK file was named *jdk-7u10-linux-x64.rpm*, the commands to perform the installation would read as follows:

su

rpm -ihv jdk-7u10-linux-x64.rpm

To install using the compressed tar package (tar.gz) perform the following steps:

1. Create the directory into which the JDK is to be installed (for the purposes of this example we will assume */home/demo/java*).

2. Download the appropriate tar.gz package from the Oracle web site into the directory.

3. Execute the following command (where *<jdk-file>* is replaced by the name of the downloaded JDK file):

tar xvfz <jdk-file>.tar.gz

4. Remove the downloaded tar.gz file.

5. Add the path to the *bin* directory of the JDK installation to your \$PATH variable. For example, assuming that the JDK ultimately installed into */home/demo/java/jdk1.7.0_10* the following would need to be added to your \$PATH environment variable:

/home/demo/java/jdk1.7.0_10/bin

This can typically be achieved by adding a command to the *.bashrc* file in your home directory (specifics may differ depending on the particular Linux distribution in use). For example, change directory to your home directory, edit the *.bashrc* file contained therein and add the following line at the end of the file (modifying the path to match the location of the JDK on your system):

export PATH=/home/demo/java/jdk1.7.0 10/bin:\$PATH

Having saved the change, future terminal sessions will include the JDK in the \$PATH environment variable.

2.3 Downloading the Android Developer Tools (ADT) Bundle

Most of the work involved in developing applications for Android will be performed using the Eclipse Integrated Development Environment (IDE). If you are already using Eclipse to develop for other platforms, then the Android Developer Tools (ADT) plug-in can be integrated into your existing Eclipse installation (a topic covered later in this chapter). If, on the other hand, you are entirely new to Eclipse based development, the most convenient path to take is to install a package known as the *ADT Bundle*. This bundle includes many of the tools necessary to begin developing Android applications in a single download.

The ADT Bundle may be downloaded from the following web page:

https://developer.android.com/sdk/index.html

From this page, either click on the download button if it lists the correct platform (for example on a Windows based web browser the button will read "Download the SDK ADT Bundle for Windows"), or select the "Download for Other Platforms" option to manually select the appropriate package for your platform and operating system. On the subsequent screen, accept the terms and conditions, the target architecture of your computer system (32-bit or 64-bit) and click on the download button. Note that your choice of 32-bit or 64-bit should match the architecture chosen for the JDK installation. Attempting to run a 64-bit ADT bundle using a 32-bit JDK, for example, will result in errors when attempting to launch Eclipse.

2.4 Installing the ADT Bundle

The ADT Bundle is downloaded as a compressed ZIP archive file which must be unpacked to complete the installation process. The exact steps to achieve this differ depending on the operating system.

2.4.1 Installation on Windows

Locate the downloaded ADT Bundle zip file in a Windows Explorer window, right-click on it and select the *Extract All...* menu option. In the resulting dialog, choose a suitable location into which to unzip the file

before clicking on the *Extract* button. When choosing a suitable location, keep in mind that the extraction will create a sub-folder in the chosen location named either *adt-bundle-windows-x86* or *adt-bundle-windows-x86_64* containing the bundle packages.

Once the extraction is complete, navigate in Windows Explorer to the directory containing the ADT bundle, move into the *eclipse* sub-folder and double click on the *eclipse* executable to start the Eclipse IDE environment. For easier future access, right click on the eclipse executable and select *Pin to Taskbar* from the resulting menu.

It is possible that Windows will display a Security Warning dialog before Eclipse will launch stating that the publisher could not be verified. In the event that this warning appears, uncheck the "Always ask before opening this file" option before clicking the *Run* button. Once invoked, Eclipse will prompt for the location of the workspace. All projects will be stored by default into this folder. Browse for a suitable location, or choose the default offered by Eclipse and click on *OK*.

2.4.2 Installation on Mac OS X

On Mac OS X systems, open a terminal window, change directory to the location where Eclipse is to be installed and execute the following command:

unzip /<path to package>/<package name>.zip

For example, assuming a package file named *adt-bundle-mac-x86_64.zip* has been downloaded to */home/demo/Downloads*, the following command would be needed to install Eclipse:

unzip /home/demo/Downloads/adt-bundle-mac-x86 64.zip

Note that the bundle will be installed into a sub-directory named *adt-bundle-mac-x86_64*. Assuming, therefore, that the above command was executed in */Users/demo*, the software packages will be unpacked into */Users/demo/adt-bundle-mac-x86_64*. Within this directory, the files comprising the Eclipse IDE are installed in a sub-directory named *eclipse*.

Using the Finder tool, navigate to the *eclipse* sub-directory of the ADT bundle installation directory and double click on the *eclipse* executable to launch the application. For future easier access to the tool, simply drag the eclipse icon from the Finder window and drop it onto the dock.

