

Android Studio Development Essentials

Android Studio Development Essentials – First Edition

ISBN-13: 978-0-9860273-7-6

© 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 1.0



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 Studio Development Environment	3
2.1 System Requirements	3
2.2 Installing the Java Development Kit (JDK).....	3
2.2.1 Windows JDK Installation	4
2.2.2 Mac OS X JDK Installation	4
2.3 Linux JDK Installation	5
2.4 Downloading the Android Studio Package	6
2.5 Installing Android Studio.....	7
2.5.1 Installation on Windows	7
2.5.2 Installation on Mac OS X	7
2.5.3 Installation on Linux	8
2.6 Installing the Latest Android SDK Packages	9
2.7 Making the Android SDK Tools Command-line Accessible	10
2.7.1 Windows 7.....	10
2.7.2 Windows 8.1.....	11
2.7.3 Linux	12
2.7.4 Mac OS X	12
2.8 Updating the Android Studio and the SDK.....	13
2.9 Summary	13
3. Creating an Example Android App in Android Studio	15
3.1 Creating a New Android Project	15
3.2 Defining the Project and SDK Settings	16
3.3 Creating an Activity	17
3.4 Modifying the Example Application.....	19
3.5 Reviewing the Layout and Resource Files	26
3.6 Previewing the Layout	28
3.7 Summary	29
4. A Tour of the Android Studio User Interface	31
4.1 The Welcome Screen	31
4.2 The Main Window.....	32
4.3 The Tool Windows	33

4.4 Android Studio Keyboard Shortcuts	38
4.5 Switcher and Recent Files Navigation.....	38
4.6 Changing the Android Studio Theme.....	39
4.7 Summary.....	40
5. Creating an Android Virtual Device (AVD) in Android Studio	41
5.1 About Android Virtual Devices	41
5.2 Creating a New AVD	42
5.3 Starting the Emulator	45
5.4 Running the Application in the AVD	45
5.5 Run/Debug Configurations	47
5.6 Stopping a Running Application.....	48
5.7 AVD Command-line Creation.....	51
5.8 Android Virtual Device Configuration Files.....	53
5.9 Moving and Renaming an Android Virtual Device.....	53
5.10 Summary.....	54
6. Testing Android Studio Apps on a Physical Android Device	55
6.1 An Overview of the Android Debug Bridge (ADB)	55
6.2 Enabling ADB on Android 4.4 based Devices	56
6.2.1 Mac OS X ADB Configuration	57
6.2.2 Windows ADB Configuration.....	58
6.2.3 Linux adb Configuration	61
6.3 Testing the adb Connection.....	63
6.4 Summary.....	64
7. The Basics of the Android Studio Code Editor	65
7.1 The Android Studio Editor	65
7.2 Splitting the Editor Window	68
7.3 Code Completion	69
7.4 Statement Completion	71
7.5 Parameter Information.....	71
7.6 Code Generation.....	72
7.7 Code Folding.....	73
7.8 Quick Documentation Lookup.....	75
7.9 Code Reformatting	76
7.10 Summary.....	76
8. An Overview of the Android Architecture.....	77
8.1 The Android Software Stack	77
8.2 The Linux Kernel	78
8.3 Android Runtime - Dalvik Virtual Machine	79

8.4 Android Runtime – Core Libraries.....	79
8.4.1 Dalvik VM Specific Libraries.....	79
8.4.2 Java Interoperability Libraries	79
8.4.3 Android Libraries	80
8.4.4 C/C++ Libraries	81
8.5 Application Framework.....	81
8.6 Applications	82
8.7 Summary	82
9. The Anatomy of an Android Application	83
9.1 Android Activities.....	83
9.2 Android Intents.....	84
9.3 Broadcast Intents.....	84
9.4 Broadcast Receivers.....	84
9.5 Android Services	85
9.6 Content Providers	85
9.7 The Application Manifest.....	85
9.8 Application Resources.....	86
9.9 Application Context	86
9.10 Summary.....	86
10. Understanding Android Application and Activity Lifecycles.....	87
10.1 Android Applications and Resource Management	87
10.2 Android Process States	88
10.2.1 Foreground Process	88
10.2.2 Visible Process	89
10.2.3 Service Process	89
10.2.4 Background Process.....	89
10.2.5 Empty Process.....	89
10.3 Inter-Process Dependencies	89
10.4 The Activity Lifecycle.....	89
10.5 The Activity Stack.....	90
10.6 Activity States	91
10.7 Configuration Changes.....	91
10.8 Handling State Change.....	92
10.9 Summary.....	92
11. Handling Android Activity State Changes	93
11.1 The Activity Class	93
11.2 Dynamic State vs. Persistent State	96
11.3 The Android Activity Lifecycle Methods	97

11.4 Activity Lifetimes	98
11.5 Summary.....	99
12. Android Activity State Changes by Example.....	101
12.1 Creating the State Change Example Project	101
12.2 Designing the User Interface	103
12.3 Overriding the Activity Lifecycle Methods.....	104
12.4 Filtering the LogCat Panel.....	108
12.5 Running the Application	109
12.6 Experimenting with the Activity	110
12.7 Summary.....	111
13. Saving and Restoring the State of an Android Activity	113
13.1 Saving Dynamic State	113
13.2 Default Saving of User Interface State.....	113
13.3 The Bundle Class.....	115
13.4 Saving the State	115
13.5 Restoring the State	117
13.6 Testing the Application	118
13.7 Summary.....	118
14. Understanding Android Views, View Groups and Layouts	119
14.1 Designing for Different Android Devices	119
14.2 Views and View Groups	119
14.3 Android Layout Managers	120
14.4 The View Hierarchy.....	121
14.5 Creating User Interfaces	123
14.6 Summary.....	123
15. A Guide to the Android Studio Designer Tool	125
15.1 The Android Studio Designer	125
15.2 Design Mode.....	125
15.3 Text Mode.....	127
15.4 Setting Properties	129
15.5 Type Morphing	130
15.6 Creating a Custom Device Definition	131
15.7 Summary.....	131
16. Designing a User Interface using the Android Studio Designer Tool	133
16.1 An Android Studio Designer Tool Example	133
16.2 Creating a New Activity	133
16.3 Designing the User Interface	135

