

Android 4.4 App Development Essentials

Android 4.4 App Development Essentials – First Edition

ISBN-13: 978-1-4953580-6-7

© 2014 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 2.0



Table of Contents

1. Introduction	1
1.1 Downloading the Code Samples	1
1.2 Feedback.....	2
1.3 Errata	2
2. Setting up an Android Development Environment	3
2.1 System Requirements	3
2.2 Installing the Java Development Kit (JDK).....	3
<i>2.2.1 Windows JDK Installation</i>	<i>4</i>
<i>2.2.2 Mac OS X JDK Installation</i>	<i>4</i>
2.3 Linux JDK Installation	5
2.4 Downloading the Android Developer Tools (ADT) Bundle.....	6
2.5 Installing the ADT Bundle.....	7
<i>2.5.1 Installation on Windows</i>	<i>7</i>
<i>2.5.2 Installation on Mac OS X</i>	<i>7</i>
<i>2.5.3 Installation on Linux</i>	<i>8</i>
2.6 Installing the Latest Android SDK Packages	9
2.7 Making the Android SDK Tools Command-line Accessible	10
<i>2.7.1 Windows 7</i>	<i>11</i>
<i>2.7.2 Windows 8.1</i>	<i>11</i>
<i>2.7.3 Linux</i>	<i>12</i>
<i>2.7.4 Mac OS X</i>	<i>12</i>
2.8 Updating the ADT	13
2.9 Adding the ADT Plugin to an Existing Eclipse Integration	13
2.10 Summary.....	15
3. Creating an Android Virtual Device (AVD)	17
3.1 About Android Virtual Devices.....	17
3.2 Creating a New AVD.....	18
3.3 Starting the Emulator.....	21
3.4 AVD Command-line Creation	21
3.5 Android Virtual Device Configuration Files.....	23
3.6 Moving and Renaming an Android Virtual Device	23
3.7 Summary.....	23
4. Creating an Example Android Application	25
4.1 Creating a New Android Project	25
4.2 Defining the Project Name and SDK Settings.....	26

4.3 Project Configuration Settings.....	27
4.4 Configuring the Launcher Icon.....	28
4.5 Creating an Activity.....	29
4.6 Running the Application in the AVD	30
4.7 Stopping a Running Application.....	32
4.8 Modifying the Example Application.....	34
4.9 Reviewing the Layout and Resource Files.....	39
4.10 Summary.....	41
5. Testing Android Applications on a Physical Android Device with ADB	43
5.1 An Overview of the Android Debug Bridge (ADB)	43
5.2 Enabling ADB on Android 4.4 based Devices	44
5.2.1 <i>Mac OS X ADB Configuration</i>	45
5.2.2 <i>Windows ADB Configuration</i>	46
5.2.3 <i>Linux adb Configuration</i>	49
5.3 Testing the adb Connection.....	51
5.4 Manual Selection of the Application Run Target.....	51
5.5 Summary.....	53
6. An Overview of the Android Architecture.....	55
6.1 The Android Software Stack	55
6.2 The Linux Kernel	56
6.3 Android Runtime - Dalvik Virtual Machine	57
6.4 Android Runtime – Core Libraries	57
6.4.1 <i>Dalvik VM Specific Libraries</i>	57
6.4.2 <i>Java Interoperability Libraries</i>	57
6.4.3 <i>Android Libraries</i>	58
6.4.4 <i>C/C++ Libraries</i>	59
6.5 Application Framework	59
6.6 Applications	60
6.7 Summary.....	60
7. The Anatomy of an Android Application.....	61
7.1 Android Activities	61
7.2 Android Intents.....	62
7.3 Broadcast Intents.....	62
7.4 Broadcast Receivers.....	62
7.5 Android Services	63
7.6 Content Providers.....	63
7.7 The Application Manifest.....	63
7.8 Application Resources	64

7.9 Application Context	64
7.10 Summary.....	64
8. Understanding Android Application and Activity Lifecycles.....	65
8.1 Android Applications and Resource Management	65
8.2 Android Process States	66
8.2.1 <i>Foreground Process</i>	66
8.2.2 <i>Visible Process</i>	67
8.2.3 <i>Service Process</i>	67
8.2.4 <i>Background Process</i>	67
8.2.5 <i>Empty Process</i>	67
8.3 Inter-Process Dependencies	67
8.4 The Activity Lifecycle.....	67
8.5 The Activity Stack.....	68
8.6 Activity States	69
8.7 Configuration Changes.....	69
8.8 Handling State Change.....	70
8.9 Summary.....	70
9. Handling Android Activity State Changes	71
9.1 The Activity Class	71
9.2 Dynamic State vs. Persistent State	72
9.3 The Android Activity Lifecycle Methods	73
9.4 Activity Lifetimes	75
9.5 Summary.....	76
10. Android Activity State Changes by Example	77
10.1 Creating the State Change Example Project	77
10.2 Designing the User Interface	78
10.3 Overriding the Activity Lifecycle Methods.....	79
10.4 Enabling and Filtering the LogCat Panel	81
10.5 Running the Application	82
10.6 Experimenting with the Activity	83
10.7 Saving Dynamic State.....	84
10.8 Summary.....	84
11. Saving and Restoring the User Interface State of an Android Activity	87
11.1 Saving Dynamic State.....	87
11.2 The Bundle Class	87
11.3 Saving the State	88
11.4 Restoring the State	90
11.5 Testing the Application	90

11.6 Summary.....	90
12. Understanding Android Views, View Groups and Layouts	93
12.1 Designing for Different Android Devices	93
12.2 Views and View Groups	93
12.3 Android Layout Managers	94
12.4 The View Hierarchy.....	95
12.5 Creating User Interfaces	97
12.6 Summary.....	97
13. Designing an Android User Interface using the Graphical Layout Tool	99
13.1 The Android Graphical Layout Tool	99
13.2 A Graphical Layout Tool Example	99
13.3 Adding an XML Resource File to the Project.....	100
13.4 Editing View Properties	102
13.5 Using the View Properties Sheet	103
13.6 Creating a New Activity	104
13.7 Adding the New Activity to the Manifest File.....	106
13.8 Running the Application	108
13.9 Manually Creating an XML Layout.....	109
13.10 Using the Hierarchy Viewer	111
13.11 Summary.....	113
14. Creating an Android User Interface in Java Code	115
14.1 Java Code vs. XML Layout Files	115
14.2 Creating Views	116
14.3 Properties and Layout Parameters	116
14.4 Creating the Example Project	117
14.5 Adding Views to an Activity	117
14.6 Setting View Properties	118
14.7 Adding Layout Parameters and Rules.....	119
14.8 Using View IDs	122
14.9 Converting Density Independent Pixels (dp) to Pixels (px)	123
14.10 Summary.....	125
15. Using the Android GridLayout Manager in the Graphical Layout Tool	127
15.1 Introducing the Android GridLayout and Space Classes	127
15.2 The GridLayout Example.....	128
15.3 Creating the GridLayout Project	128
15.4 Creating the GridLayout Instance	128
15.5 An Overview of the GridLayout in the Graphical Layout Tool	129
15.6 Adding Views to GridLayout Cells	130

15.7 Implementing Cell Row and Column Spanning	132
15.8 Changing the Gravity of a GridLayout Child.....	133
15.9 Summary.....	136
16. Working with the Android GridLayout in XML Layout Resources.....	137
16.1 GridLayouts in XML Resource Files	137
16.2 Adding Child Views to the GridLayout	138
16.3 Declaring Cell Spanning, Gravity and Margins	139
16.4 Summary.....	141
17. An Overview and Example of Android Event Handling	143
17.1 Understanding Android Events	143
17.2 Using the android:onClick Resource	144
17.3 Event Listeners and Callback Methods	144
17.4 An Event Handling Example	145
17.5 Designing the User Interface	145
17.6 The Event Listener and Callback Method	148
17.7 Consuming Events.....	149
17.8 Summary.....	151
18. Android Touch and Multi-touch Event Handling.....	153
18.1 Intercepting Touch Events	153
18.2 The MotionEvent Object.....	154
18.3 Understanding Touch Actions.....	154
18.4 Handling Multiple Touches	154
18.5 An Example Multi-Touch Application	155
18.6 Designing the Activity User Interface	155
18.7 Implementing the Touch Event Listener	157
18.8 Running the Example Application.....	161
18.9 Summary.....	161
19. Detecting Common Gestures using the Android Gesture Detector Class	163
19.1 Implementing Common Gesture Detection.....	163
19.2 Creating an Example Gesture Detection Project	164
19.3 Implementing the Listener Class.....	164
19.4 Creating the GestureDetectorCompat Instance	168
19.5 Implementing the onTouchEvent() Method	169
19.6 Testing the Application	169
19.7 Summary.....	170
20. Implementing Custom Gesture and Pinch Recognition on Android	171
20.1 The Android Gesture Builder Application	171

20.2 The GestureOverlayView Class	171
20.3 Detecting Gestures	171
20.4 Identifying Specific Gestures	172
20.5 Adding SD Card Support to an AVD	172
20.6 Building and Running the Gesture Builder Application	173
20.7 Creating a Gestures File.....	174
20.8 Extracting the Gestures File from the SD Card	175
20.9 Creating the Example Project	176
20.10 Designing the User Interface	176
20.11 Loading the Gestures File	177
20.12 Registering the Event Listener	178
20.13 Implementing the onGesturePerformed Method	179
20.14 Testing the Application.....	181
20.15 Configuring the GestureOverlayView	181
20.16 Intercepting Gestures	182
20.17 Detecting Pinch Gestures	182
20.18 A Pinch Gesture Example Project	182
20.19 Summary.....	185
21. An Introduction to Android Fragments	187
21.1 What is a Fragment?.....	187
21.2 Creating a Fragment	188
21.3 Adding a Fragment to an Activity using the Layout XML File	189
21.4 Adding and Managing Fragments in Code	192
21.5 Handling Fragment Events.....	193
21.6 Implementing Fragment Communication.....	194
21.7 Summary.....	196
22. Using Fragments in Android - A Worked Example.....	197
22.1 About the Example Fragment Application.....	197
22.2 Creating the Example Project	197
22.3 Adding the Android Support Library.....	198
22.4 Creating the First Fragment Layout	199
22.5 Creating the First Fragment Class	202
22.6 Creating the Second Fragment Layout	203
22.7 Adding the Fragments to the Activity	205
22.8 Making the Toolbar Fragment Talk to the Activity	206
22.9 Making the Activity Talk to the Text Fragment	211
22.10 Testing the Application.....	212
22.11 Summary.....	213
23. An Android Master/Detail Flow Tutorial.....	215

23.1 The Master/Detail Flow	215
23.2 Creating a Master/Detail Flow Activity	217
23.3 The Anatomy of the Master/Detail Flow Template	218
23.4 Modifying the Master/Detail Flow Template	220
23.5 Changing the Content Model.....	220
23.6 Changing the Detail Pane.....	222
23.7 Modifying the WebsiteDetailFragment Class	222
23.8 Adding Manifest Permissions	224
23.9 Running the Application	224
23.10 Summary.....	225
24. Creating and Managing Overflow Menus on Android.....	227
24.1 The Overflow Menu	227
24.2 Creating an Overflow Menu.....	228
24.3 Displaying an Overflow Menu.....	229
24.4 Responding to Menu Item Selections	229
24.5 Creating Checkable Item Groups	230
24.6 Creating the Example Project	232
24.7 Modifying the Menu Description	232
24.8 Implementing the onMenuItemSelected() Method	233
24.9 Testing the Application	234
24.10 Summary.....	235
25. Animating User Interfaces with the Android Transitions Framework	237
25.1 Introducing Android Transitions and Scenes	237
25.2 Using Interpolators with Transitions.....	238
25.3 Working with Scene Transitions	239
25.4 Custom Transitions and TransitionSets in Code	240
25.5 Custom Transitions and TransitionSets in XML.....	241
25.6 Working with Interpolators	243
25.7 Creating a Custom Interpolator	245
25.8 Using the beginDelayedTransition Method	246
25.9 Summary.....	246
26. An Android Transition Tutorial using beginDelayedTransition	247
26.1 Creating the TransitionDemo Project	247
26.2 Preparing the Project Files.....	247
26.3 Implementing beginDelayedTransition Animation	248
26.4 Customizing the Transition	251
26.5 Summary.....	252
27. Implementing Android Scene Transitions – A Tutorial.....	253

27.1 An Overview of the Scene Transition Project	253
27.2 Creating the SceneTransitions Project.....	253
27.3 Identifying and Preparing the Root Container.....	253
27.4 Designing the First Scene.....	254
27.5 Designing the Second Scene	256
27.6 Entering the First Scene.....	258
27.7 Loading Scene 2	259
27.8 Implementing the Transitions.....	260
27.9 Adding the Transition File.....	260
27.10 Loading and Using the Transition Set	261
27.11 Configuring Additional Transitions	262
27.12 Summary.....	263
28. An Overview of Android Intents	265
28.1 An Overview of Intents	265
28.2 Explicit Intents	266
28.3 Returning Data from an Activity	267
28.4 Implicit Intents.....	268
28.5 Using Intent Filters.....	269
28.6 Checking Intent Availability	270
28.7 Summary.....	270
29. Android Explicit Intents – A Worked Example.....	271
29.1 Creating the Explicit Intent Example Application	271
29.2 Designing the User Interface Layout for ActivityA.....	271
29.3 Creating the Second Activity Class.....	273
29.4 Creating the User Interface for ActivityB.....	274
29.5 Adding ActivityB to the Application Manifest File	276
29.6 Creating the Intent	277
29.7 Extracting Intent Data.....	278
29.8 Launching ActivityB as a Sub-Activity	279
29.9 Returning Data from a Sub-Activity	280
29.10 Testing the Application	281
29.11 Summary.....	281
30. Android Implicit Intents – A Worked Example	283
30.1 Creating the Implicit Intent Example Project.....	283
30.2 Designing the User Interface	283
30.3 Creating the Implicit Intent.....	285
30.4 Adding a Second Matching Activity	286
30.5 Adding the Web View to the UI	286