2.4.3 Installation on Linux

On Linux systems, open a terminal window, change directory to the location where Eclipse is to be installed and execute the following command:

unzip /<path to package>/<package name>.zip

For example, assuming a package file named *adt-bundle-linux-x86.zip* has been downloaded to */home/demo/Downloads*, the following command would be needed to install Eclipse:

unzip /home/demo/Downloads/adt-bundle-linux-x86.zip

Note that the bundle will be installed into a sub-directory named either *adt-bundle-linux-x86* or adt-*bundle-linux-x86_64* depending on whether the 32-bit or 64-bit edition was downloaded. Assuming, therefore, that the above command was executed in */home/demo*, the software packages will be unpacked into */home/demo/adt-bundle-linux-x86*. Within this directory, the files comprising the Eclipse IDE are installed in a sub-directory named *eclipse*.

To launch Eclipse, open a terminal window, change directory to the *eclipse* sub-directory of the ADT bundle installation directory and execute the following command:

./eclipse

Once invoked, Eclipse will prompt for the location of the workspace. All projects will be stored by default into this folder. Browse for a suitable location, or choose the default offered by Eclipse and click on *OK*.

Having verified that the Eclipse IDE is installed correctly, keep Eclipse running so that it can be used to install additional Android SDK packages.

2.5 Installing the Latest Android SDK Packages

The steps performed so far have installed Java, the Eclipse IDE and the current set of default Android SDK packages. Before proceeding, it is worth taking some time to verify which packages are installed and to install any missing packages.

This task can be performed using the *Android SDK Manager*, which may be launched from within the Eclipse tool by selecting the *Window -> Android SDK Manager* menu option. Once invoked, the SDK Manager tool will appear as illustrated in Figure 2-1:

SDK Path: C\Users\nas\adt-bundle-windows-x86-20130219\sdk Packages Image: Solution of the state of	Packages Tools					
Packages Image API Rev. Status Image: Tools Image: Tools Image: Tools Image: Tools Image: Android SDK Polatform-tools 17 Image: Image: Tools Image: Tools Image: Android SDK Polatform-tools 17 Image: Image: Tools Image: Tools Image: Android SDK Build-tools 17 Image: Image: Tools Image: Tools Image: Tools 17 2 Not installed Image: Tools 17 2 Image: Tools Image: Tools 17 1 Not installed Image: Tools 17 1 Not installed Image: Tools 17 3 Image: Image: Tools Image: Tools 17 3 Image: Image: Tools Image: Tools 17 1 Not installed Image: Tools 17 1 Not installed Image: Tools Android All 2(API14) A	DK Path: C:\Users\nas\adt-bundle-windows-x86-20130219	\sdk				
Image API Rev. Status Image: Tools Z20.1 Image: Tools Z20.1 Image: Android SDK Pols Z20.1 Image: Tools Z20.1 Image: Android SDK Pols Image: Tools Image: Tools Image: Tools Image: Android SDK Pols Image: Tools Image: Tools Image: Tools Image: Tools Image: Android SDK Build-tools Image: Tools Image: Tools Image: Tools Image: Tools Image: Tools Image: Tools Imag	Packages					
Tools Android SDK Tools Android SDK Blafform-tools Android SDK Blafform-tools Android SDK Blafform-tools I7 Android SDK Blafform-tools I7 Android SDK Blafform-tools I7 Android SDK Blafform-tools I7 Android 4.22 (API17) Android 5DK I7 Android 4.22 (API17) Android 4.22 (API17) Android 4.22 (API17) Android 4.23 System Image I7 I Installed Android 4.24 (API17) Android 4.25 (API17) Android 4.26 (API14) Android 3.2 (API13) Android 3.2 (API14) Android 3.2 (API13) Android 3.2 (API14)	ițți Name	API	Rev.	Status		
Android SDK Platform-tools Android SDK Platform-tools Android SDK Build-tools Android SDK Build-tools Android ADK Platform Documentation for Android SDK I7 Android ADX (API11) Somples for SDK I7 I Installed Installed Somples for SDK I7 I Installed In	a 🔲 🧰 Tools					
	🖂 🥓 Android SDK Tools		22.0.1	👼 Installed		
Android SDK Build-tools Android 4.22 (API17) Android 5DK Build-tools Android 4.22 (API17) Documentation for Android SDK 17 C Not installed Android 5DK I7 C Installed Android 4.22 (API17) Android 4.22 (API17) Android 4.22 (API17) Android 4.23 (API17) Android 4.23 (API17) Android 4.23 (API12) Android 3.2 (API13) Android 3.1 (API12) Android 3.0 (API11) Android 3.0 (API13) Android 3.0 (API14) Android 3	🔲 🥓 Android SDK Platform-tools		17	👼 Installed		
Android 4.2 (API17) Android 5DK 17 2 Not installed SDK Platform 17 2 Constant of the second	🔲		17	👼 Installed		
Image: Construction for Android SDK 17 2 Not installed Image: Construction for Android SDK 17 2 Installed Image: Construction for Android SDK 17 1 Installed Image: Construction for Android SDK 17 1 Installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android SDK 17 1 Not installed Image: Construction for Android AD (API16) Image: Construction for Android AD (API13) Image: Construction for Android AD (API14) Image: Construction for Android AD (API13) Image: Construction for Android AD (API11) Image: Construction for Android AD (API11) Image: Construction for Android AD (API12) Image: Construction for Android AD (API11) Image: Construction for Android AD (API12)	Android 4.2.2 (API 17)					
Image: Solx Platform 17 2 Image: Solx Platform Image: Solx Platform 17 1 Image: Solx Platform Image: Solx Platform 17 3 Image: Solx Platform Image: Solx Platform 17 1 Not installed Image: Solx Platfore	Documentation for Android SDK	17	2	Not installed		
Image: Samples for SDK 17 1 Image: Ima	🔲 🖷 SDK Platform	17	2	😿 Installed		
Image 17 2 Installed Image 17 1 Not installed Image 17 1 Not installed Image 17 1 Not installed Image 17 3 Installed Image 17 1 Not installed Image Image 17 1	🔲 📥 Samples for SDK	17	1	👼 Installed		
Image 17 1 Not installed Image 17 1 Not installed Image 17 1 Not installed Image 17 3 Image Image Image 17 1 Not installed Image Image Image 17 1 Not installed Image Image Image Image Image	📄 💵 ARM EABI v7a System Image	17	2	큕 Installed		
Image 17 1 Not installed Image 17 3 Installed Image 17 3 Installed Image 17 1 Not installed Image Android 4.1.2 (AP116) Image Image: Android 4.0 (AP113) Image Image: Android 3.2 (AP113) Image: Android 3.0 (AP111) Image: Android 3.0 (AP111) Image: Im	🔲 🌆 Intel x86 Atom System Image	17	1	Not installed		
Image: Coogle APIs 17 3 Image: Image: Image: Coople APIs Image: Coople APIs 17 1 Not installed Image: Coople APIs 17 1 10	🔲 🌆 MIPS System Image	17	1	Not installed		
Image: Sources for Android SDK 17 1 Not installed Image: Sources for Android 4.12 (AP116) Image: Sources for Android 4.12 (AP116) Image: Sources for Android 4.12 (AP116) Image: Sources for Android 4.0 (AP114) Image: Sources for Android 3.1 (AP112) Image: Sources for Android 3.1 (AP112) Image: Sources for Android 3.0 (AP111) Image: Sources for Android 3.0 (AP111) Image: Sources for Android 3.0 (AP111) Image: Sources for Android 3.0 (AP111) Image: Sources for Android 3.0 (AP111) Image: Sources for Android 3.0 (AP111)	🔲 🫱 Google APIs	17	3	👼 Installed		
> C	Sources for Android SDK	17	1	Not installed		
> C_2 Android 4.0.3 (API 15) > C_2 Android 3.0 (API 14) > C_2 Android 3.2 (API 13) > C_2 Android 3.0 (API 12) > C_2 Android 3.0 (API 12) > C_2 Android 3.0 (API 11)	Image: Provide the second s					
> C_Android 4.0 (API14) > C_Android 3.2 (API13) > C_Android 3.1 (API12) > C_Android 3.0 (API11)	Image: Provide Android 4.0.3 (API 15)					
D Cg. Android 32 (API13) D Cg. Android 31 (API12) D Cg. Android 3.0 (API11)	Image: Marce Android 4.0 (API 14)					
▷ □ </td <td>Android 3.2 (API 13)</td> <td></td> <td></td> <td></td> <td></td> <td></td>	Android 3.2 (API 13)					
p Cat Android 3.0 (API 11) Show: ✓ Updates/New ✓ Installed Obsolete Select New or Updates	Android 3.1 (API 12)					
Show: ♥ Updates/New ♥ Installed Obsolete Select <u>New</u> or <u>Updates</u> Install 3 packages	Image: Provide the second state of the seco					
	Show: 🔽 Updates/New 💟 Installed 🛛 Obsolete Sel	ect <u>New</u> or <u>Up</u>	<u>dates</u>		Install 3 packag	ges
Sort by: API level Repository Deselect All Delete 3 packages	Sort by: API level Repository	select All			Delete 3 packa	ges

Figure 2-1

Once the SDK Manager is running, return to the main Eclipse window and select the *File -> Exit* menu option to exit from the Eclipse environment. This will leave the Android SDK Manager running whilst ensuring that the Eclipse session does not conflict with the installation process.