16.4 Editing View Properties.....	136
16.5 Running the Application	137
16.6 Manually Creating an XML Layout	137
16.7 Using the Hierarchy Viewer	140
16.8 Summary.....	144
17. Creating an Android User Interface in Java Code	145
17.1 Java Code vs. XML Layout Files	145
17.2 Creating Views	146
17.3 Properties and Layout Parameters	146
17.4 Creating the Example Project in Android Studio	147
17.5 Adding Views to an Activity	147
17.6 Setting View Properties	149
17.7 Adding Layout Parameters and Rules	150
17.8 Using View IDs	152
17.9 Converting Density Independent Pixels (dp) to Pixels (px)	154
17.10 Summary	156
18. Using the Android GridLayout Manager in Android Studio Designer	159
18.1 Introducing the Android GridLayout and Space Classes	159
18.2 The GridLayout Example	160
18.3 Creating the GridLayout Project	160
18.4 Creating the GridLayout Instance	160
18.5 Adding Views to GridLayout Cells	162
18.6 Moving and Deleting Rows and Columns	164
18.7 Implementing Cell Row and Column Spanning.....	165
18.8 Changing the Gravity of a GridLayout Child.....	166
18.9 Summary.....	169
19. Working with the Android GridLayout using XML Layout Resources	171
19.1 GridLayouts in XML Resource Files	171
19.2 Adding Child Views to the GridLayout	172
19.3 Declaring Cell Spanning, Gravity and Margins	174
19.4 Summary.....	176
20. An Overview and Example of Android Event Handling	177
20.1 Understanding Android Events	177
20.2 Using the android:onClick Resource	178
20.3 Event Listeners and Callback Methods	178
20.4 An Event Handling Example	179
20.5 Designing the User Interface	179
20.6 The Event Listener and Callback Method	181

20.7 Consuming Events	183
20.8 Summary.....	185
21. Android Touch and Multi-touch Event Handling	187
21.1 Intercepting Touch Events	187
21.2 The MotionEvent Object.....	188
21.3 Understanding Touch Actions.....	188
21.4 Handling Multiple Touches	188
21.5 An Example Multi-Touch Application	189
21.6 Designing the Activity User Interface	189
21.7 Implementing the Touch Event Listener.....	191
21.8 Running the Example Application.....	195
21.9 Summary.....	196
22. Detecting Common Gestures using the Android Gesture Detector Class.....	197
22.1 Implementing Common Gesture Detection	197
22.2 Creating an Example Gesture Detection Project	198
22.3 Implementing the Listener Class	199
22.4 Creating the GestureDetectorCompat Instance	202
22.5 Implementing the onTouchEvent() Method	202
22.6 Testing the Application	203
22.7 Summary.....	204
23. Implementing Custom Gesture and Pinch Recognition on Android.....	205
23.1 The Android Gesture Builder Application	205
23.2 The GestureOverlayView Class	205
23.3 Detecting Gestures	205
23.4 Identifying Specific Gestures	206
23.5 Adding SD Card Support to an AVD	206
23.6 Building and Running the Gesture Builder Application	207
23.7 Creating a Gestures File	208
23.8 Extracting the Gestures File from the SD Card	210
23.9 Creating the Example Project	211
23.10 Adding the Gestures File to the Project.....	211
23.11 Designing the User Interface	211
23.12 Loading the Gestures File	213
23.13 Registering the Event Listener	214
23.14 Implementing the onGesturePerformed Method	214
23.15 Testing the Application	216
23.16 Configuring the GestureOverlayView	216
23.17 Intercepting Gestures	217

23.18 Detecting Pinch Gestures.....	217
23.19 A Pinch Gesture Example Project.....	218
23.20 Summary.....	220
24. An Introduction to Android Fragments.....	221
24.1 What is a Fragment?	221
24.2 Creating a Fragment	222
24.3 Adding a Fragment to an Activity using the Layout XML File	223
24.4 Adding and Managing Fragments in Code	225
24.5 Handling Fragment Events.....	226
24.6 Implementing Fragment Communication.....	227
24.7 Summary.....	229
25. Using Fragments in Android Studio - An Example.....	231
25.1 About the Example Fragment Application.....	231
25.2 Creating the Example Project	231
25.3 Creating the First Fragment Layout	232
25.4 Creating the First Fragment Class	234
25.5 Creating the Second Fragment Layout	235
25.6 Adding the Fragments to the Activity.....	238
25.7 Making the Toolbar Fragment Talk to the Activity	241
25.8 Making the Activity Talk to the Text Fragment.....	245
25.9 Testing the Application	247
25.10 Summary.....	248
26. An Android Studio Master/Detail Flow Tutorial	249
26.1 The Master/Detail Flow	249
26.2 Creating a Master/Detail Flow Activity.....	251
26.3 The Anatomy of the Master/Detail Flow Template	252
26.4 Modifying the Master/Detail Flow Template	254
26.5 Changing the Content Model.....	254
26.6 Changing the Detail Pane.....	256
26.7 Modifying the WebsiteDetailFragment Class	257
26.8 Adding Manifest Permissions	259
26.9 Running the Application	260
26.10 Summary.....	260
27. Creating and Managing Overflow Menus on Android.....	261
27.1 The Overflow Menu	261
27.2 Creating an Overflow Menu.....	262
27.3 Displaying an Overflow Menu.....	263
27.4 Responding to Menu Item Selections.....	264

27.5 Creating Checkable Item Groups	264
27.6 Creating the Example Project	266
27.7 Modifying the Menu Description.....	266
27.8 Modifying the onOptionsItemSelected() Method	267
27.9 Testing the Application	269
27.10 Summary.....	269
28. Animating User Interfaces with the Android Transitions Framework.....	271
28.1 Introducing Android Transitions and Scenes	271
28.2 Using Interpolators with Transitions	272
28.3 Working with Scene Transitions	273
28.4 Custom Transitions and TransitionSets in Code	274
28.5 Custom Transitions and TransitionSets in XML	275
28.6 Working with Interpolators	277
28.7 Creating a Custom Interpolator	279
28.8 Using the beginDelayedTransition Method	280
28.9 Summary.....	280
29. An Android Transition Tutorial using beginDelayedTransition	281
29.1 Creating the Android Studio TransitionDemo Project	281
29.2 Preparing the Project Files.....	281
29.3 Implementing beginDelayedTransition Animation.....	282
29.4 Customizing the Transition	285
29.5 Summary.....	286
30. Implementing Android Scene Transitions – A Tutorial	287
30.1 An Overview of the Scene Transition Project	287
30.2 Creating the Android Studio SceneTransitions Project.....	287
30.3 Identifying and Preparing the Root Container.....	287
30.4 Designing the First Scene.....	288
30.5 Designing the Second Scene	291
30.6 Entering the First Scene	293
30.7 Loading Scene 2	294
30.8 Implementing the Transitions.....	295
30.9 Adding the Transition File.....	295
30.10 Loading and Using the Transition Set	296
30.11 Configuring Additional Transitions	297
30.12 Summary.....	298
31. An Overview of Android Intents	299
31.1 An Overview of Intents	299
31.2 Explicit Intents	300