30.6 Obtaining the Intent URL	287
30.7 Modifying the MyWebView Project Manifest File.....	289
30.8 Installing the MyWebView Package on a Device	291
30.9 Testing the Application	291
30.10 Summary.....	292
31. Android Broadcast Intents and Broadcast Receivers	293
31.1 An Overview of Broadcast Intents	293
31.2 An Overview of Broadcast Receivers	294
31.3 Obtaining Results from a Broadcast	296
31.4 Sticky Broadcast Intents.....	296
31.5 The Broadcast Intent Example	296
31.6 Creating the Example Application.....	297
31.7 Creating and Sending the Broadcast Intent	297
31.8 Creating the Broadcast Receiver.....	298
31.9 Configuring a Broadcast Receiver in the Manifest File	299
31.10 Testing the Broadcast Example.....	301
31.11 Listening for System Broadcasts	301
31.12 Summary.....	303
32. A Basic Overview of Android Threads and Thread Handlers	305
32.1 An Overview of Threads.....	305
32.2 The Application Main Thread.....	305
32.3 Thread Handlers.....	305
32.4 A Basic Threading Example	306
32.5 Creating a New Thread	309
32.6 Implementing a Thread Handler.....	310
32.7 Passing a Message to the Handler	312
32.8 Summary.....	313
33. An Overview of Android Started and Bound Services	315
33.1 Started Services	315
33.2 Intent Service.....	316
33.3 Bound Service	316
33.4 The Anatomy of a Service	317
33.5 Controlling Destroyed Service Restart Options	317
33.6 Declaring a Service in the Manifest File.....	318
33.7 Starting a Service Running on System Startup.....	319
33.8 Summary.....	319
34. Implementing an Android Started Service – A Worked Example	321
34.1 Creating the Example Project	321

34.2 Creating the Service Class.....	321
34.3 Adding the Service to the Manifest File.....	323
34.4 Starting the Service.....	324
34.5 Testing the IntentService Example	325
34.6 Using the Service Class	326
34.7 Creating the New Service	326
34.8 Modifying the User Interface.....	328
34.9 Running the Application	330
34.10 Creating a New Thread for Service Tasks	330
34.11 Summary.....	331
35. Android Local Bound Services – A Worked Example	333
35.1 Understanding Bound Services.....	333
35.2 Bound Service Interaction Options.....	333
35.3 A Local Bound Service Example	334
35.4 Adding a Bound Service to the Project	334
35.5 Implementing the Binder.....	334
35.6 Binding the Client to the Service	337
35.7 Completing the Example.....	339
35.8 Testing the Application.....	341
35.9 Summary.....	341
36. Android Remote Bound Services – A Worked Example	343
36.1 Client to Remote Service Communication.....	343
36.2 Creating the Example Application	343
36.3 Designing the User Interface	344
36.4 Implementing the Remote Bound Service.....	344
36.5 Configuring a Remote Service in the Manifest File.....	346
36.6 Launching and Binding to the Remote Service	347
36.7 Sending a Message to the Remote Service.....	349
36.8 Summary.....	349
37. An Overview of Android SQLite Databases	351
37.1 Understanding Database Tables	351
37.2 Introducing Database Schema	352
37.3 Columns and Data Types	352
37.4 Database Rows	352
37.5 Introducing Primary Keys.....	352
37.6 What is SQLite?.....	353
37.7 Structured Query Language (SQL)	353
37.8 Trying SQLite on an Android Virtual Device (AVD)	354

37.9 Android SQLite Java Classes	356
37.9.1 <i>Cursor</i>	356
37.9.2 <i>SQLiteDatabase</i>	357
37.9.3 <i>SQLiteOpenHelper</i>	357
37.9.4 <i>ContentValues</i>	358
37.10 Summary	358
38. An Android TableLayout and TableRow Tutorial	359
38.1 The TableLayout and TableRow Layout Views	359
38.2 Creating the Database Project	361
38.3 Designing the User Interface Layout	361
38.4 Summary	367
39. An Android SQLite Database Tutorial	369
39.1 About the Database Example	369
39.2 Creating the Data Model	370
39.3 Implementing the Data Handler	371
39.3.1 <i>The Add Handler Method</i>	374
39.3.2 <i>The Query Handler Method</i>	374
39.3.3 <i>The Delete Handler Method</i>	375
39.4 Implementing the Activity Event Methods	376
39.5 Testing the Application	378
39.6 Summary	378
40. Understanding Android Content Providers	379
40.1 What is a Content Provider?	379
40.2 The Content Provider	379
40.2.1 <i>onCreate()</i>	380
40.2.2 <i>query()</i>	380
40.2.3 <i>insert()</i>	380
40.2.4 <i>update()</i>	380
40.2.5 <i>delete()</i>	380
40.2.6 <i>getType()</i>	380
40.3 The Content URI	380
40.4 The Content Resolver	381
40.5 The <provider> Manifest Element	381
40.6 Summary	382
41. Implementing an Android Content Provider	383
41.1 Copying the Database Project	383
41.2 Adding the Content Provider Package	383
41.3 Creating the Content Provider Class	384

41.4 Constructing the Authority and Content URI.....	386
41.5 Implementing URI Matching in the Content Provider	386
41.6 Implementing the Content Provider onCreate() Method.....	388
41.7 Implementing the Content Provider insert() Method	389
41.8 Implementing the Content Provider query() Method	390
41.9 Implementing the Content Provider update() Method	392
41.10 Implementing the Content Provider delete() Method	393
41.11 Declaring the Content Provider in the Manifest File	394
41.12 Modifying the Database Handler.....	395
41.13 Summary.....	398
42. Accessing Cloud Storage using the Android Storage Access Framework	399
42.1 The Storage Access Framework	399
42.2 Working with the Storage Access Framework	401
42.3 Filtering Picker File Listings.....	401
42.4 Handling Intent Results.....	403
42.5 Reading the Content of a File	403
42.6 Writing Content to a File	404
42.7 Deleting a File	405
42.8 Gaining Persistent Access to a File	405
42.9 Summary.....	406
43. An Android Storage Access Framework Example.....	407
43.1 About the Storage Access Framework Example	407
43.2 Creating the Storage Access Framework Example	407
43.3 Designing the User Interface	408
43.4 Declaring Request Codes	410
43.5 Creating a New Storage File.....	411
43.6 The onActivityResult() Method.....	412
43.7 Saving to a Storage File.....	414
43.8 Opening and Reading a Storage File	417
43.9 Testing the Storage Access Application	420
43.10 Summary.....	420
44. Implementing Video Playback on Android using the VideoView and MediaController Classes	421
44.1 Introducing the Android VideoView Class	421
44.2 Introducing the Android MediaController Class	422
44.3 Testing Video Playback	422
44.4 Creating the Video Playback Example	423
44.5 Designing the VideoPlayer Layout	423
44.6 Configuring the VideoView	424

44.7 Adding Internet Permission	425
44.8 Adding the MediaController to the Video View	426
44.9 Setting up the onPreparedListener.....	427
44.10 Summary.....	428
45. Video Recording and Image Capture on Android using Camera Intents.....	429
45.1 Checking for Camera Support.....	429
45.2 Calling the Video Capture Intent	430
45.3 Calling the Image Capture Intent	431
45.4 Creating an Example Video Recording Project	432
45.5 Designing the User Interface Layout.....	432
45.6 Checking for the Camera	433
45.7 Launching the Video Capture Intent	434
45.8 Handling the Intent Return	435
45.9 Testing the Application	436
45.10 Summary.....	437
46. Android Audio Recording and Playback using MediaPlayer and MediaRecorder.....	439
46.1 Playing Audio	439
46.2 Recording Audio and Video using the MediaRecorder Class	440
46.3 About the Example Project	441
46.4 Creating the AudioApp Project	442
46.5 Designing the User Interface	442
46.6 Checking for Microphone Availability.....	443
46.7 Performing the Activity Initialization	443
46.8 Implementing the recordAudio() Method	445
46.9 Implementing the stopClicked() Method.....	446
46.10 Implementing the playAudio() method	447
46.11 Configuring Permissions in the Manifest File	447
46.12 Testing the Application	448
46.13 Summary.....	448
47. Working with the Google Maps Android API	449
47.1 The Elements of the Google Maps Android API.....	449
47.2 Getting Ready to use the Google Maps Android API	450
47.2.1 <i>Installing the Google APIs</i>	450
47.2.2 <i>Downloading the Google Play Services SDK</i>	451
47.2.3 <i>Adding the Google Play Services Library Project to the Eclipse Workspace</i>	452
47.2.4 <i>Adding the Google Play Services Library to a Project Build Path</i>	452
47.2.5 <i>Obtaining Your Developer Signature</i>	454
47.2.6 <i>Registering the Project in the Google APIs Console</i>	454

47.3 Adding Map Support to the <code>AndroidManifest.xml</code> File.....	457
47.4 Checking for Google Play Services Support	460
47.5 Understanding Geocoding and Reverse Geocoding	460
47.6 Adding a Map to an Application	462
47.7 Displaying the User's Current Location	463
47.8 Changing the Map Type.....	463
47.9 Displaying Map Controls to the User	464
47.10 Handling Map Gesture Interaction	465
<i>47.10.1 Map Zooming Gestures</i>	465
<i>47.10.2 Map Scrolling/Panning Gestures</i>	465
<i>47.10.3 Map Tilt Gestures</i>	466
<i>47.10.4 Map Rotation Gestures</i>	466
47.11 Creating Map Markers	466
47.12 Controlling the Map Camera	467
47.13 Summary.....	469
48. Printing with the Android Printing Framework	471
48.1 The Android Printing Architecture.....	471
48.2 The HP Print Services Plugin	471
48.3 Google Cloud Print.....	472
48.4 Printing to Google Drive	473
48.5 Save as PDF	473
48.6 Printing from Android Devices.....	473
48.7 Options for Building Print Support into Android Apps	475
<i>48.7.1 Image Printing</i>	475
<i>48.7.2 Creating and Printing HTML Content</i>	476
<i>48.7.3 Printing a Web Page</i>	478
<i>48.7.4 Printing a Custom Document</i>	479
48.8 Summary.....	479
49. An Android HTML and Web Content Printing Example	481
49.1 Creating the HTML Printing Example Application.....	481
49.2 Printing Dynamic HTML Content	481
49.3 Creating the Web Page Printing Example	485
49.4 Designing the User Interface Layout	485
49.5 Loading the Web Page into the <code>WebView</code>	487
49.6 Adding the Print Menu Option	488
49.7 Summary.....	491
50. A Guide to Android Custom Document Printing.....	493
50.1 An Overview of Android Custom Document Printing	493

<i>50.1.1 Custom Print Adapters</i>	494
50.2 Preparing the Custom Document Printing Project	494
50.3 Creating the Custom Print Adapter	496
50.4 Implementing the <code>onLayout()</code> Callback Method	497
50.5 Implementing the <code>onWrite()</code> Callback Method	501
50.6 Checking a Page is in Range	504
50.7 Drawing the Content on the Page Canvas	505
50.8 Starting the Print Job	508
50.9 Testing the Application	509
50.10 Summary.....	510
51. Handling Different Android Devices and Displays.....	511
51.1 Handling Different Device Displays.....	511
51.2 Creating a Layout for each Display Size	511
51.3 Providing Different Images	512
51.4 Checking for Hardware Support	513
51.5 Providing Device Specific Application Binaries	514
51.6 Summary.....	514
52. Signing and Preparing an Android Application for Release	515
52.1 The Release Preparation Process.....	515
52.2 Accessing the Export Wizard.....	515
52.3 Creating a Keystore File	516
52.4 Generating a Private Key	517
52.5 Creating the Application APK File	518
52.6 Register for a Google Play Developer Console Account	519
52.7 Summary.....	520
53. Integrating Google Play In-app Billing into an Android Application	521
53.1 Installing the Google Play Billing Library.....	521
53.2 Creating the Example In-app Billing Project	522
53.3 Adding Billing Permission to the Manifest File	522
53.4 Adding the <code>IInAppBillingService.aidl</code> File to the Project.....	523
53.5 Adding the Utility Classes to the Project.....	524
53.6 Designing the User Interface	526
53.7 Implementing the “Click Me” Button	527
53.8 Google Play Developer Console and Google Wallet Accounts	529
53.9 Obtaining the Public License Key for the Application	529
53.10 Setting Up Google Play Billing in the Application	530
53.11 Initiating a Google Play In-app Billing Purchase.....	532
53.12 Implementing the <code>onActivityResult</code> Method	533

53.13 Implementing the Purchase Finished Listener	534
53.14 Consuming the Purchased Item.....	535
53.15 Releasing the labHelper Instance	536
53.16 Modifying the Security.java File	536
53.17 Testing the In-app Billing Application	538
53.18 Building a Release APK.....	538
53.19 Creating a New In-app Product.....	539
53.20 Adding In-app Billing Test Accounts	540
53.21 Resolving Problems with In-App Purchasing	541
53.22 Summary.....	543
Index	545

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 4.4 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 in-depth 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. This edition of the book also covers features introduced with Android 4.4 including printing, transitions and cloud-based file storage.