Begin by checking that the *SDK Path:* setting at the top of the SDK Manager window matches the location into which the ADT Bundle package was unzipped. If it does not, relaunch Eclipse and select the *Window -> Preferences* option. In the *Preferences* dialog, select the *Android* option from the left hand panel and change the *SDK Location* setting so that it references the *sdk* sub-folder of the directory into which the ADT Bundle was unzipped before clicking on *Apply* followed by *OK*.

Within the Android SDK Manager, make sure that the check boxes next to the following packages are selected:

- Tools > Android SDK Tools
- Tools > Android SDK Platform-tools
- SDK Platform Android 4.2.2 API 17 > SDK Platform
- SDK Platform Android 4.2.2 API 17 > ARM EABI v7a System Image
- Extras > Android Support Library

With the appropriate package selections made (and assuming these packages were not already installed), click on the *Install packages* button to initiate the installation process. In the resulting dialog, accept the license agreements before clicking on the *Install* button. The SDK Manager will then begin to download and install the designated packages. As the installation proceeds, a progress bar will appear at the bottom of the manager window indicating the status of the installation.

Once the installation is complete, review the package list and make sure that the selected packages are now listed as *Installed* in the *Status* column. If any are listed as *Not installed*, make sure they are selected and click on the *Install packages...* button again.

2.6 Making the Android SDK Tools Command-line Accessible

Most of the time, the underlying tools of the Android SDK will be accessed from within the Eclipse environment. That being said, however, there will also be instances where it will be useful to be able to invoke those tools from a command prompt or terminal window. In order for the operating system on which you are developing to be able to find these tools, it will be necessary to add them to the system's *PATH* environment variable.

Regardless of operating system, the PATH variable needs to be configured to include the following paths (where *<path_to_adt_installation>* represents the file system location into which the ADT bundle was installed):

```
<path_to_adt_installation>/sdk/tools
<path_to_adt_installation>/sdk/platform-tools
```

The steps to achieve this are operating system dependent:

2.6.1 Windows 7

- 1. Right click on *Computer* in the desktop start menu and select *Properties* from the resulting menu.
- 2. In the properties panel, select the *Advanced System Settings* link and, in the resulting dialog, click on the *Environment Variables...* button.
- 3. In the Environment Variables dialog, locate the *Path* variable in the *System variables* list, select it and click on *Edit*.... Locate the end of the current variable value string and append the path to the android platform tools to the end, using a semicolon to separate the path from the preceding values. For example, assuming the ADT bundle was installed into */Users/demo/adt-bundle-windows-x86_64*, the following would be appended to the end of the current Path value:

;C:\Users\demo\adt-bundle-windows-x86_64\sdk\platformtools;C:\Users\demo\adt-bundle-windows-x86_64\sdk\tools

4. Click on OK in each dialog box and close the system properties control panel.

Once the above steps are complete, verify that the path is correctly set by opening a *Command Prompt* window (*Start -> All Programs -> Accessories -> Command Prompt*) and at the prompt enter:

echo %Path%

The returned path variable value should include the paths to the Android SDK platform tools folders. Verify that the *platform-tools* value is correct by attempting to run the *adb* tool as follows:

adb

The tool should output a list of command line options when executed.

Similarly, check the tools path setting by attempting to launch the Android SDK Manager:

android

In the event that a message similar to following message appears for one or both of the commands, it is most likely that an incorrect path was appended to the Path environment variable:

'adb' is not recognized as an internal or external command, operable program or batch file.

2.6.2 Windows 8

- 1. On the start screen, move the mouse to the bottom right hand corner of the screen and select *Search* from the resulting menu. In the search box, enter *Control Panel*. When the Control Panel icon appears in the results area, click on it to launch the tool on the desktop.
- 2. Within the Control Panel, use the *Category* menu to change the display to *Large Icons*. From the list of icons select, the one labeled *System*.
- 3. Follow the steps outlined for Windows 7 starting from step 2.

2.6.3 Linux

On Linux this will involve once again editing the *.bashrc* file. Assuming that the bundle package was installed into */home/demo/adt-bundle-linux-x86*, the export line in the *.bashrc* file would now read as follows:

```
export PATH=/home/demo/java/jdk1.7.0_10/bin:/home/demo/adt-bundle-linux-
x86/sdk/platform-tools:/home/demo/adt-bundle-linux-x86/sdk/tools:$PATH
```