31.3 Returning Data from an Activity	301
31.4 Implicit Intents	302
31.5 Using Intent Filters.....	303
31.6 Checking Intent Availability	304
31.7 Summary.....	304
32. Android Explicit Intents – A Worked Example	305
32.1 Creating the Explicit Intent Example Application	305
32.2 Designing the User Interface Layout for ActivityA	305
32.3 Creating the Second Activity Class.....	307
32.4 Designing the User Interface Layout for ActivityB	308
32.5 Reviewing the Application Manifest File	310
32.6 Creating the Intent.....	311
32.7 Extracting Intent Data.....	312
32.8 Launching ActivityB as a Sub-Activity	313
32.9 Returning Data from a Sub-Activity	314
32.10 Testing the Application	315
32.11 Summary.....	315
33. Android Implicit Intents – A Worked Example.....	317
33.1 Creating the Android Studio Implicit Intent Example Project.....	317
33.2 Designing the User Interface	317
33.3 Creating the Implicit Intent.....	319
33.4 Adding a Second Matching Activity	320
33.5 Adding the Web View to the UI	321
33.6 Obtaining the Intent URL	322
33.7 Modifying the MyWebView Project Manifest File.....	324
33.8 Installing the MyWebView Package on a Device	325
33.9 Testing the Application	327
33.10 Summary.....	327
34. Android Broadcast Intents and Broadcast Receivers	329
34.1 An Overview of Broadcast Intents	329
34.2 An Overview of Broadcast Receivers	330
34.3 Obtaining Results from a Broadcast	332
34.4 Sticky Broadcast Intents.....	332
34.5 The Broadcast Intent Example	333
34.6 Creating the Example Application.....	333
34.7 Creating and Sending the Broadcast Intent	333
34.8 Creating the Broadcast Receiver.....	334
34.9 Configuring a Broadcast Receiver in the Manifest File	336

34.10 Testing the Broadcast Example.....	337
34.11 Listening for System Broadcasts	337
34.12 Summary.....	339
35. A Basic Overview of Android Threads and Thread Handlers	341
35.1 An Overview of Threads	341
35.2 The Application Main Thread	341
35.3 Thread Handlers	341
35.4 A Basic Threading Example	342
35.5 Creating a New Thread	345
35.6 Implementing a Thread Handler.....	346
35.7 Passing a Message to the Handler	348
35.8 Summary.....	350
36. An Overview of Android Started and Bound Services	351
36.1 Started Services	351
36.2 Intent Service.....	352
36.3 Bound Service	352
36.4 The Anatomy of a Service	353
36.5 Controlling Destroyed Service Restart Options	353
36.6 Declaring a Service in the Manifest File.....	354
36.7 Starting a Service Running on System Startup.....	355
36.8 Summary.....	355
37. Implementing an Android Started Service – A Worked Example.....	357
37.1 Creating the Example Project	357
37.2 Creating the Service Class.....	357
37.3 Adding the Service to the Manifest File.....	359
37.4 Starting the Service.....	360
37.5 Testing the IntentService Example	361
37.6 Using the Service Class	361
37.7 Creating the New Service	362
37.8 Modifying the User Interface.....	363
37.9 Running the Application	365
37.10 Creating a New Thread for Service Tasks	365
37.11 Summary.....	367
38. Android Local Bound Services – A Worked Example	369
38.1 Understanding Bound Services	369
38.2 Bound Service Interaction Options.....	369
38.3 An Android Studio Local Bound Service Example	370
38.4 Adding a Bound Service to the Project	370

38.5 Implementing the Binder	371
38.6 Binding the Client to the Service.....	374
38.7 Completing the Example.....	375
38.8 Testing the Application	378
38.9 Summary.....	378
39. Android Remote Bound Services – A Worked Example	379
39.1 Client to Remote Service Communication	379
39.2 Creating the Example Application.....	379
39.3 Designing the User Interface	380
39.4 Implementing the Remote Bound Service	380
39.5 Configuring a Remote Service in the Manifest File	382
39.6 Launching and Binding to the Remote Service	383
39.7 Sending a Message to the Remote Service	385
39.8 Summary.....	385
40. An Overview of Android SQLite Databases.....	387
40.1 Understanding Database Tables	387
40.2 Introducing Database Schema	388
40.3 Columns and Data Types	388
40.4 Database Rows.....	388
40.5 Introducing Primary Keys.....	388
40.6 What is SQLite?	389
40.7 Structured Query Language (SQL).....	389
40.8 Trying SQLite on an Android Virtual Device (AVD)	390
40.9 Android SQLite Java Classes.....	392
40.9.1 <i>Cursor</i>	392
40.9.2 <i>SQLiteDatabase</i>	393
40.9.3 <i>SQLiteOpenHelper</i>	393
40.9.4 <i>ContentValues</i>	394
40.10 Summary	394
41. An Android TableLayout and TableRow Tutorial	395
41.1 The TableLayout and TableRow Layout Views.....	395
41.2 Creating the Database Project	397
41.3 Adding the TableLayout to the User Interface.....	397
41.4 Adding and Configuring the TableRows	398
41.5 Adding the Button Bar to the Layout.....	400
41.6 Adjusting the Layout Margins	401
41.7 Summary.....	404
42. An Android SQLite Database Tutorial	405