In addition to covering general Android development techniques, the book also includes Google Play specific topics such as 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/print/android44/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/android44.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 / Windows 8.1
- 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_65"
Java(TM) SE Runtime Environment (build 1.6.0_65-b14-462-11M4609)
Java HotSpot(TM) 64-Bit Server VM (build 20.65-b04-462, mixed mode)
```

In the event that Java is not installed, issuing the “java” command in the terminal window will result in the appearance of a message which reads as follows together with a dialog on the desktop providing the option to display the Oracle Java web page:

```
No Java runtime present, requesting install
```

The exact steps that need to be taken to install Java vary from one release of Mac OS X to the next so check the Apple documentation for your particular Mac OS X version.

At the time of writing the latest release of Mac OS X is 10.9 (Mavericks). To install Java on this release of Mac OS X, open a Safari browser window and navigate to the following URL:

<http://support.apple.com/kb/DL1572>

This should display the Java for OS X 2013-005 web page. Click on the *Download* button to download the Java package to your system. Open the downloaded disk image (.dmg file) and double click on the *JavaForOSX.pkg* package file (Figure 2-1) contained within:



Figure 2-1

The Java for OS X installer window will appear and take you through the steps involved in installing the JDK. Once the installation is complete, return to the Terminal window and run the following command, at which point the previously outlined Java version information should appear:

```
java -version
```

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-7u45-linux-x64.rpm*, the commands to perform the installation would read as follows:

```
su  
rpm -ihv jdk-7u45-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_45` the following would need to be added to your \$PATH environment variable:

```
/home/demo/java/jdk1.7.0_45/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_45/bin:$PATH
```

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

2.4 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.

2.5 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.

2.5.1 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*.

2.5.2 Installation on Mac OS X

When using Safari to download the ADT bundle archive, it is possible that Safari will automatically unzip the file once the download is complete (this is typically the default setting for Safari these days). This being the case, the location into which the download was saved will contain a directory named as follows:

adt-bundle-mac-x86_64-<version>

Setting up an Android Development Environment

Using the Finder, simply move this folder to a permanent location on your file system.

In the event that the browser did not automatically unzip the bundle archive, 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_64-20131030.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_64-20131030.zip
```

Note that, in the above example, the bundle will be installed into a sub-directory named *adt-bundle-mac-x86_64-20131030*. Assuming, therefore, that the above command was executed in */Users/demo*, the software packages will be unpacked into */Users/demo/adt-bundle-mac-x86_64-20131030*. 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.

2.5.3 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-x86-20131030.zip* has been downloaded to */home/demo/Downloads*, the following command would be needed to install Eclipse:

```
unzip /home/demo/Downloads/adt-bundle-linux-x86-20131030.zip
```

Note that the bundle will be installed into a sub-directory named either *adt-bundle-linux-x86-20131030* or *adt-bundle-linux-x86_64-20131030* 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-x86-20131030*. 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.

2.6 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 2-2:

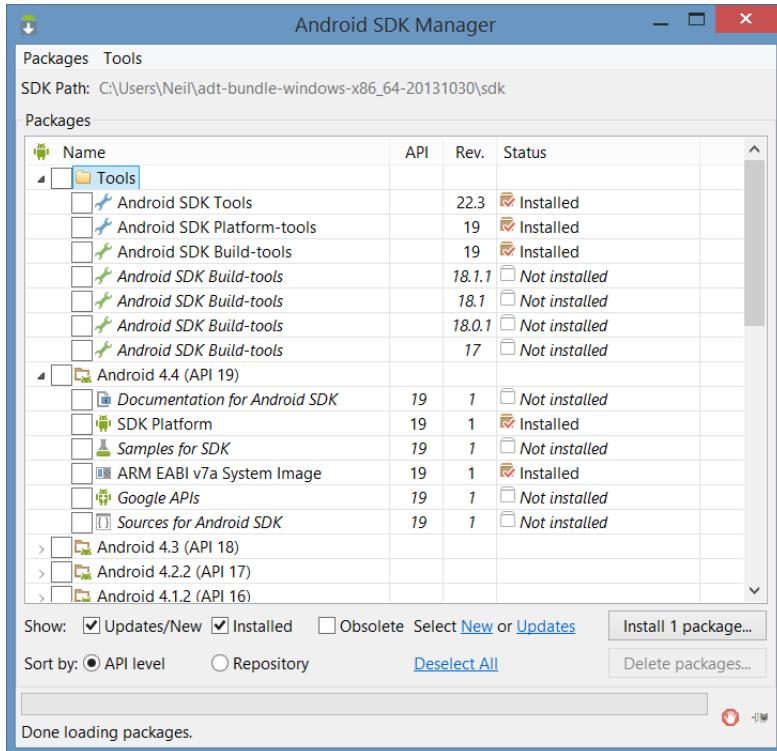


Figure 2-2

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.

Setting up an Android Development Environment

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 listed as *Installed* in the Status column:

- Tools > Android SDK Tools
- Tools > Android SDK Platform-tools
- SDK Platform Android 4.4 (API 19) > SDK Platform
- SDK Platform Android 4.4 (API 19) > ARM EABI v7a System Image
- Extras > Android Support Library

In the event that any of the above packages are listed as *Not Installed*, simply select the checkboxes next to those packages and 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.

2.7 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:

2.7.1 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_64-20131030*, the following would be appended to the end of the current Path value:

```
;C:\Users\demo\adt-bundle-windows-x86_64-20131030\sdk\platform-tools;C:\Users\demo\adt-bundle-windows-x86_64-20131030\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.
```

2.7.2 Windows 8.1

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.

Setting up an Android Development Environment

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 through to step 4.

Open the command prompt window (move the mouse to the bottom right hand corner of the screen, select the Search option and enter *cmd* into the search box). Select *Command Prompt* from the search results.

Within the Command Prompt window, 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.
```

2.7.3 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-x86-20131030*, 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-x86-20131030/sdk/platform-tools:/home/demo/adt-bundle-linux-x86-  
20131030/sdk/tools:$PATH
```

2.7.4 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-*

20131030, 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_64-20131030/sdk/tools  
/Users/demo/adt-bundle-mac-x86_64-20131030/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
```

2.8 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.

2.9 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:

Setting up an Android Development Environment

<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 2-3:

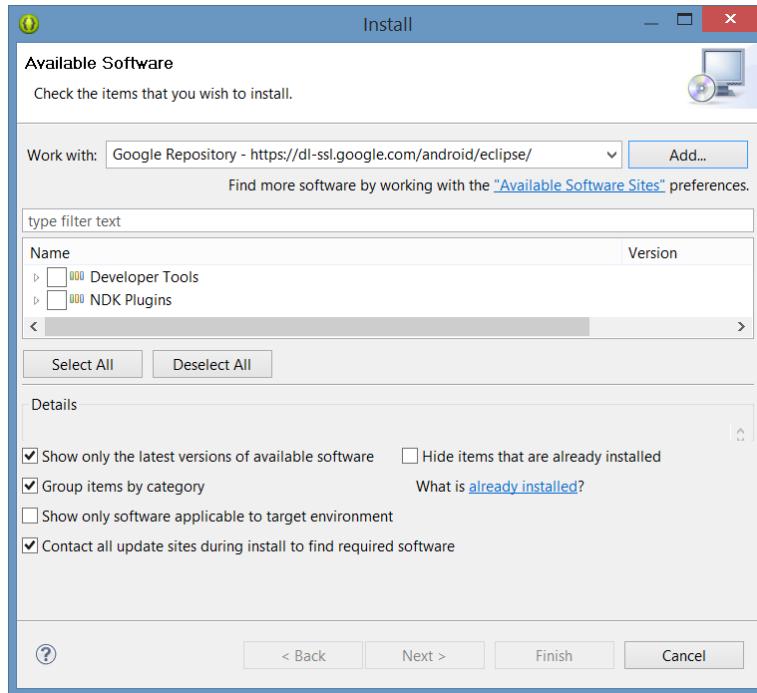


Figure 2-3

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 2-4:

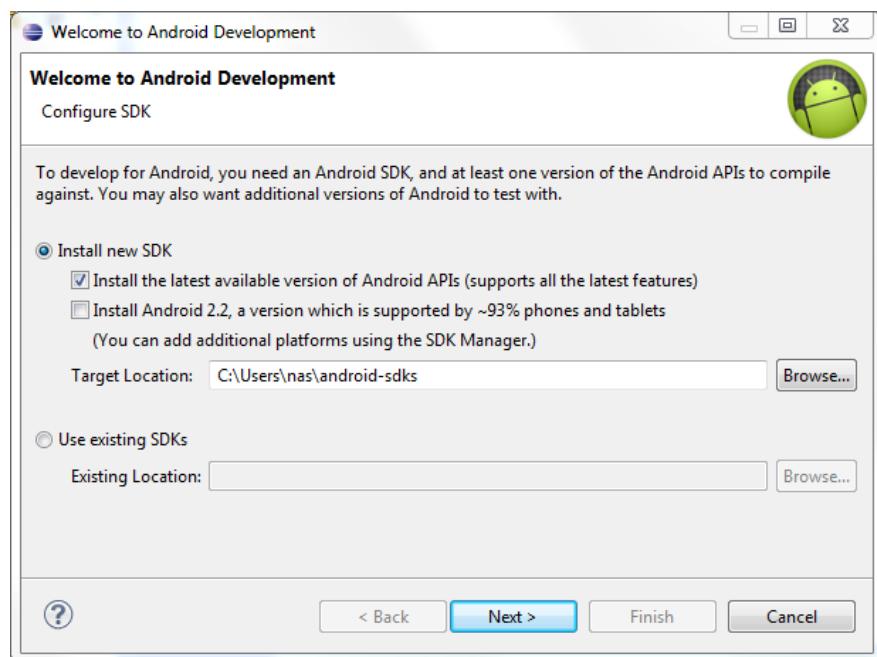


Figure 2-4

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.

2.10 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.

Creating an Android Virtual Device (AVD)

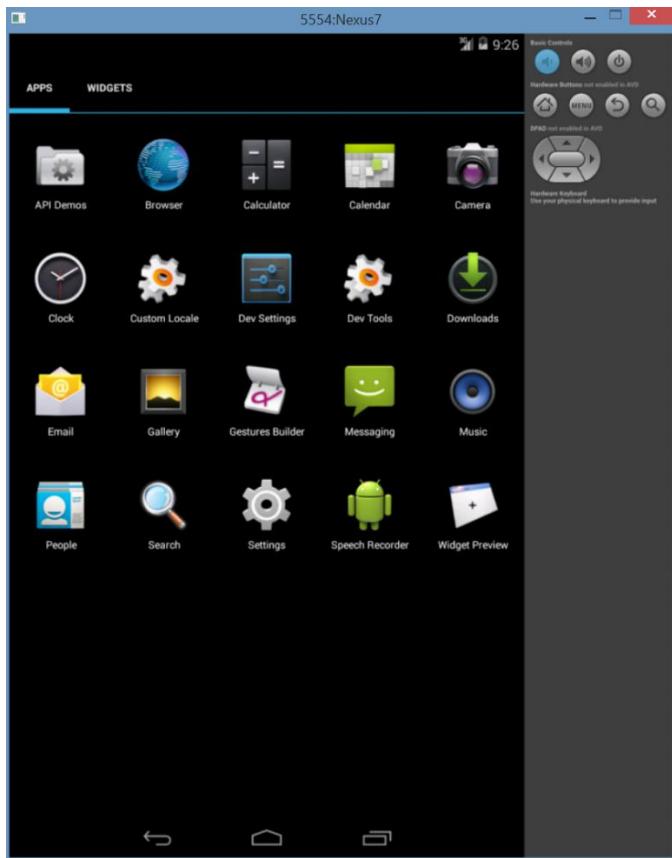


Figure 3-1

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:

**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 first generation 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 4.4 – API Level 19*.
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*.

Creating an Android Virtual Device (AVD)

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.