2.6.4 Mac OS X

A number of techniques may be employed to modify the \$PATH environment variable on Mac OS X. Arguably the cleanest method is to add a new file in the */etc/paths.d* directory containing the paths to be added to \$PATH. Assuming an installation location of */Users/demo/adt-bundle-mac-x86_64*, the path may be configured by creating a new file named *android-sdk* in the */etc/paths.d* directory containing the following lines:

```
/Users/demo/adt-bundle-mac-x86_64/sdk/tools
/Users/demo/adt-bundle-mac-x86_64/sdk/platform-tools
```

Note that since this is a system directory it will be necessary to use the *sudo* command when creating the file. For example:

sudo vi /etc/paths.d/android-sdk

2.7 Updating the ADT

From time to time new versions of the Android ADT and SDK are released. New versions of the SDK are installed using the Android SDK Manager. When new versions of the SDK have been installed on your system the ADT will also often need to be updated to a matching version. The latest version of the ADT can be installed by selecting the Eclipse *Help -> Install New Software* menu option. When prompted, enter the following URL and a suitable name for the update (the choice of name is not important):

https://dl-ssl.google.com/android/eclipse/

Having entered the required information Eclipse will list any available updates. If updates are listed, simply proceed with the installation process. Once complete, restart Eclipse to use the latest version of the ADT.

2.8 Adding the ADT Plugin to an Existing Eclipse Integration

The steps outlined so far in this chapter have assumed that the Eclipse IDE is not already installed on your system. In the event that you are already using Eclipse for Java based development, the appropriate Android development tools and SDKs can be added to this existing Eclipse installation. Eclipse editions with which the ADT Plugin is compatible are as follows:

- Eclipse IDE for Java Developers
- Eclipse Classic (versions 3.5.1 and higher)
- Eclipse IDE for Java EE Developers
- Eclipse for Mobile Developers

The ADT Plugin for Eclipse adds a range of Android specific features to what is otherwise a general-purpose Java edition of the Eclipse environment. To install this plugin, launch Eclipse and select the *Help -> Install New Software...* menu option. In the resulting window, click on the *Add...* button to display the *Add Repository* dialog. Enter "ADT Plugin" into the *Name* field and the following URL into the *Location* field:

```
https://dl-ssl.google.com/android/eclipse/
```

Click on the *OK* button and wait while Eclipse connects to the Android repository. Once the information has been downloaded, new items will be listed entitled *Developer Tools* and *NDK Plugins* as illustrated in Figure 2-2:

Install	
Available Software Check the items that you wish to install.	
Work with: ADT Plugin - https://dl-ssl.google.com/a	android/eclipse
Find n	nore software by working with the <u>"Available Software Sites"</u> preferences.
type filter text	
Name	Version
 Developer Tools 000 NDK Plugins 	
•	4
Select All Deselect All	
Details	12
Show only the latest versions of available software	Hide items that are already installed
Group items by category	What is <u>already installed</u> ?
Show only software applicable to target environmen	t
Contact all update sites during install to find require	d software
?	< Back Next > Finish Cancel

Figure 2-2

Select the checkbox next to the *Developer Tools* entry and click on the *Next* > button. After requirements and dependencies have been calculated by the installer, a more detailed list of the packages to be installed will appear. Once again click on the *Next* > button to proceed. On the subsequent licensing page, select the option to accept the terms of the agreements (assuming that you do, indeed, agree) and click on *Finish* to complete the installation. During the download and installation process, you may be prompted to confirm that you wish to install unsigned content. In the event that this happens, simply click on the option to proceed with the installation.

When the ADT Plugin installation is complete, a dialog will appear providing the option to restart Eclipse in order to complete the installation. Click on *Yes* and wait for the tool to exit and re-launch.

Upon restarting, the Welcome to Android Development dialog will appear as illustrated in Figure 2-3:

Welcome to Android	Development	_	23
Welcome to Android	l Development	(
To develop for Android against. You may also v	l, you need an Android SDK, and at least one version of the Android AP want additional versions of Android to test with.	'Is to com	pile
Install new SDK			
📝 Install the latest	t available version of Android APIs (supports all the latest features)		
Install Android	2.2, a version which is supported by ~93% phones and tablets		
(You can add ad	dditional platforms using the SDK Manager.)		
Target Location:	C:\Users\nas\android-sdks	Bro	owse
O Use existing SDKs			
Existing Location:		Bro	owse
(?)	< Back Next > Finish	Cano	el

Figure 2-3

At this stage there is no existing SDK installed so the *Use Existing SDKs* choice is not a viable option. Unfortunately, the ADT Plugin does not provide the option at this point to install the SDKs of our choice so we will need to install the latest available SDK version. With this in mind, select the option to install the latest available version of the Android APIs. Make a note of the *Target Location* path and change it if you prefer the SDKs to be installed in a different location, then click *Next*. Choose whether to send usage information to Google, accept all the licensing terms and click on *Install*. The Android SDK Manager will now download and install the latest Android SDKs.

At this point, the Eclipse environment is ready to begin the development of Android applications.