42.1 About the Database Example	405
42.2 Creating the Data Model	406
42.3 Implementing the Data Handler	407
42.3.1 <i>The Add Handler Method</i>	409
42.3.2 <i>The Query Handler Method</i>	410
42.3.3 <i>The Delete Handler Method</i>	411
42.4 Implementing the Activity Event Methods	411
42.5 Testing the Application	414
42.6 Summary	414
43. Understanding Android Content Providers	415
43.1 What is a Content Provider?	415
43.2 The Content Provider	415
43.2.1 <i>onCreate()</i>	416
43.2.2 <i>query()</i>	416
43.2.3 <i>insert()</i>	416
43.2.4 <i>update()</i>	416
43.2.5 <i>delete()</i>	416
43.2.6 <i>getType()</i>	416
43.3 The Content URI	416
43.4 The Content Resolver	417
43.5 The <provider> Manifest Element	417
43.6 Summary	418
44. Implementing an Android Content Provider in Android Studio	419
44.1 Copying the Database Project	419
44.2 Adding the Content Provider Package	419
44.3 Creating the Content Provider Class	420
44.4 Constructing the Authority and Content URI	422
44.5 Implementing URI Matching in the Content Provider	423
44.6 Implementing the Content Provider onCreate() Method	424
44.7 Implementing the Content Provider insert() Method	425
44.8 Implementing the Content Provider query() Method	426
44.9 Implementing the Content Provider update() Method	428
44.10 Implementing the Content Provider delete() Method	429
44.11 Declaring the Content Provider in the Manifest File	430
44.12 Modifying the Database Handler	431
44.13 Summary	434
45. Accessing Cloud Storage using the Android Storage Access Framework	435
45.1 The Storage Access Framework	435

45.2 Working with the Storage Access Framework	437
45.3 Filtering Picker File Listings	437
45.4 Handling Intent Results.....	439
45.5 Reading the Content of a File	439
45.6 Writing Content to a File.....	440
45.7 Deleting a File	441
45.8 Gaining Persistent Access to a File.....	441
45.9 Summary.....	442
46. An Android Storage Access Framework Example.....	443
46.1 About the Storage Access Framework Example	443
46.2 Creating the Storage Access Framework Example.....	443
46.3 Designing the User Interface	443
46.4 Declaring Request Codes	446
46.5 Creating a New Storage File.....	447
46.6 The onActivityResult() Method.....	448
46.7 Saving to a Storage File.....	450
46.8 Opening and Reading a Storage File	453
46.9 Testing the Storage Access Application	456
46.10 Summary.....	456
47. Implementing Video Playback on Android using the VideoView and MediaController Classes.....	457
47.1 Introducing the Android VideoView Class	457
47.2 Introducing the Android MediaController Class	458
47.3 Testing Video Playback	458
47.4 Creating the Video Playback Example	459
47.5 Designing the VideoPlayer Layout	459
47.6 Configuring the VideoView	460
47.7 Adding Internet Permission	461
47.8 Adding the MediaController to the Video View	462
47.9 Setting up the onPreparedListener.....	463
47.10 Summary.....	465
48. Video Recording and Image Capture on Android using Camera Intents.....	467
48.1 Checking for Camera Support.....	467
48.2 Calling the Video Capture Intent	468
48.3 Calling the Image Capture Intent.....	469
48.4 Creating an Android Studio Video Recording Project	470
48.5 Designing the User Interface Layout.....	470
48.6 Checking for the Camera	472
48.7 Launching the Video Capture Intent.....	473

48.8 Handling the Intent Return	474
48.9 Testing the Application	475
48.10 Summary	475
49. Android Audio Recording and Playback using MediaPlayer and MediaRecorder	477
49.1 Playing Audio	477
49.2 Recording Audio and Video using the MediaRecorder Class	478
49.3 About the Example Project	479
49.4 Creating the AudioApp Project	480
49.5 Designing the User Interface	480
49.6 Checking for Microphone Availability	481
49.7 Performing the Activity Initialization	482
49.8 Implementing the recordAudio() Method	483
49.9 Implementing the stopAudio() Method	484
49.10 Implementing the playAudio() method	485
49.11 Configuring Permissions in the Manifest File	485
49.12 Testing the Application	486
49.13 Summary	486
50. Working with the Google Maps Android API in Android Studio	487
50.1 The Elements of the Google Maps Android API	487
50.2 Creating the Google Maps Project	488
50.3 Obtaining Your Developer Signature	488
50.4 Testing the Application	490
50.5 Understanding Geocoding and Reverse Geocoding	491
50.6 Adding a Map to an Application	493
50.7 Displaying the User's Current Location	493
50.8 Changing the Map Type	494
50.9 Displaying Map Controls to the User	495
50.10 Handling Map Gesture Interaction	496
50.10.1 Map Zooming Gestures	496
50.10.2 Map Scrolling/Panning Gestures	497
50.10.3 Map Tilt Gestures	497
50.10.4 Map Rotation Gestures	497
50.11 Creating Map Markers	497
50.12 Controlling the Map Camera	499
50.13 Summary	501
51. Printing with the Android Printing Framework	503
51.1 The Android Printing Architecture	503
51.2 The HP Print Services Plugin	503

51.3 Google Cloud Print	504
51.4 Printing to Google Drive	505
51.5 Save as PDF	505
51.6 Printing from Android Devices	505
51.7 Options for Building Print Support into Android Apps	507
51.7.1 Image Printing	507
51.7.2 Creating and Printing HTML Content	508
51.7.3 Printing a Web Page	510
51.7.4 Printing a Custom Document	511
51.8 Summary	511
52. An Android HTML and Web Content Printing Example	513
52.1 Creating the HTML Printing Example Application	513
52.2 Printing Dynamic HTML Content	513
52.3 Creating the Web Page Printing Example	517
52.4 Designing the User Interface Layout	517
52.5 Loading the Web Page into the WebView	520
52.6 Adding the Print Menu Option	521
52.7 Summary	523
53. A Guide to Android Custom Document Printing	525
53.1 An Overview of Android Custom Document Printing	525
53.1.1 Custom Print Adapters	526
53.2 Preparing the Custom Document Printing Project	527
53.3 Creating the Custom Print Adapter	528
53.4 Implementing the onLayout() Callback Method	529
53.5 Implementing the onWrite() Callback Method	533
53.6 Checking a Page is in Range	536
53.7 Drawing the Content on the Page Canvas	537
53.8 Starting the Print Job	540
53.9 Testing the Application	541
53.10 Summary	542
54. Signing and Preparing an Android Application for Release	543
54.1 The Release Preparation Process	543
54.2 Changing the Build Variant	543
54.3 Creating a Keystore File	545
54.4 Generating a Private Key	546
54.5 Creating the Application APK File	547
54.6 Register for a Google Play Developer Console Account	550
54.7 Uploading New APK Versions to the Google Play Developer Console	550