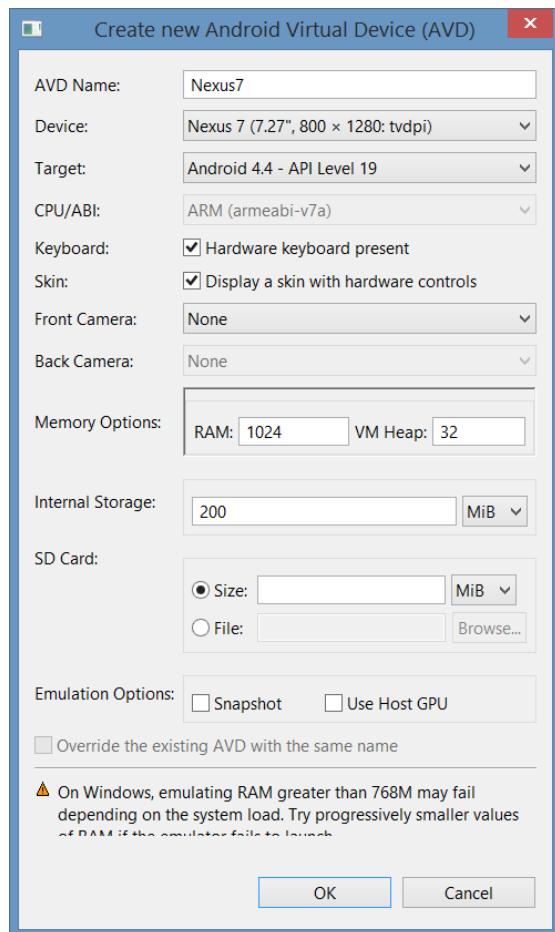


Figure 3-3

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-19"  
    Name: Android 4.4  
    Type: Platform  
    API level: 19  
    Revision: 1  
    Skins: HVGA, QVGA, WQVGA400, WQVGA432, WSVGA, WVGA800 (default),  
    WVGA854, WXGA720, WXGA800, WXGA800-7in  
    ABIs : armeabi-v7a
```

Creating an Android Virtual Device (AVD)

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 4.4 API level 19 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 a basic Android 4.4 device, also providing the option to create a custom configuration to match the specification of a specific device if required. 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 4.4 (API level 19)
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

Creating an Android Virtual Device (AVD)

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.

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*.

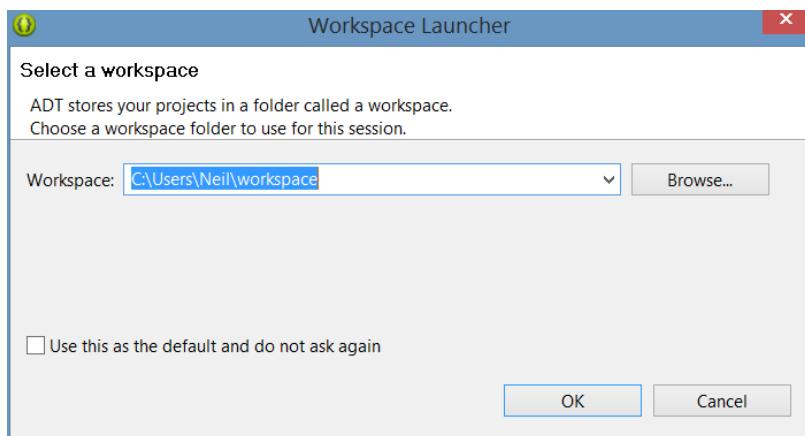


Figure 4-1

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 19: Android 4.4 (Kit Kat)*. Once these settings have been configured, the dialog should match that shown in Figure 4-2:

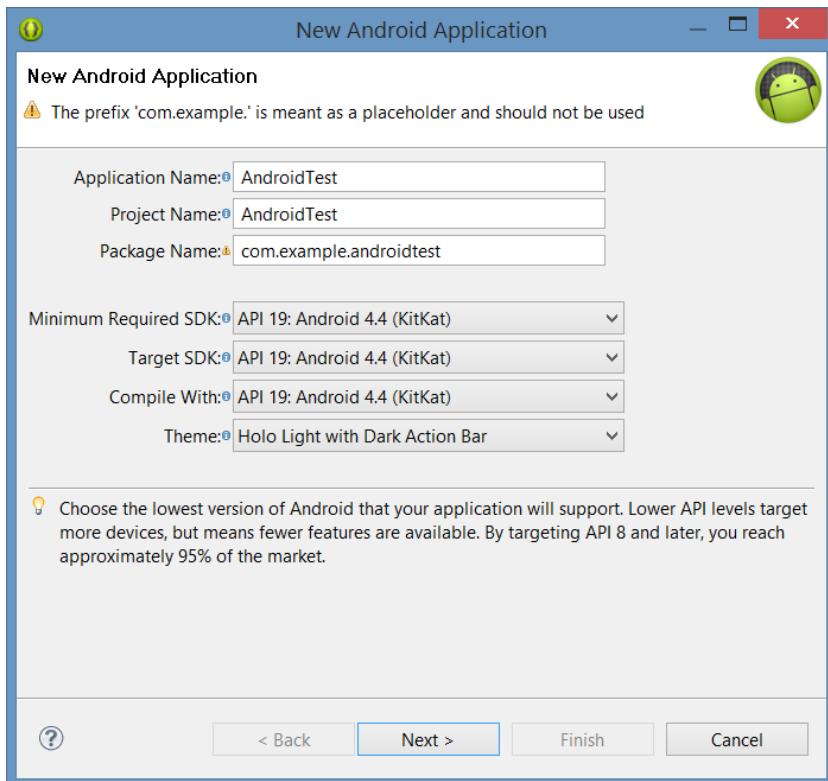


Figure 4-2

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.

The custom launcher selection, on the other hand, will provide the option to specify the icon that will represent the application on the device screen.

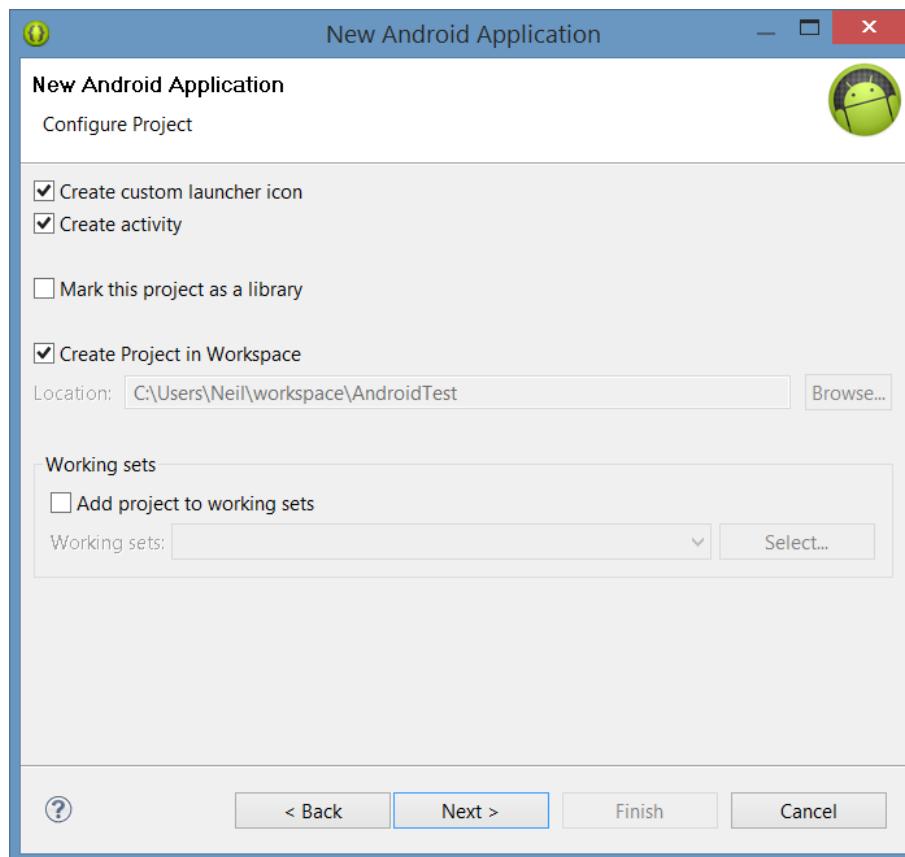


Figure 4-3

4.4 Configuring the Launcher Icon

Clicking the *Next >* button will proceed to the launcher icon configuration screen. Before the application can be submitted to the Google Play app store for sale it will need to have an icon associated with it. This icon is displayed on the screen of the Android device and is touched by the user to launch the application. The launch icon can take the form of a set of PNG image files, clipart or even text. Options are also provided on this screen to configure the background color of the launcher and to change the shape surrounding the icon.

When assigning image files for the launcher icon, images for a variety of screen resolutions and densities may be specified. If only a single image size is provided, the Android system will scale the image for different screens, potentially leading to some image quality degradation. For the purposes of this example, however, it is adequate to use the default icon images:

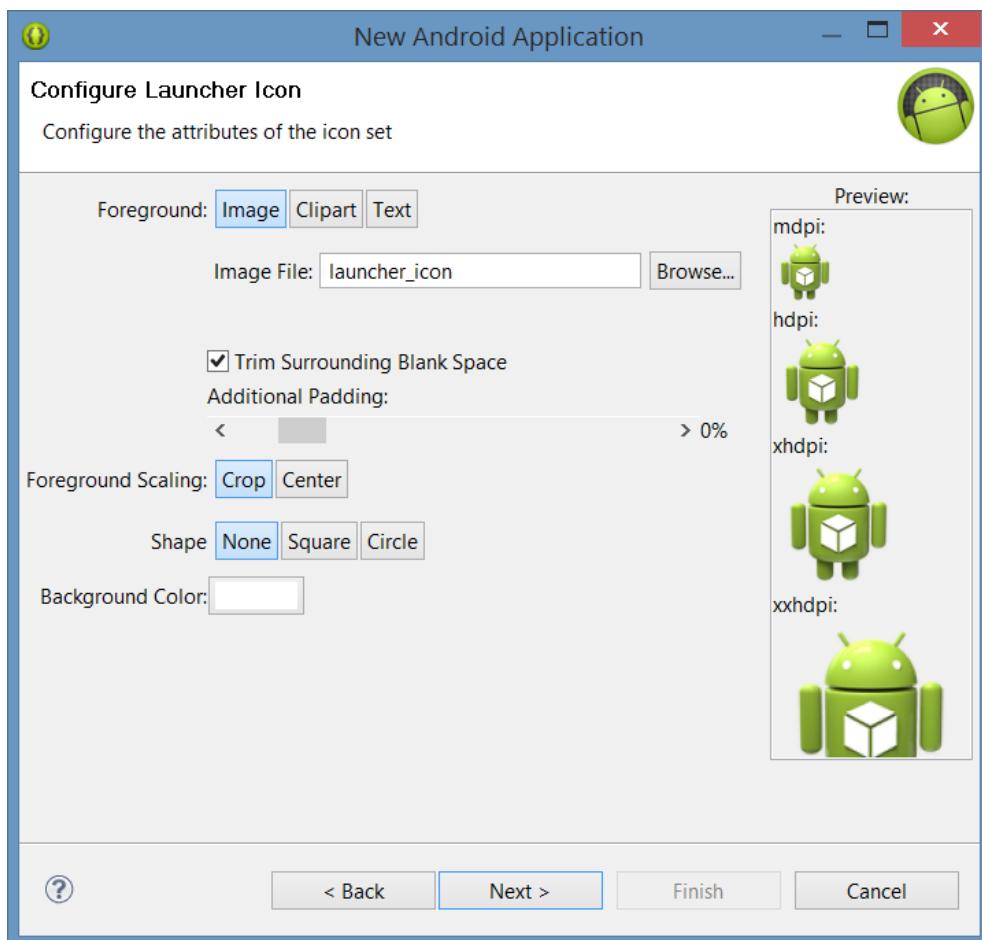


Figure 4-4

Click *Next >* to proceed with the configuration process.

4.5 Creating an Activity

The next step is to define the type of initial activity that is to be created for the application. A range of different activity types is available when developing Android applications. The *Master/Detail Flow* option will be covered in a later chapter. For the purposes of this example, however, simply select the option to create a *BlankActivity* before clicking *Next >*.

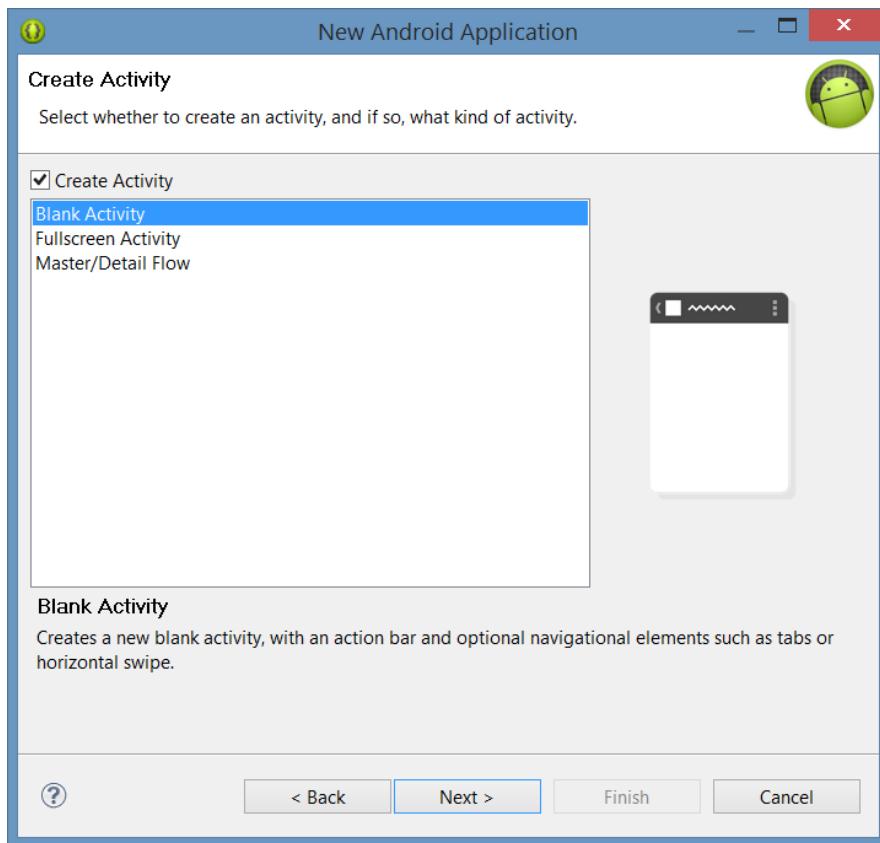


Figure 4-5

With the Blank Activity option selected, click *Next >*. On the final screen (Figure 4-6) name the activity *AndroidTestActivity*. The activity will consist of a single user interface screen layout which, for the purposes of this example, should be named *activity_android_test*. Finally, since this is a very simple, single screen activity, there is no need to select a navigation type, so leave this menu set to *None*.

Finally, click on *Finish* to initiate the project creation process.

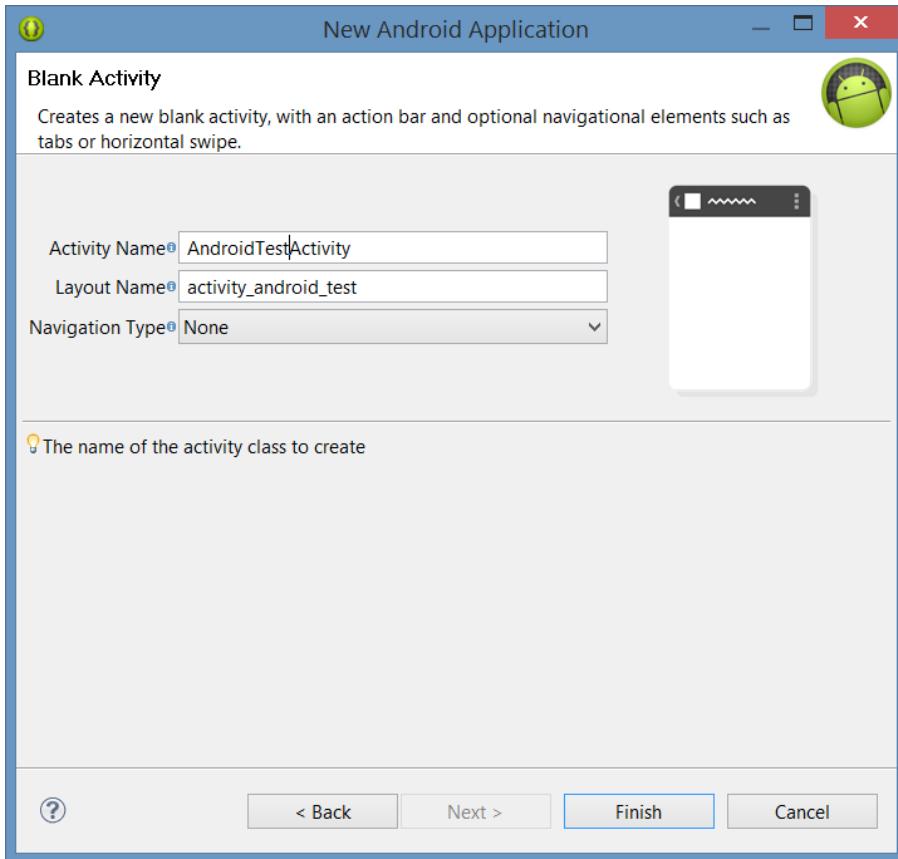


Figure 4-6

4.6 Running the Application in the AVD

At this point, Eclipse has created a minimal example application project and opened the main workbench screen. If the “Welcome!” panel is still displayed, close it by clicking on the “X” in the “Android IDE” tab.

The newly created project and references to associated files are listed in the *Package Explorer* located in a panel on the left hand side of the main Eclipse window. Clicking on the right facing arrow next to the *AndroidTest* project name will unfold the project and list the various files and sub-folders contained therein. This essentially mirrors the directory hierarchy and files located in the project’s workspace folder on the local file system of the development computer:

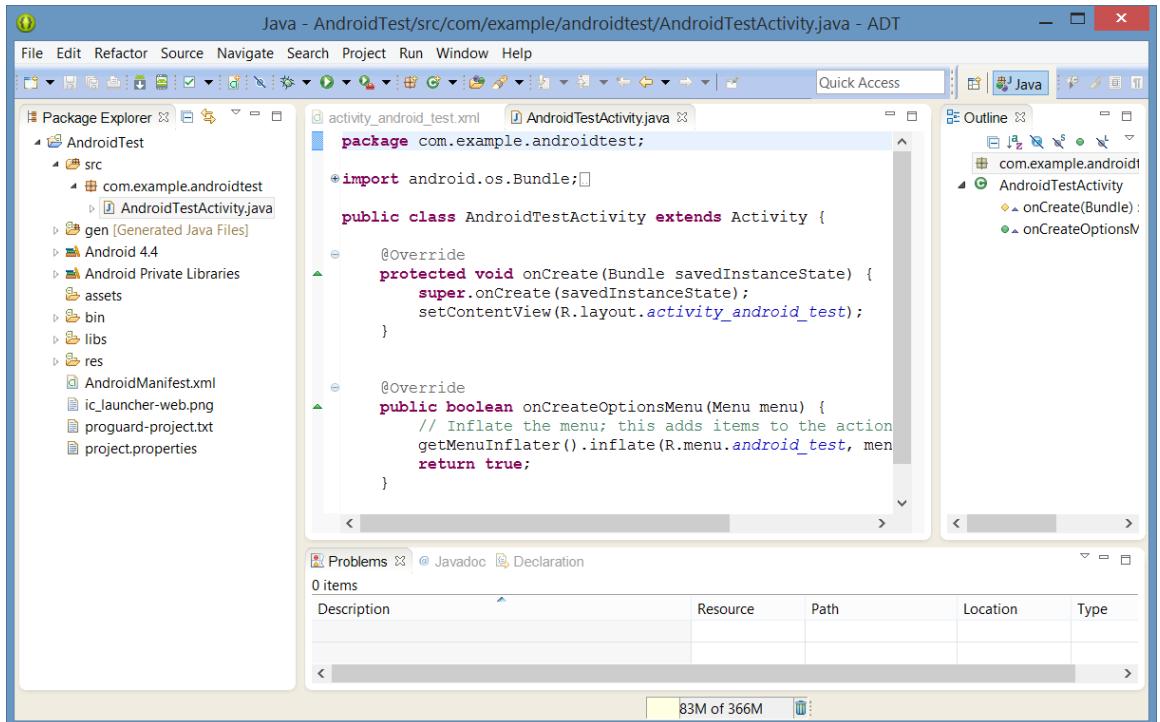


Figure 4-7

The example project created for us when we selected the option to create an activity consists of a user interface containing a label that will read “Hello World” when the application is executed. In order to run the application in the AVD that was created in the chapter entitled *Creating an Android Virtual Device (AVD)*, simply right-click on the application name in the Package Explorer and select *Run As -> Android Application* from the resulting menu. If, at this point, more than one AVD emulator has been configured on the system, a window will appear providing the option to select which AVD environment the application will run in. If multiple AVDs are listed, select the *Nexus7* emulator created in the earlier chapter. In the event that only one AVD is available, Eclipse will automatically launch that virtual device. The AVD window typically appears immediately, but a delay may be encountered as the emulator starts up and loads the Android operating system. Once the operating system has loaded, the example application will run and appear within the emulator as shown in Figure 4-8:

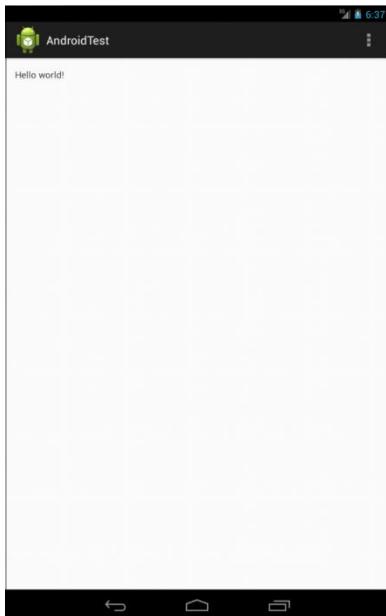


Figure 4-8

In the event that the activity does not automatically launch, check to see if the launch icon has appeared on the emulator screen. If it has, simply click on it to launch the application.