2.9 Summary

Prior to beginning the development of Android based applications, the first step is to set up a suitable development environment. This consists of the Java Development Kit (JDK), Android SDKs, Eclipse IDE and the Android ADT Plugin for Eclipse. In this chapter, we have covered the steps necessary to install these packages on Windows, Mac OS X and Linux.

3. Creating an Android Virtual Device (AVD)

In the course of developing Android apps it will be necessary to compile and run an application multiple times. An Android application may be tested by installing and running it either on a physical device or in an *Android Virtual Device (AVD)* emulator environment. Before an AVD can be used, it must first be created and configured to match the specification of a particular device model. The goal of this chapter, therefore, is to work through the steps involved in creating such a virtual device using the Nexus 7 tablet as a reference example.

3.1 About Android Virtual Devices

AVDs are essentially emulators that allow Android applications to be tested without the necessity to install the application on a physical Android based device. An AVD may be configured to emulate a variety of hardware features including options such as screen size, memory capacity and the presence or otherwise of features such as a camera, GPS navigation support or an accelerometer. As part of the installation process outlined in the previous chapter, a number of emulator template definitions were installed allowing AVDs to be configured for a range of different devices. Additional templates may be loaded or custom configurations created to match any physical Android device by specifying properties such as process type, memory capacity, screen size and density. Check the online developer documentation for your device to find out if emulator definitions are available for download and installation into the ADT environment.

When launched, an AVD will appear as a window containing an emulated Android device environment. Figure 3-1, for example, shows an AVD session configured to emulate the Google Nexus 7 device.

New AVDs are created and managed using the Android Virtual Device Manager, which may be used either in command-line mode or with a more user-friendly graphical user interface.



Figure 3-1

3.2 Creating a New AVD

In order to test the behavior of an application, it will be necessary to create an AVD for an Android device configuration.

To create a new AVD, the first step is to launch the AVD Manager. This can be achieved from within the Eclipse environment using the *Window -> Android Virtual Device Manager* menu option. Alternatively, the tool may be launched from the command-line using the following command:

android avd

Once launched, the tool will appear as outlined in Figure 3-2. Assuming a new Android SDK installation, no AVDs will currently be listed:

ð	Android Virtual Device Manager								
To	Tools								
A	Android Virtual Devices Device Definitions								
	List of existing A	Android	Virtual Devices	located at C	:\Users\nas	\.android\avd	1		
	AVD Name	Target	Name	Platform	API Level	CPU/ABI	New		
	No AVD available Edit								
	Delete								
							Repair		
							Details		
							Start		
	Refresh								
	🗸 A valid Android Virtual Device. 📩 A repairable Android Virtual Device.								
	× An Android Virtual Device that failed to load. Click 'Details' to see the error.								

Figure 3-2

Begin the AVD creation process by clicking on the *New...* button in order to invoke the *Create a New Android Virtual Device (AVD)* dialog. Within the dialog, perform the following steps to create a Nexus 7 compatible emulator:

- 1. Enter a descriptive name (for example *Nexus7*) into the name field. Note that spaces and other special characters are not permitted in the name.
- 2. Set the Device menu to Nexus 7 (7.27" 800 x 1280: tvhdpi).
- 3. Set the *Target* menu to *Android* 4.2.2 API Level 17.
- 4. Set the *CPU/ABI* menu to *ARM* (armeabi-v7a).
- 5. Leave the default *RAM* value in *Memory Options* and the *Internal Storage* value unchanged from the default settings. Keep in mind however, that it may be necessary to reduce the RAM value below 768M on Windows systems with less available memory.
- 6. If the host computer contains a web cam the *Front Camera:* emulation may be configured to use this camera. Alternatively, an emulated camera may be selected. If camera functionality is not required by the application, simply leave this set to *None*.

Whether or not you enable the *Hardware Keyboard* and *Display skin with hardware controls* options is optional. When the hardware keyboard option is selected, it will be possible to use the physical keyboard on the system on which the emulator is running. As such, the Android software keyboard will not appear within the emulator.

The skin with hardware controls option controls whether or not buttons appear as part of the emulator to simulate the hardware buttons present on the sides of the physical Android device.

Note that it may also be possible to speed the performance of the emulator by enabling the *Use Host GPU* option. In the event that the emulator crashes during startup when this option is selected, edit the virtual device properties and disable this option.

Figure 3-3 illustrates the dialog with the appropriate settings implemented for a Nexus 7 emulator. Once the configuration settings are complete, click on the *OK* button.