54.8 Summary.....	552
55. Integrating Google Play In-app Billing into an Android Application.....	553
55.1 Installing the Google Play Billing Library.....	553
55.2 Creating the Example In-app Billing Project	554
55.3 Adding Billing Permission to the Manifest File	555
55.4 Adding the IInAppBillingService.aidl File to the Project	555
55.5 Adding the Utility Classes to the Project	557
55.6 Designing the User Interface	558
55.7 Implementing the “Click Me” Button	560
55.8 Google Play Developer Console and Google Wallet Accounts	562
55.9 Obtaining the Public License Key for the Application	562
55.10 Setting Up Google Play Billing in the Application	563
55.11 Initiating a Google Play In-app Billing Purchase	565
55.12 Implementing the onActivityResult Method	566
55.13 Implementing the Purchase Finished Listener	566
55.14 Consuming the Purchased Item.....	567
55.15 Releasing the IabHelper Instance	568
55.16 Modifying the Security.java File	569
55.17 Testing the In-app Billing Application	570
55.18 Building a Release APK.....	571
55.19 Creating a New In-app Product.....	572
55.20 Publishing the Application to the Alpha Distribution Channel	573
55.21 Adding In-app Billing Test Accounts	573
55.22 Configuring Group Testing.....	575
55.23 Resolving Problems with In-App Purchasing	576
55.24 Summary.....	577
56. An Overview of Gradle in Android Studio	579
56.1 An Overview of Gradle.....	579
56.2 Gradle and Android Studio	579
56.2.1 Sensible Defaults	580
56.2.2 Dependencies	580
56.2.3 Build Variants	580
56.2.4 Manifest Entries	581
56.2.5 APK Signing	581
56.2.6 ProGuard Support.....	581
56.3 The Top-level Gradle Build File	581
56.4 Module Level Gradle Build Files	583
56.5 Configuring Signing Settings in the Build File	585
56.6 Running Gradle Tasks from the Command-line.....	587

56.7 Summary	588
57. An Android Studio Gradle Build Variants Example	591
57.1 Creating the Build Variant Example Project.....	591
57.2 Adding the Build Flavors to the Module Build File.....	592
57.3 Adding the Flavors to the Project Structure	594
57.4 Adding Resource Files to the Flavors	595
57.5 Testing the Build Flavors	596
57.6 Build Variants and Class Files	597
57.7 Adding Packages to the Build Flavors	597
57.8 Customizing the Activity Classes	598
57.9 Summary	599

1. Introduction

The goal of this book is to teach the skills necessary to develop Android based applications using the Android Studio 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 overview of Android Studio is included covering areas such as tool windows, the code editor and the Designer tool. An introduction to the architecture of Android is followed by an in-depth look at the design of Android applications and user interfaces using the Android Studio environment. 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, in-app billing and submitting apps to the Google Play Developer Console.

Chapters also cover advanced features of Android Studio such as Gradle build configuration and the implementation of build variants to target multiple Android device types from a single project code base.

Assuming you already have some Java programming experience, are ready to download Android Studio 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 Android Studio project files for the examples contained in this book are available for download at:

<http://www.ebookfrenzy.com/print/androidstudio/index.php>

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/androidstudio.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 Studio 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) and the Android Studio Integrated Development Environment (IDE) which also includes the Android Software Development Kit (SDK).

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)

The Android SDK was 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:

Setting up an Android Studio Development Environment

```
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 Studio Package

Most of the work involved in developing applications for Android will be performed using the Android Studio environment. Android Studio may be downloaded from the following web page:

<http://developer.android.com/sdk/installing/studio.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 Android Studio 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 to initiate the download.

2.5 Installing Android Studio

Once downloaded, the exact steps to install Android Studio differ depending on the operating system on which the installation is being performed.

2.5.1 Installation on Windows

Locate the downloaded Android Studio installation executable file (named *android-studio-bundle-<version>.exe*) in a Windows Explorer window and double click on it to start the installation process, clicking the *Yes* button in the User Account Control dialog if it appears.

Once the Android Studio setup wizard appears, work through the various screens to configure the installation to meet your requirements in terms of the file system location into which Android Studio should be installed and whether or not it should be made available to other users of the system. Although there are no strict rules on where Android Studio should be installed on the system, the remainder of this book will assume that the installation was performed into a sub-folder of the user's home directory named *android-studio*. Once the options have been configured, click on the *Install* button to begin the installation process.

On versions of Windows with a Start menu, the newly installed Android Studio can be launched from the entry added to that menu during the installation. On Windows 8, the executable can be pinned to the task bar for easy access by navigating to the *android-studio\bin* directory, right-clicking on the executable and selecting the *Pin to Taskbar* menu option. Note that the executable is provided in 32-bit (*studio*) and 64-bit (*studio64*) executable versions. If you are running a 32-bit system be sure to use the *studio* executable.

2.5.2 Installation on Mac OS X

Android Studio for Mac OS X is downloaded in the form of a disk image (.dmg) file. Once the *android-studio-bundle-<version>-mac.dmg* file has been downloaded, locate it in a Finder window and double click on it to open it as shown in Figure 2-2:

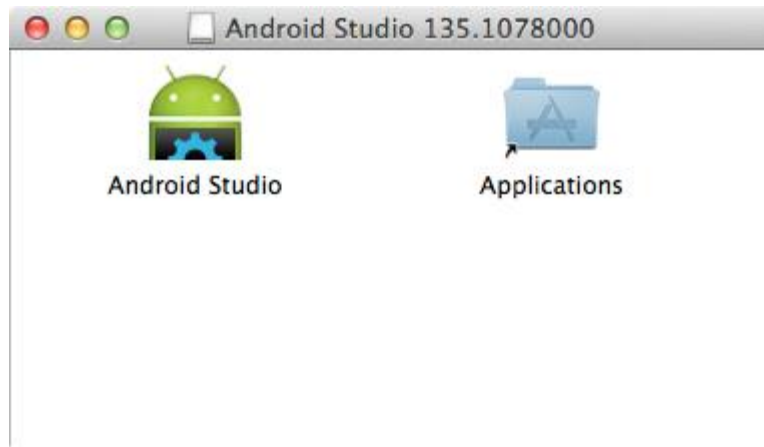


Figure 2-2

To install the package, simply drag the Android Studio icon and drop it onto the Applications icon. The Android Studio package will then be installed into the Applications folder of the system, a process which will typically take a few minutes to complete.

To launch Android Studio, locate the executable in the Applications folder using a Finder window and double click on it.

For future easier access to the tool, drag the Android Studio icon from the Finder window and drop it onto the dock.

2.5.3 Installation on Linux

Having downloaded the Linux Android Studio package, open a terminal window, change directory to the location where Android Studio is to be installed and execute the following command:

```
tar xvf /<path to package>/android-studio-bundle-<version>.tgz
```

Note that the Android Studio bundle will be installed into a sub-directory named *android-studio*. Assuming, therefore, that the above command was executed in */home/demo*, the software packages will be unpacked into */home/demo/android-studio*.

To launch Android Studio, open a terminal window, change directory to the *android-studio/bin* sub-directory and execute the following command:

```
./studio.sh
```

2.6 Installing the Latest Android SDK Packages

The steps performed so far have installed Java, the Android Studio 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 Android Studio tool by selecting the *Configure -> SDK Manager* option from within the Android Studio welcome dialog. Once invoked, the SDK Manager tool will appear as illustrated in Figure 2-3:

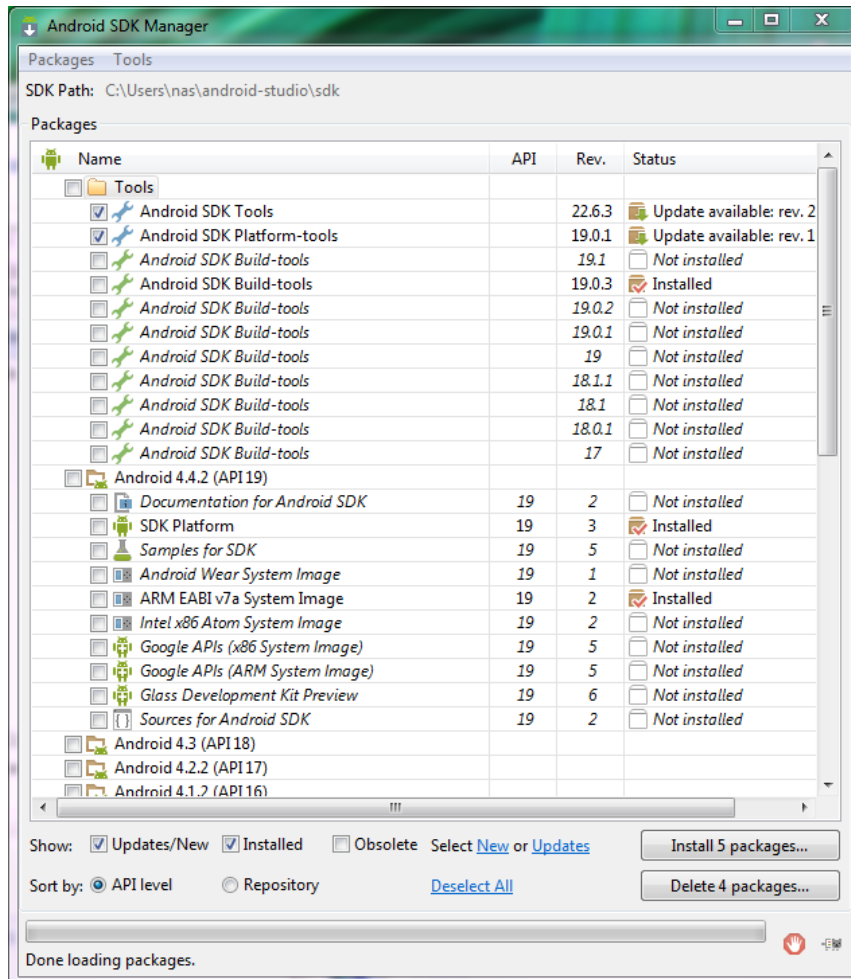


Figure 2-3

Begin by checking that the *SDK Path*: setting at the top of the SDK Manager window matches the location into which the Android Studio was installed. Within the Android SDK Manager, make sure that the check boxes next to the following packages are listed as *Installed* in the Status column:

Setting up an Android Studio Development Environment

- Tools > Android SDK Tools
 - Tools > Android SDK Platform-tools
 - SDK Platform (most recent version)> SDK Platform
 - SDK Platform (most recent version) > ARM EABI v7a System Image
- Note that the SDK Platform is now available in standard, L and W variants. For the purposes of this book it is not necessary to download the L and W versions of the SDK platform packages.
- Extras -> Android Support Repository
 - Extras > Android Support Library
 - Extras -> Google Repository
 - Extras -> Google USB Driver

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 Android Studio 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_android_studio_installation>* represents the file system location into which the ADT bundle was installed):

```
<path_to_android_studio_installation>/sdk/tools  
<path_to_android_studio_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 Android Studio was installed into */Users/demo/android-studio*, the following would be appended to the end of the current Path value:

```
;C:\Users\demo\android-studio\sdk\platform-tools;C:\Users\demo\android-studio\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 the 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.
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.

Setting up an Android Studio Development Environment

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 the 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 Android Studio bundle package was installed into */home/demo/android-studio*, the export line in the *.bashrc* file would now read as follows:

```
export PATH=/home/demo/java/jdk1.7.0_10/bin:/home/demo/android-  
studio/sdk/platform-tools:/home/demo/android-  
studio/sdk/tools:/home/demo/android-studio/bin:$PATH
```

Note also that the above command adds the *android-studio/bin* directory to PATH variable. This will enable the *studio.sh* script to be executed regardless of the current directory within a terminal window.

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 */Applications/Android Studio.app*, the path may be configured by creating a new file named *android-sdk* in the */etc/paths.d* directory containing the following lines:


```
/Applications/Android Studio.app/sdk/tools  
/Applications/Android Studio.app/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 Android Studio and the SDK

From time to time new versions of Android Studio and the Android SDK are released. New versions of the SDK are installed using the Android SDK Manager. Android Studio will typically notify you when an update is ready to be installed.