When the application has launched, an additional window may appear asking whether or not LogCat messages should be monitored. When an application is running, a range of diagnostic messages is output by the system. In addition, the application developer may have included diagnostic messages into the application code. It is generally recommended, therefore, that monitoring of these messages be enabled.

Assuming that the application loads into the emulator and runs as expected, we have safely verified that the Android development environment is correctly installed and configured.

4.7 Stopping a Running Application

When building and running an application for testing purposes, each time a new revision of the application is compiled and run, the previous instance of the application running on the device or emulator will be terminated automatically and replaced with the new version. It is also possible to manually stop a running application from within the Eclipse environment.

To stop a running application, begin by displaying the Eclipse DDMS perspective (DDMS stands for Dalvik Debug Monitor Server). The default configuration for Eclipse is to launch showing the Java perspective and for a button to be located in the top right hand corner of the main Eclipse screen (Figure 4-9) that allows the DDMS perspective to be displayed.

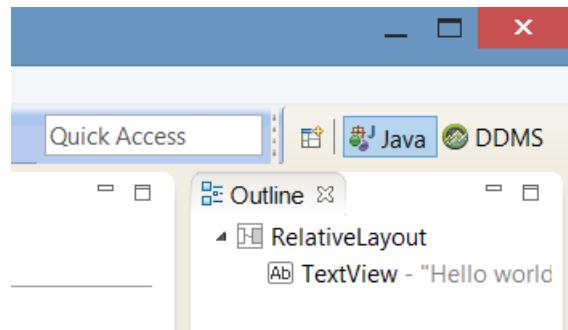


Figure 4-9

In the event that this button is not present, the perspective may be displayed using the *Window -> Open Perspective -> DDMS* menu option. Once selected, the DDMS perspective will appear as illustrated in (Figure 4-10).

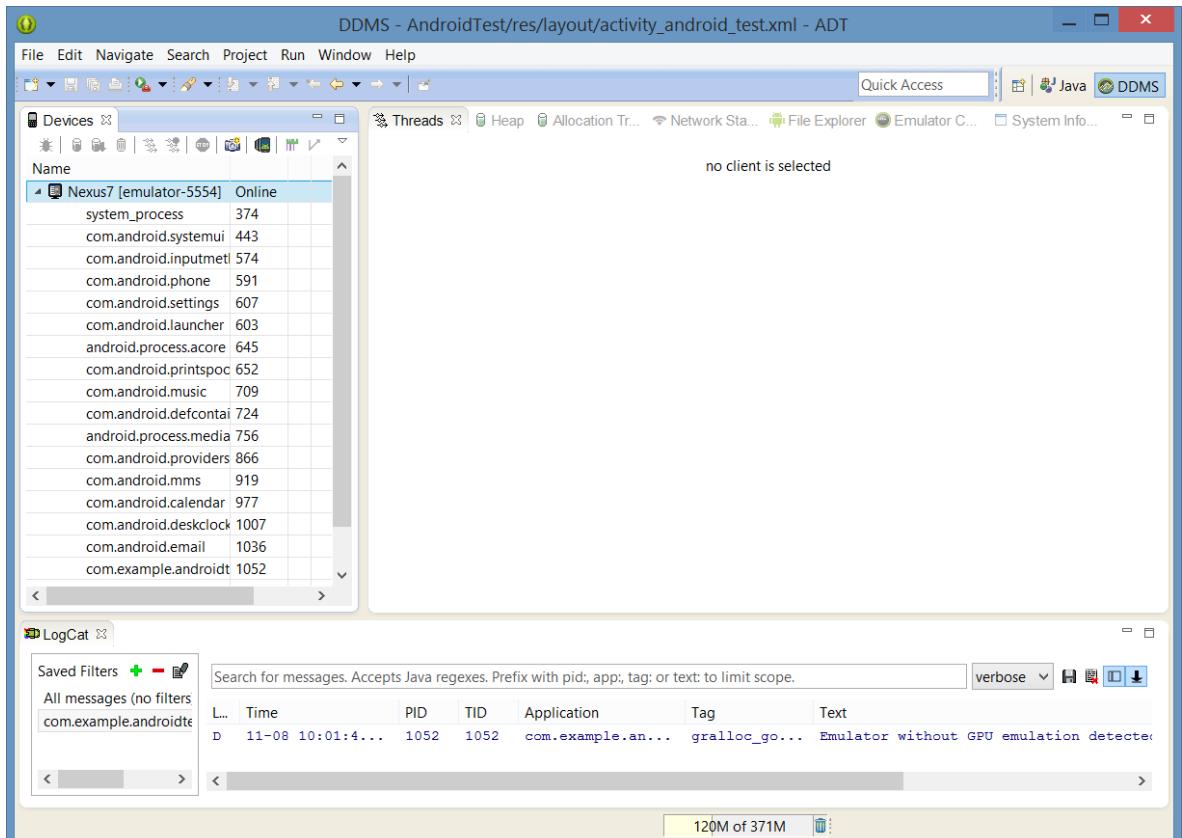


Figure 4-10

Creating an Example Android Application

The left hand panel, entitled Devices, lists any devices or emulators to which the development environment is attached. Under each device is a list of processes currently running on that device or emulator instance. Figure 4-11, for example, shows the Devices panel with the AndroidTest application running:

Name	Online	ID
Nexus7 [emulator-5554]	Online	Nexus7 [4....
system_process	374	8600
com.android.systemui	443	8601
com.android.inputmethod.l	574	8609
com.android.phone	591	8612
com.android.settings	607	8614
com.android.launcher	603	8615
android.process.acore	645	8617
com.android.printspooler	652	8619
com.android.music	709	8621
com.android.defcontainer	724	8623
android.process.media	756	8626
com.android.providers.calen	866	8628
com.android.mms	919	8629
com.android.calendar	977	8631
com.android.deskclock	1007	8634
com.android.email	1036	8636
com.example.androidtest	1052	8638 / 87...
com.svox.pico	1100	8639
com.android.exchange	1116	8640

Figure 4-11

To terminate the AndroidTest application, select the process from the list and click on the red Stop button located in the Devices panel toolbar.

To return to the main Java development perspective, simply click on the Java button in the Eclipse toolbar.

4.8 Modifying the Example Application

The next step in this tutorial is to modify the user interface of our application so that it displays a larger text view object with a different message to the one provided for us by Eclipse.

The user interface design for our activity is stored in a file named *activity_android_test.xml* which, in turn, is located under *res -> layout* in the project workspace. Using the Package Explorer panel, locate this file as illustrated in Figure 4-12:

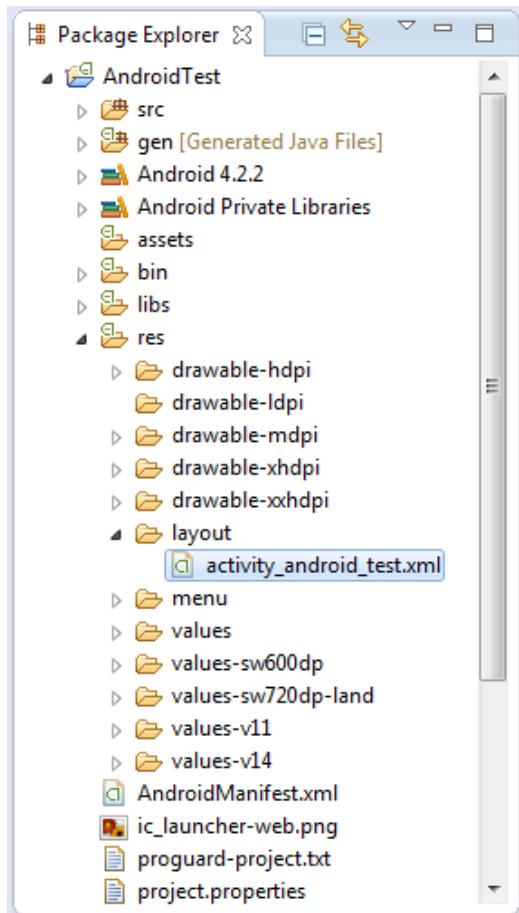


Figure 4-12

Once located, double click on the file to load it into the user interface builder tool which will appear in the center panel of the Eclipse main window:

Creating an Example Android Application

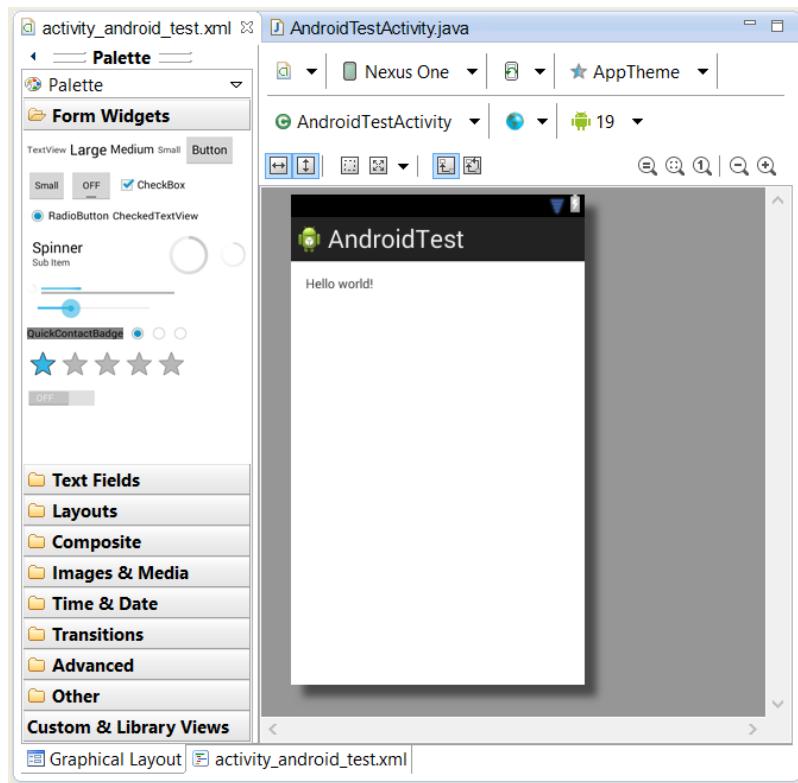


Figure 4-13

In the toolbar across the top of the layout editor panel is a menu that is currently set to *Nexus One*. Since we are designing a layout for the screen of a Nexus 7 device, click on the menu and select the *Nexus 7 (800 x 1280: tvhdpi)* menu option. The visual representation of the device screen will subsequently change to reflect the dimensions of the Nexus 7 device. If you have created a custom AVD for another device, this may also be selected from the menu so that the design canvas matches that device.

To change the orientation between landscape and portrait simply use the drop down menu immediately to the right of the device selection menu showing the icon.

In the center of the panel is the graphical representation of the user interface design, now within the context of a Nexus 7 device. As can be seen, this includes the label that displays the Hello World message. Running down the left hand side of the panel is a palette containing different categories of user interface components that may be used to construct a user interface, such as buttons, labels and text fields. It should be noted, however, that not all user interface components are obviously visible to the user. One such category consists of *layouts*. Android supports a variety of different layouts that provide different levels of control over how visual user interface components are positioned and managed on the screen. Though it is difficult to tell from looking at the visual representation of the user interface, the current

design has been created using a `RelativeLayout`. This can be confirmed by reviewing the information in the *Outline* panel that, by default, is located on the upper right hand side of the Eclipse main window and is shown in Figure 4-14:

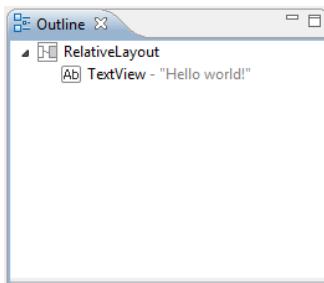


Figure 4-14

As we can see from the outline, the user interface consists of a `RelativeLayout` parent that has as a child the `TextView` object.