() Create new Andro	id Virtual Device (AVD)
AVD Name:	Nexus
Device:	Nexus 7 (7.27", 800 × 1280: tvdpi)
Target:	Android 4.2.2 - API Level 17 🔹
CPU/ABI:	ARM (armeabi-v7a)
Keyboard:	Hardware keyboard present
Skin:	Display a skin with hardware controls
Front Camera:	None
Back Camera:	None 👻
Memory Options:	RAM: 1024 VM Heap: 32
Internal Storage:	200 MiB 🔻
SD Card:	
	● Size: MiB ▼
	© File: Browse
Emulation Options:	Snapshot Use Host GPU
Override the exist	ing AVD with the same name
	OK Cancel



With the AVD created, the AVD Manager may now be closed. If future modifications to the AVD are necessary, simply re-open the AVD Manager, select the AVD from the list and click on the *Edit...* button.

3.3 Starting the Emulator

To perform a test run of the newly created AVD emulator, simply select the emulator from the Android Virtual Device Manager and click on the *Start...* button followed by *Launch* in the resulting *Launch Options* dialog. The emulator will appear in a new window and, after a short period of time, the "android" logo will appear in the center of the screen. The first time the emulator is run, it can take up to 10 minutes for the emulator to fully load and start. On subsequent invocations, this will typically reduce to a few minutes. In the event that the startup time on your system is considerable, do not hesitate to leave the emulator running. The ADT system will detect that it is already running and attach to it when applications are launched, thereby saving considerable amounts of startup time.

Once fully loaded, the emulator will display either the standard Android lock screen.

3.4 AVD Command-line Creation

As previously discussed, in addition to the graphical user interface it is also possible to create a new AVD directly from the command-line. This is achieved using the *android* tool in conjunction with some command-line options. Once initiated, the tool will prompt for additional information before creating the new AVD.

Assuming that the system has been configured such that the Android SDK *tools* directory is included in the PATH environment variable, a list of available targets for the new AVD may be obtained by issuing the following command in a terminal or command window:

```
android list targets
```

The resulting output from the above command will contain a list of Android SDK versions that are available on the system. For example:

```
Available Android targets:
_____
id: 1 or "android-17"
    Name: Android 4.2.2
    Type: Platform
    API level: 17
    Revision: 2
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WSVGA, WVGA800 (default),
WVGA854, WXGA720, WXGA800, WXGA800-7in
    ABIs : armeabi-v7a
_____
id: 2 or "Google Inc.:Google APIs:17"
    Name: Google APIs
    Type: Add-On
    Vendor: Google Inc.
    Revision: 3
    Description: Android + Google APIs
    Based on Android 4.2.2 (API level 17)
    Libraries:
      * com.google.android.media.effects (effects.jar)
          Collection of video effects
      * com.android.future.usb.accessory (usb.jar)
         API for USB Accessories
      * com.google.android.maps (maps.jar)
         API for Google Maps
     Skins: WVGA854, WQVGA400, WSVGA, WXGA800-7in, WXGA720, HVGA, WQVGA432,
WVGA800 (default), QVGA, WXGA800
     ABIs : armeabi-v7a
 _____
```

The syntax for AVD creation is as follows:

android create avd -n <name> -t <targetID> [-<option> <value>]

For example, to create a new AVD named *Nexus7* using the target id for the Android 4.2.2 API level 17 device (in this case id 1), the following command may be used:

android create avd -n Nexus7 -t 1

The android tool will create the new AVD to the specifications required for an Android 4.2.2 device. Once a new AVD has been created from the command line, it may not show up in the Android Device Manager tool until the *Refresh* button is clicked.

In addition to the creation of new AVDs, a number of other tasks may be performed from the command line. For example, a list of currently available AVDs may be obtained using the *list avd* command line arguments:

```
android list avd
C:\Users\nas>android list avd
Available Android Virtual Devices:
_____
   Name: KindleFireHD7
    Path: C:\Users\nas\.android\avd\KindleFireHD7.avd
  Target: Kindle Fire HD 7" (Amazon)
         Based on Android 4.0.3 (API level 15)
    ABI: armeabi-v7a
   Skin: 800x1280
_____
   Name: Nexus7
   Path: C:\Users\nas\.android\avd\Nexus7.avd
 Target: Android 4.2.2 (API level 17)
    ABI: armeabi-v7a
   Skin: 800x1280
_____
   Name: Nexus7Google
    Path: C:\Users\nas\.android\avd\Nexus7Google.avd
  Target: Google APIs (Google Inc.)
         Based on Android 4.2.2 (API level 17)
    ABI: armeabi-v7a
    Skin: 800x1280
```

Similarly, to delete an existing AVD, simply use the *delete* option as follows:

android delete avd -name <avd name>

3.5 Android Virtual Device Configuration Files

By default, the files associated with an AVD are stored in the *.android/avd* sub-directory of the user's home directory, the structure of which is as follows (where *<avd name>* is replaced by the name assigned to the AVD):

```
<avd name>.avd/config.ini
<avd name>.avd/userdata.img
```

<avd name>.ini

The *config.ini* file contains the device configuration settings such as display dimensions and memory specified during the AVD creation process. These settings may be changed directly within the configuration file and will be adopted by the AVD when it is next invoked.