The first step in modifying the application is to delete the `TextView` component from the design. Begin by clicking on the `TextView` object within the user interface view so that it appears with a blue border around it. Once selected, press the Delete key on the keyboard.

From the Palette panel, select the *Form Widgets* category if it is not already selected. Click and drag the *Large TextView* object and drop it in the center of the user interface design (green marker lines will appear to indicate the center of the display):

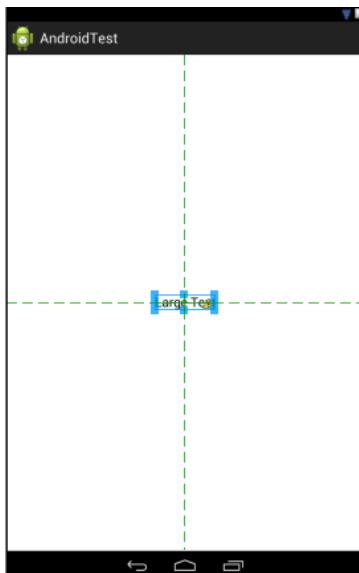


Figure 4-15

Creating an Example Android Application

Right-click over the `TextView` and select *Edit Text...* from the menu. When developing applications, attributes and values such as text strings should be stored in the form of *resources* wherever possible. Doing so enables changes to the appearance of the application to be made by modifying resource files instead of changing the application source code. This can be especially valuable when translating a user interface to a different spoken language. If all of the text in a user interface is contained in a single resource file, for example, that file can be given to a translator who will then perform the translations and return the translated file for inclusion in the application. This enables multiple languages to be targeted without the necessity for any source code changes to be made. In this instance, we are going to create a new resource named `welcomestring` and assign the string “Welcome to Android” to it. In the *Resource Chooser* dialog that is currently displayed, click on the *New String...* button and in the resulting *Create New Android String* dialog enter “Welcome to Android” into the *String* field and `welcomestring` into the *New R.string* field before clicking on *OK*. On returning to the Resource Chooser, make sure `welcomestring` is selected before clicking on *OK*.

Once changes have been made to a file within Eclipse, it is important to remember to save the changes before moving on to other tasks. This can be achieved by selecting the *File -> Save* menu option, or by using the *Ctrl-S* keyboard shortcut. Eclipse also allows multiple files to be open for editing simultaneously. Each open file is represented by a tab along the top edge of the editing panel. To close an open file, simply click on the X next to the file name in the corresponding tab.

When there is insufficient space to display a tab for each open file, a `>>` symbol appears to the far right of the tab bar together with a number indicating the number of open files beyond those currently visible. Clicking on this will display a dropdown list of all open files. Figure 4-16, for example, shows tabs for three currently open files together with an indication that another seven files are open but not visible. The drop down menu shows the names of all ten open files.

Double clicking on a tab will cause that editing session to expand to fill the entire Eclipse window. Double clicking a second time reverts the panel to its original size. Clicking and dragging a tab outside the Eclipse window results in the editing session for the corresponding file appearing in an entirely separate window on the desktop.

Editing panels may be displayed side by side in a tiled arrangement by clicking and dragging a tab to a location to the right or left of, or above or below an existing editing panel. As the dragging motion approaches different locations, guidelines will appear indicating whether the editing panels will be tiled vertically or horizontally.

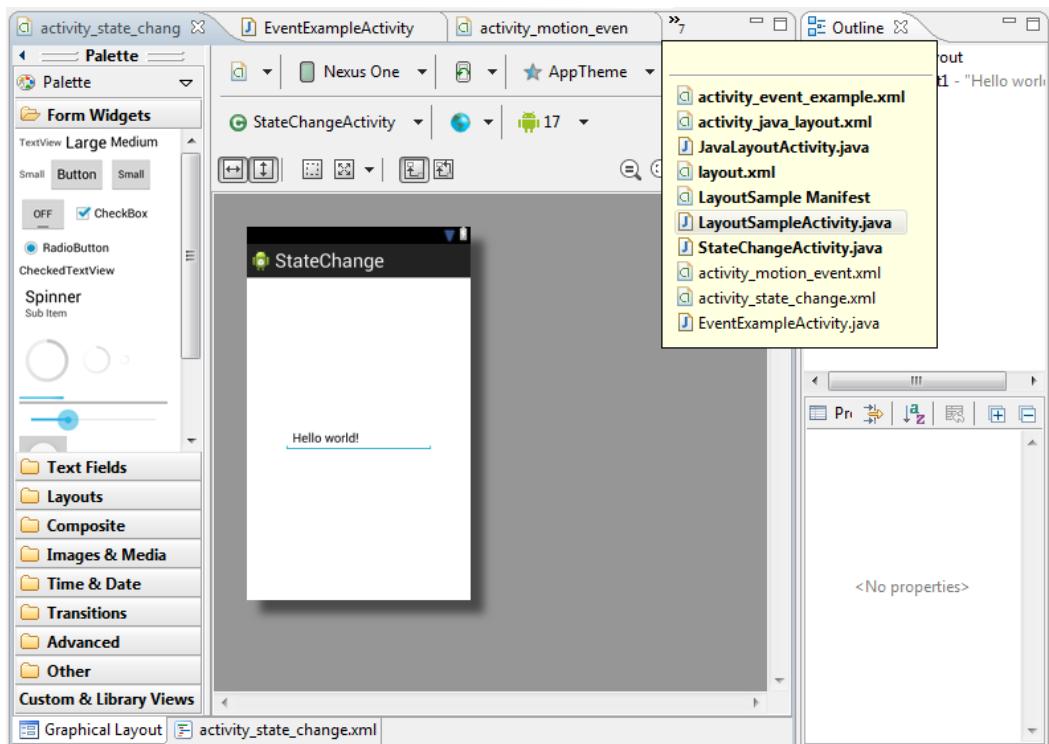


Figure 4-16

The design is now complete so once again run the application in the emulator environment. This time the larger TextView will appear in the center of the display containing the new string resource value.

4.9 Reviewing the Layout and Resource Files

Before moving on to the next chapter, we are going to look at some of the internal aspects of user interface designs and resource handling. In the previous section, we made some changes to the user interface by modifying the *activity_android_test.xml* file using the Graphical Layout tool. In fact, all that the Graphical Layout was doing was providing a user-friendly way to edit the underlying XML content of the file. In practice, there is no reason why you cannot modify the XML directly in order to make user interface changes, and in some instances, this will actually be quicker than using the graphical layout tool. At the bottom of the Graphical Layout panel are two tabs labeled *Graphical Layout* and *activity_android_test.xml* respectively. To switch to the XML view simply select the *activity_android_test.xml* tab as shown in Figure 4-17:

Creating an Example Android Application

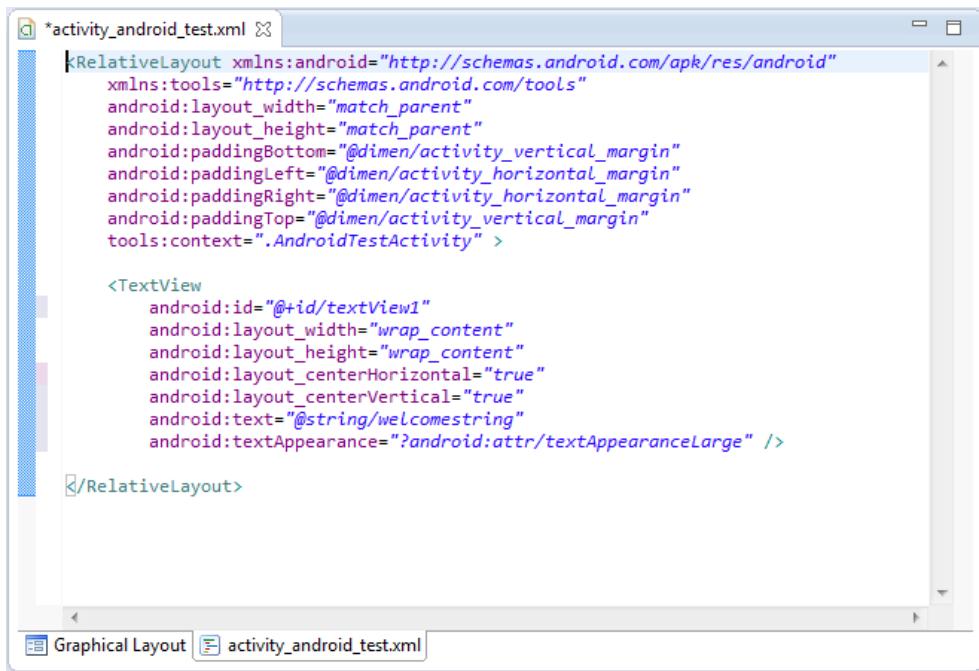


Figure 4-17

As can be seen from the structure of the XML file, the user interface consists of the `RelativeLayout` component, which in turn, is the parent of the `TextView` object. We can also see that the `text` property of the `TextView` is set to our `welcomestring` resource. Although varying in complexity and content, all user interface layouts are structured in this hierarchical, XML based way.

Finally, use the Package Explorer to locate the `res -> values -> strings.xml` file and double click on it to load it into the editor. Tabs at the bottom of the editor pane provide options to use the resource editor (*Resources*) or to view the raw XML content of the file (*strings.xml*). Currently the XML should read as follows:

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

    <string name="app_name">AndroidTest</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="welcomestring">Welcome to Android</string>

</resources>
```

As a demonstration of resources in action, change the string value currently assigned to the *welcomestring* resource then run the application again. Note that the application has picked up the new resource value for the welcome string.

4.10 Summary

Whilst not excessively complex, a number of steps are involved in setting up an Android development environment. Having performed those steps, it is worth working through a simple example to make sure the environment is correctly installed and configured. In this chapter, we have created an application and then run it within an AVD emulation session. The Eclipse Graphical Layout tool was then used to modify the user interface of the application. In so doing, we explored the importance of using resources wherever possible, particularly in the case of string values, and briefly touched on the topic of layouts. Finally, we looked at the underlying XML that is used to store the user interface designs of Android applications.

Now that we have looked at running applications within an AVD emulator environment, the next chapter will cover *Testing Android Applications on a Physical Android Device with ADB*.

5. Testing Android Applications on a Physical Android Device with ADB

Whilst much can be achieved by testing applications using an Android Virtual Device (AVD), there is no substitute for performing real world application testing on a physical Android device and there are a number of features of Android that are only available on physical Android devices.

Communication with both AVD instances and connected Android devices is handled by the *Android Debug Bridge (ADB)*. In this chapter we will work through the steps to configure the adb environment to enable application testing on a physical Android device with Mac OS X, Windows and Linux based systems.

5.1 An Overview of the Android Debug Bridge (ADB)

The primary purpose of the ADB is to facilitate interaction between the development system and both AVD emulators and physical Android devices for the purposes of running and debugging applications.

The ADB consists of a client, a server process running in the background on the development system and a daemon background process running in either AVDs or physical Android devices such as phones and tablets.

The ADB client can take a variety of forms. For example, a client is provided in the form of a command-line tool named *adb* located in the Android SDK *platform-tools* sub-directory. Similarly, the Eclipse ADT Plugin also has a built-in client.

A variety of tasks may be performed using the *adb* command-line tool. For example, a listing of currently active virtual or physical devices may be obtained using the *devices* command-line argument. The following command output indicates the presence of an AVD on the system but no physical devices:

```
$ adb devices
List of devices attached
emulator-5554    device
```

5.2 Enabling ADB on Android 4.4 based Devices

Before ADB can connect to an Android device, that device must first be configured to allow the connection. On phone and tablet devices running Android 4.4 or later, the steps to achieve this are as follows:

1. Open the Settings app on the device and select the *About tablet* or *About phone* option.
2. On the *About* screen, scroll down to the *Build number* field (Figure 5-1) and tap on it seven times until a message appears indicating that developer mode has been enabled.

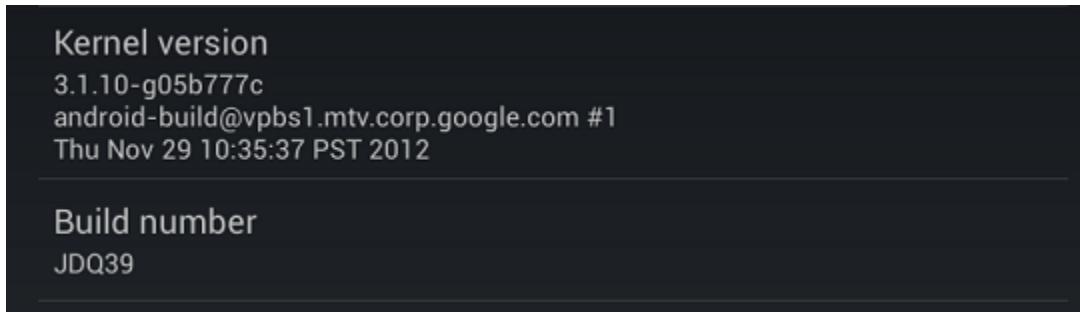


Figure 5-1

3. Return to the main Settings screen and note the appearance of a new option titled *Developer options*. Select this option and locate the setting on the developer screen entitled *USB debugging*. Enable the checkbox next to this item as illustrated in Figure 5-2 to enable the adb debugging connection.

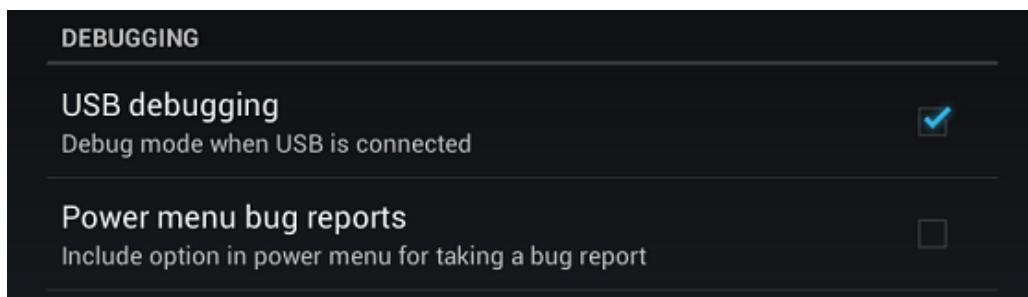


Figure 5-2

4. Swipe downward from the top of the screen to display the notifications panel (Figure 5-3) and note that the device is currently connected as a *media device*.

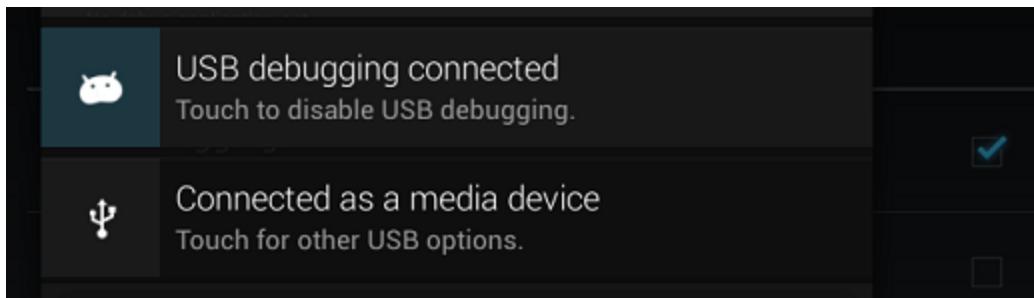


Figure 5-3

5. Select the media device notification entry and on the resulting panel (Figure 5-4) change the connection type to *Camera (PTP)*.

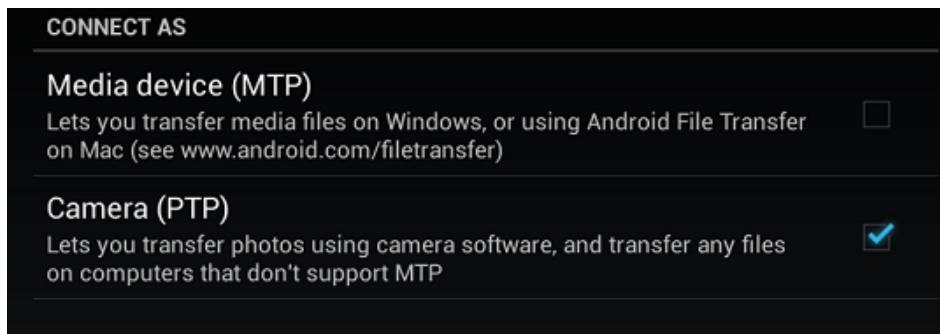


Figure 5-4

At this point, the device is now configured to accept debugging connections from adb on the development system. All that remains is to configure the development system to detect the device when it is attached. Whilst this is a relatively straightforward process, the steps involved differ depending on whether the development system is running Windows, Mac OS X or Linux. Note that the following steps assume that the Android SDK *platform-tools* directory is included in the operating system PATH environment variable as described in the chapter entitled *Setting up an Android Development Environment*.

5.2.1 Mac OS X ADB Configuration

In order to configure the ADB environment on a Mac OS X system, connect the device to the computer system using a USB cable, open a terminal window and execute the following command:

```
android update adb
```

Next, restart the adb server by issuing the following commands in the terminal window:

```
$ adb kill-server
```

Testing Android Applications on a Physical Android Device with ADB

```
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
```

Once the server is successfully running, execute the following command to verify that the device has been detected:

```
$ adb devices
List of devices attached
74CE000600000001      offline
```

If the device is listed as *offline*, go to the Android device and check for the presence of the dialog shown in Figure 5-9 seeking permission to *Allow USB debugging*. Enable the checkbox next to the option that reads *Always allow from this computer*, before clicking on *OK*. Repeating the *adb devices* command should now list the device as being available:

```
List of devices attached
015d41d4454bf80c      device
```

In the event that the device is not listed, try logging out and then back in to the Mac OS X desktop and, if the problem persists, rebooting the system.

5.2.2 Windows ADB Configuration

The first step in configuring a Windows based development system to connect to an Android device using ADB is to install the appropriate USB drivers on the system. In the case of some devices, the *Google USB Driver* must be installed (a full listing of devices supported by the Google USB driver can be found online at <http://developer.android.com/sdk/win-usb.html>).

To install this driver, launch Eclipse and perform the following steps:

1. Launch Eclipse and open the Android SDK Manager (*Window -> Android SDK Manager*).
2. Scroll down to the *Extras* section and check the status of the *Google USB Driver* package to make sure that it is listed as *Installed*.
3. If the driver is not installed, select it and click on the *Install packages* button to initiate the installation.
4. Once installation is complete, close the Android SDK Manager.

For Android devices not supported by the Google USB driver, it will be necessary to download the drivers provided by the device manufacturer. A listing of drivers and download information can be obtained online at <http://developer.android.com/tools/extras/oem-usb.html>.

When an Android device is attached to a Windows system it is configured as a *Portable Device*. In order for the device to connect to ADB it must be configured as an *Android ADB Composite Device*.

First, connect the Android device to the computer system if it is not currently connected. Next, display the Control Panel and select Device Manager. In the resulting dialog, check for a category entitled *Other Devices*. Unfold this category and check to see if the Android device is listed (in the case of Figure 5-5, a Nexus 7 has been detected):

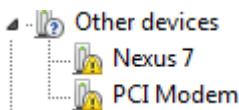


Figure 5-5

Right-click on the device name and select *Update Driver Software* from the menu. Select the option to *Browse my computer for driver software* and in the next dialog, keep the *Include subfolder* option selected and click on the *Browse...* button. Navigate to the location into which the USB drivers were installed (in the case of the Google USB driver, this will be <sdk path>\sdk\extras\google\usb_driver) and click on *OK* to select the driver folder followed by *Next* to initiate the installation.