The *<avd name>.ini* file contains a reference to the target Android SDK and the path to the AVD files. Note that a change to the *image.sysdir* value in the *config.ini* file will also need to be reflected in the *target* value of this file.

3.6 Moving and Renaming an Android Virtual Device

The current name or the location of the AVD files may be altered from the command line using the *android* tool's *move avd* argument. For example, to rename an AVD named Nexus7 to Nexus7B, the following command may be executed:

android move avd -n Nexus7 -r Nexus7B

To physically relocate the files associated with the AVD, the following command syntax should be used:

android move avd -n <avd name> -p <path to new location>

For example, to move an AVD from its current file system location to /tmp/Nexus7Test:

android move avd -n Nexus7 -p /tmp/Nexus7Test

Note that the destination directory must not already exist prior to executing the command to move an AVD.

3.7 Summary

A typical application development process follows a cycle of coding, compiling and running in a test environment. Android applications may be tested on either a physical Android device or using an Android Virtual Device (AVD) emulator. AVDs are created and managed using the Android AVD Manager tool which may be used either as a command line tool or using a graphical user interface. When creating an AVD to simulate a specific Android device model it is important that the virtual device be configured with a hardware specification that matches that of the physical device.

Now that we have created and configured an AVD, the next step is to create a simple application and run it within the AVD.

Chapter 4

4. Creating an Example Android Application

The preceding chapters of this book have covered the steps necessary to configure an environment suitable for the development of Android applications. Before moving on to slightly more advanced topics, now is a good time to validate that all of the required development packages are installed and functioning correctly. The best way to achieve this goal is to create a simple Android application, compile it and then run it within an Android Virtual Device (AVD) emulator.

4.1 Creating a New Android Project

The first step in the application development process is to create a new project within the Eclipse IDE. Begin, therefore, by launching Eclipse and accepting the default path to your workspace in the *Workspace Launcher* dialog as illustrated in Figure 4-1 (or choose another location if the default is unsuitable). Note that if you do not wish to be prompted for the location of the workspace each time Eclipse loads, simply select the *Use this* as the default and do not ask again option before clicking on *OK*.

() Workspace Launcher	X
Select a workspace	
ADT stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.	
Workspace: C:\Users\nas\workspace	Browse
Use this as the default and do not ask again	OK Cancel



Once the workspace has been selected, the main Eclipse workbench window will appear ready for a new project to be created. To create the new project, select the *File -> New -> Android Application Project* menu option.

4.2 Defining the Project Name and SDK Settings

In the New Android Project window set both the Application Name and Project Name to AndroidTest.

The *Package Name* is used to uniquely identify the application within the Android application ecosystem. It should be based on the reversed URL of your domain name followed by the name of the application. For

example, if your domain is *www.mycompany.com*, and the application has been named *AndroidTest*, then the package name might be specified as:

```
com.mycompany.androidtest
```

If you do not have a domain name, you may also use *example.com* for the purposes of testing, though this will need to be changed before an application can be published:

```
com.example.androidtest
```

The next step is to specify some SDK settings. For the purposes of this example, the *Minimum Required SDK*, *Target SDK* and *Compile With* menus should all be set to *API 17: Android 4.2 (Jelly Bean)*. Once these settings have been configured, the dialog should match that shown in Figure 4-2:

O New Android Application	1			3
New Android Applicatio	n e.' is meant as a placeholder and should not be used		F	1
Application Name:0	AndroidTest			
Project Name:	AndroidTest			
Package Name: 💩	com.example.androidtest			
Minimum Required SDK:0	API 17: Android 4.2 (Jelly Bean)	•		
Target SDK:0	API 17: Android 4.2 (Jelly Bean)	•		
Compile With:	API 17: Android 4.2 (Jelly Bean)	-		
Theme:0	Holo Light with Dark Action Bar	•		
?	< Back Next >	Finish	Cancel	
<u> </u>				



4.3 Project Configuration Settings

With the correct settings configured, click Next > to proceed to the *Configure Project* screen (Figure 4-3). Within this screen, a number of different configuration options are provided.

Make sure that the *Create Activity* and *Create customer launcher icon* options are selected. The former setting will ensure that the project is preconfigured with a template activity that will make the task of creating an example application easier. An activity is a single task that can be performed by the user within the context of an application and is typically analogous to a single user interface screen within an application. In asking for Eclipse to create an activity for us, therefore, the project will be primed with both a window onto which a user interface may be displayed and the code to ensure that the window appears when the application runs.