During the installation, a Windows Security prompt will appear seeking permission to install the driver as illustrated in Figure 5-6. When this dialog appears, click on the *Install* button to proceed.

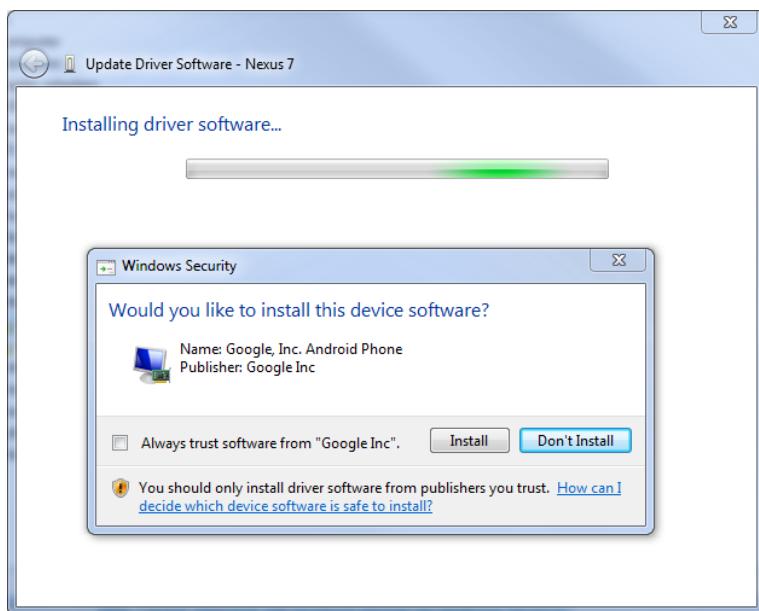


Figure 5-6

Once the installation completes, the Windows driver update screen will refresh to display a message indicating that the driver has been installed and that the device is now recognized as an Android Composite ADB Interface:

Testing Android Applications on a Physical Android Device with ADB

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:



Figure 5-7

Return to the Device Manager and note that the device is no longer listed under *Other Devices* and is now categorized as an Android Composite ADB Interface. Figure 5-8, for example, shows the device entry for a Nexus 7 tablet using the Google USB driver.

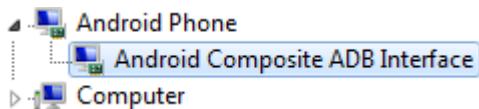


Figure 5-8

With the drivers installed and the device now being recognized as the correct device type, open a Command Prompt window and execute the following command:

```
adb devices
```

This command should output information about the connected device similar to the following:

```
List of devices attached  
015d41d4454bf80c      offline
```

If the device is listed as *offline* or *unauthorized*, go to the device display and check for the dialog shown in Figure 5-9 seeking permission to *Allow USB debugging*.

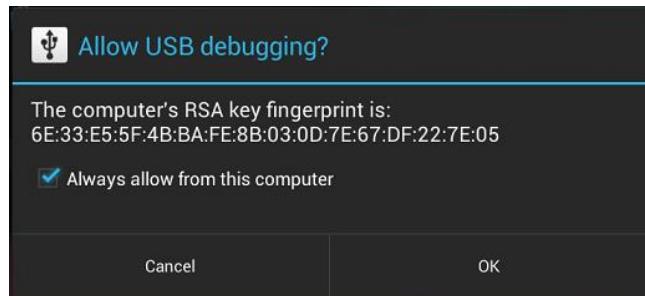


Figure 5-9

Enable the checkbox next to the option that reads *Always allow from this computer*, before clicking on *OK*. Repeating the *adb devices* command should now list the device as being ready:

```
List of devices attached  
015d41d4454bf80c      device
```

In the event that the device is not listed, execute the following commands to restart the ADB server:

```
adb kill-server  
adb start-server
```

In the event that the device is still not listed, try executing the following command:

```
android update adb
```

Note that it may also be necessary to reboot the system.

5.2.3 Linux adb Configuration

For the purposes of this chapter, we will once again use Ubuntu Linux as a reference example in terms of configuring adb on Linux to connect to a physical Android device for application testing.

Begin by attaching the Android device to a USB port on the Ubuntu Linux system. Once connected, open a Terminal window and execute the Linux *lsusb* command to list currently available USB devices:

```
~$ lsusb  
Bus 001 Device 003: ID 18d1:4e44 asus Nexus 7 [9999]  
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Each USB devices detected on the system will be listed along with a vendor ID and product ID. A list of vendor IDs can be found online at <http://developer.android.com/tools/device.html#VendorIDs>. The above output shows that a Google Nexus 7 device has been detected. Make a note of the vendor and product ID numbers displayed for your particular device (in the above case these are 18D1 and 4E44 respectively).

Use the *sudo* command to edit the *51-android.rules* file located in the */etc/udev/rules.d* directory. For example:

```
sudo gedit /etc/udev/rules.d/51-android.rules
```

Within the editor, add the appropriate entry for the Android device, replacing <vendor_id> and <product_id> with the vendor and product IDs returned previously by the *lsusb* command:

Testing Android Applications on a Physical Android Device with ADB

```
SUBSYSTEM=="usb", ATTR{idVendor}=="<vendor_id>",  
ATTRS{idProduct}=="<product_id>", MODE="0660", OWNER="root",  
GROUP="androidadb", SYMLINK+="android%n"
```

Once the entry has been added, save the file and exit from the editor.

Next, use an editor to modify (or create if it does not yet exist) the adb_usb.ini file:

```
gedit ~/.android/adb_usb.ini
```

Once the file is loaded into the editor, add the following lines (once again replacing <vendor_id> and <product_id> with the vendor and product IDs returned previously by the *lsusb* command) before saving the file and exiting:

```
0x<vendor_id>  
0x<product_id>
```

Using the above syntax, the entries for the Nexus 7 would, for example, read:

```
0x18d1  
0x4e44
```

The final task is to create the *androidadb* user group and add your user account to it. To achieve this, execute the following commands making sure to replace <user name> with your Ubuntu user account name:

```
sudo addgroup --system androidadb  
sudo adduser <username> androidadb
```

Once the above changes have been made, reboot the Ubuntu system. Once the system has restarted, open a Terminal window, start the adb server and check the list of attached devices:

```
$ adb start-server  
* daemon not running. starting it now on port 5037 *  
* daemon started successfully *  
$ adb devices  
List of devices attached  
015d41d4454bf80c      offline
```

If the device is listed as *offline*, go to the Android device and check for the dialog shown in Figure 5-9 seeking permission to *Allow USB debugging*.

5.3 Testing the adb Connection

Assuming that the adb configuration has been successful on your chosen development platform, the next step is to try running the test application created in the chapter entitled *Creating an Example Android Application* on the device itself.

Launch Eclipse, right-click on the *AndroidTest* entry in the left hand panel and select *Run As -> Android Application*. If the Android device is connected and detected by adb, the application should load and run on the device instead of within the emulator. Output on the console panel should read as follows as the application is loaded. If the console panel is not present in the Eclipse IDE window, it can be displayed by selecting the *Window -> Show View -> Console* menu option:

```
[2013-06-11 15:24:40 - AndroidTest] Performing
com.example.AndroidTest.AndroidTestActivity activity launch
[2013-06-11 15:24:43 - ] Uploading AndroidTest.apk onto device
'015d41d4454bf80c'
[2013-06-11 15:24:43 - AndroidTest] Installing AndroidTest.apk...
[2013-06-11 15:24:45 - AndroidTest] Success!
[2013-06-11 15:24:45 - AndroidTest] Starting activity
com.example.AndroidTest.AndroidTestActivity on device 015d41d4454bf80c
[2013-06-11 15:24:45 - AndroidTest] ActivityManager: Starting: Intent {
act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
cmp=com.example.AndroidTest/.AndroidTestActivity }
```

Note that the Console panel has a number of modes consisting of Android, DDMS and OpenGL Trace View. If no output is displayed in the console it may be that it is not currently in Android mode. To change the mode, click on the console toolbar button as illustrated in Figure 5-10 and select *Android* from the drop down menu.



Figure 5-10

5.4 Manual Selection of the Application Run Target

By default, Eclipse and the Android ADT Plug-in are configured to automatically choose the run target for application testing. This means that when a physical Android device is attached to the system and visible to adb, the application will run on the device. In the event that the device is not connected, however, the default is to run within the emulator.

It is possible, however, to configure the environment to prompt you for the target each time the application is launched from within Eclipse. To configure this, right-click on the application name (in this

Testing Android Applications on a Physical Android Device with ADB

case *AndroidTest*) in the right hand panel and select the *Run As -> Run Configurations...* menu option. Within the Run Configurations dialog, select the *Target* tab and change the *Deployment Target Selection Mode* to *Always prompt to pick device* as illustrated in Figure 5-11:

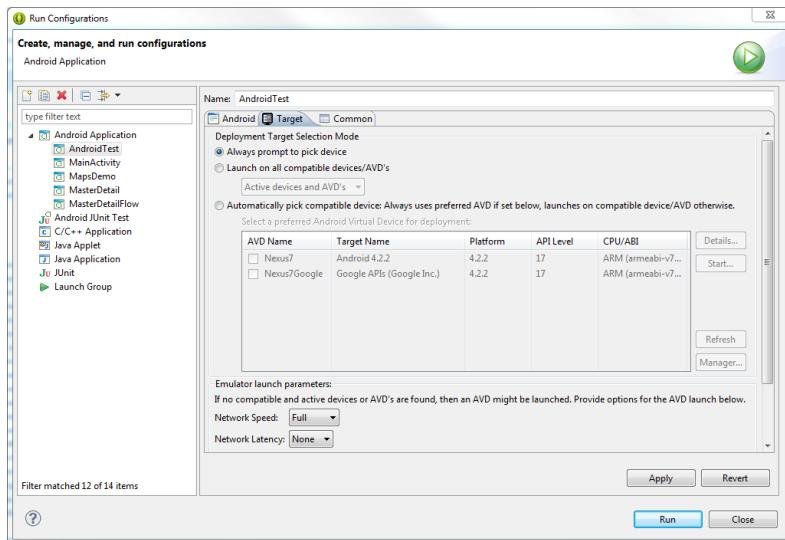


Figure 5-11

Click on the *Apply* button and then close the dialog. On future application launches, the *Android Device Chooser* window shown in Figure 5-12 will appear providing the choice between running within the emulator or on a connected physical Android device:

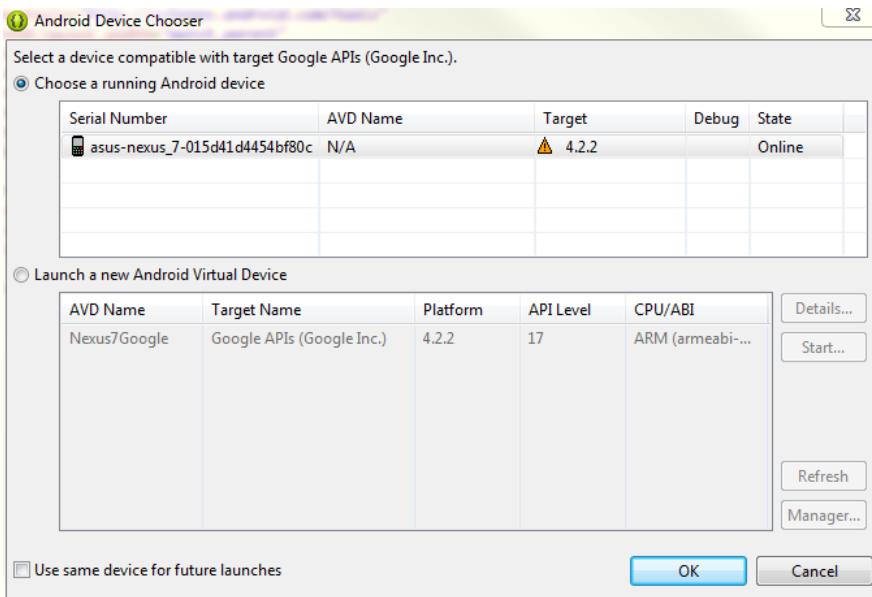


Figure 5-12

5.5 Summary

Whilst the Android Virtual Device emulator provides an excellent testing environment, it is important to keep in mind that there is no real substitute for making sure an application functions correctly on a physical Android device. This, after all, is where the application will be used in the real world.

By default, however, the Eclipse and Android ADT environments are not configured to detect Android devices as a target testing device. It is necessary, therefore, to perform some steps in order to be able to load applications directly onto an Android device from within the Android development environment. The exact steps to achieve this goal differ depending on the development platform being used. In this chapter, we have covered those steps for Linux, Mac OS X and Windows based platforms.

6. An Overview of the Android Architecture

So far in this book, steps have been taken to set up an environment suitable for the development of Android applications. An initial step has also been taken into the process of application development through the creation of a simple application.

Before delving further into the practical matters of Android application development, however, it is important to gain an understanding of some of the more abstract concepts of both the Android SDK and Android development in general. Gaining a clear understanding of these concepts now will provide a sound foundation on which to build further knowledge.

Starting with an overview of the Android architecture in this chapter, and continuing in the next few chapters of this book, the goal is to provide a detailed overview of the fundamentals of Android development.

6.1 The Android Software Stack

Android is architected in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries. This architecture can, perhaps, best be represented visually as outlined in Figure 6-1. Each layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.