To manually check for Android Studio updates, click on the *Check for updates now* link located at the bottom of the Android Studio welcome screen, or use the *Help -> Check for Update...* menu option accessible from within the Android Studio main window.

2.9 Summary

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

3. Creating an Example Android App in Android Studio

The preceding chapters of this book have covered the steps necessary to configure an environment suitable for the development of Android applications using the Android Studio IDE. 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 an Android application and compile and run it. This chapter will cover the creation of a simple Android application project using Android Studio. Once the project has been created, a later chapter will explore the use of the Android emulator environment to perform a test run of the application.

3.1 Creating a New Android Project

The first step in the application development process is to create a new project within the Android Studio environment. Begin, therefore, by launching Android Studio so that the “Welcome to Android Studio” screen appears as illustrated in Figure 3-1:



Figure 3-1

Creating an Example Android App in Android Studio

Once this window appears, Android Studio is ready for a new project to be created. To create the new project, simply click on the *New Project...* option to display the first screen of the *New Project* wizard as shown in Figure 3-2:

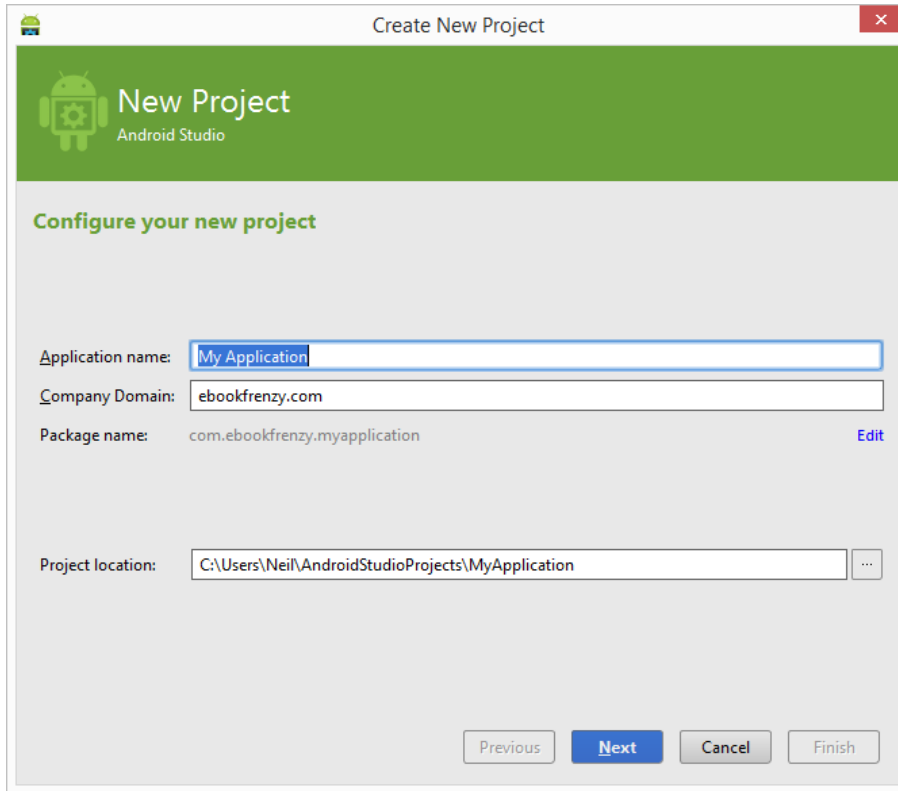


Figure 3-2

3.2 Defining the Project and SDK Settings

In the *New Project* window, set both the *Application name* field to *AndroidSample*. The application name is the name by which the application will be referenced and identified within Android Studio and is also the name that will be used when the completed application goes on sale in the Google Play store.

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 *AndroidSample*, then the package name might be specified as follows:

```
com.mycompany.androidsample
```

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

```
com.ebookfrenzy.androidsample
```

The *Project location* setting will default to a location in the folder named *AndroidStudioProjects* located in your home directory and may be changed by clicking on the button to the right of the text field containing the current path setting.

Click Next to proceed. On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 8: Android 2.2 (Froyo). Since the project is not intended for Google TV, Google Glass or wearable devices, leave the remaining options disabled before clicking *Next*.

On the final screen, enter *AndroidSampleActivity* for both the Activity Name and Title fields and name the layout *activity_android_sample*.

3.3 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 *Blank Activity* before clicking *Next*.

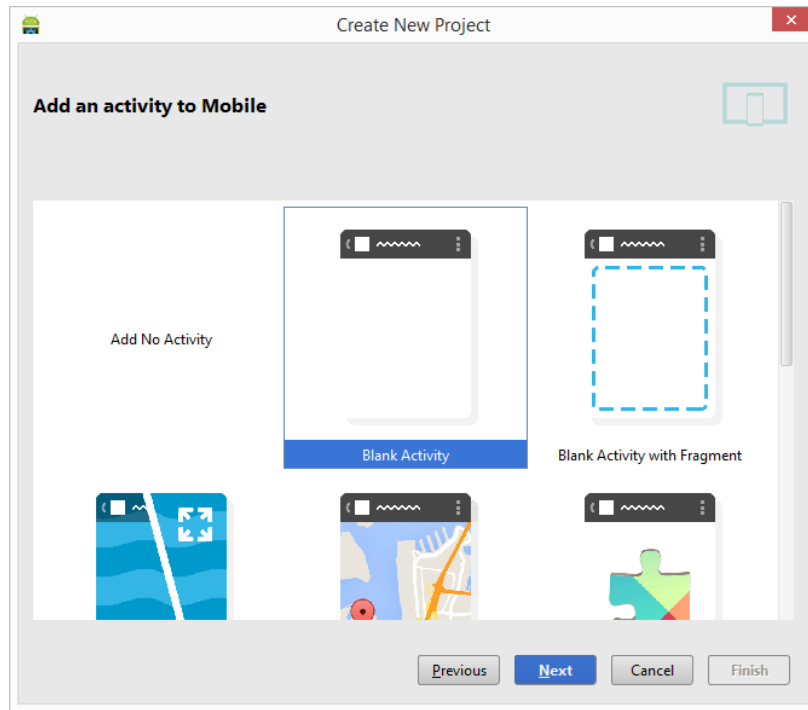


Figure 3-3

With the Blank Activity option selected, click *Next*. On the final screen (Figure 3-4) name the activity and title *AndroidSampleActivity*. The activity will consist of a single user interface screen layout which, for the purposes of this example, should be named *activity_android_sample* as shown in Figure 3-4:

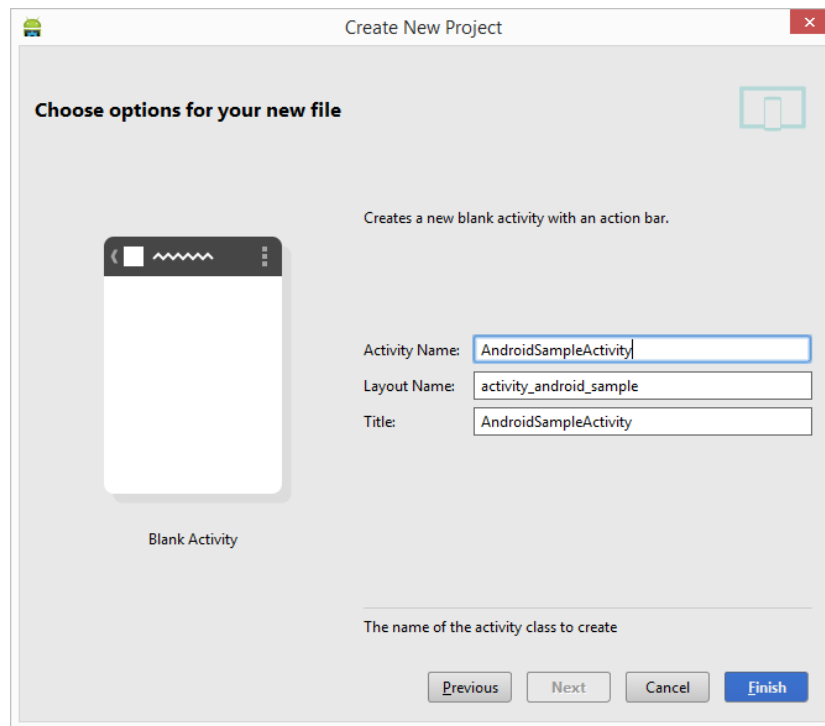


Figure 3-4

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

3.4 Modifying the Example Application

At this point, Android Studio has created a minimal example application project and opened the main window.

The newly created project and references to associated files are listed in the *Project* tool window located on the left hand side of the main project window. This essentially mirrors the directory hierarchy and files located in the project folder on the local file system of the development computer:

Creating an Example Android App in Android Studio

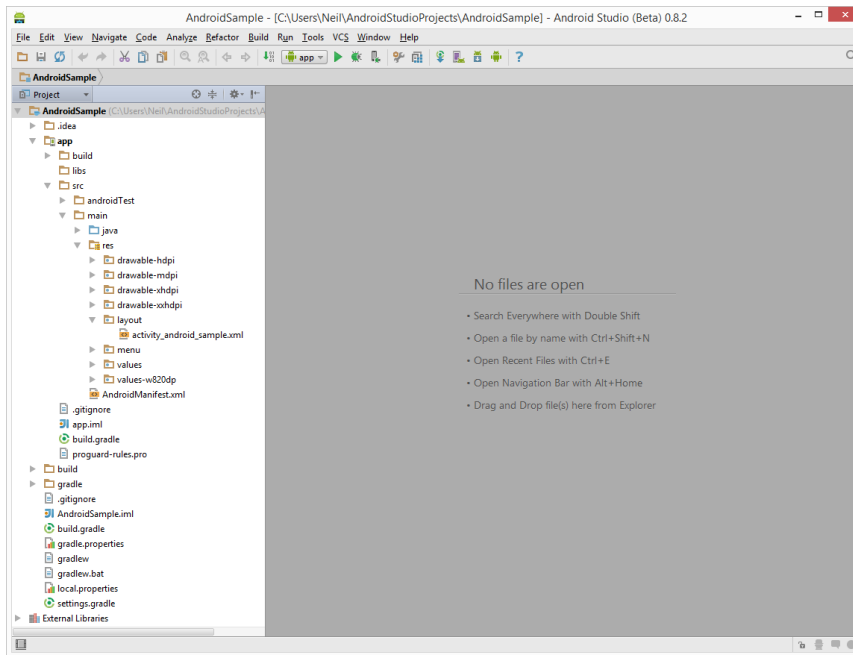


Figure 3-5

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.

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 Android Studio.

The user interface design for our activity is stored in a file named *activity_android_sample.xml* which, in turn, is located under *AndroidSample* -> *app* -> *src* -> *res* -> *layout* in the project file hierarchy. Using the Project tool window, locate this file as illustrated in Figure 3-6:

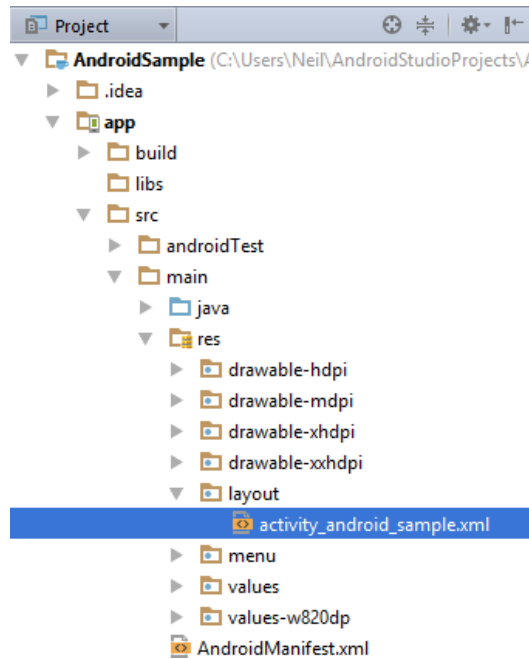


Figure 3-6

Once located, double click on the file to load it into the User Interface Designer tool which will appear in the center panel of the Android Studio main window:

Creating an Example Android App in Android Studio

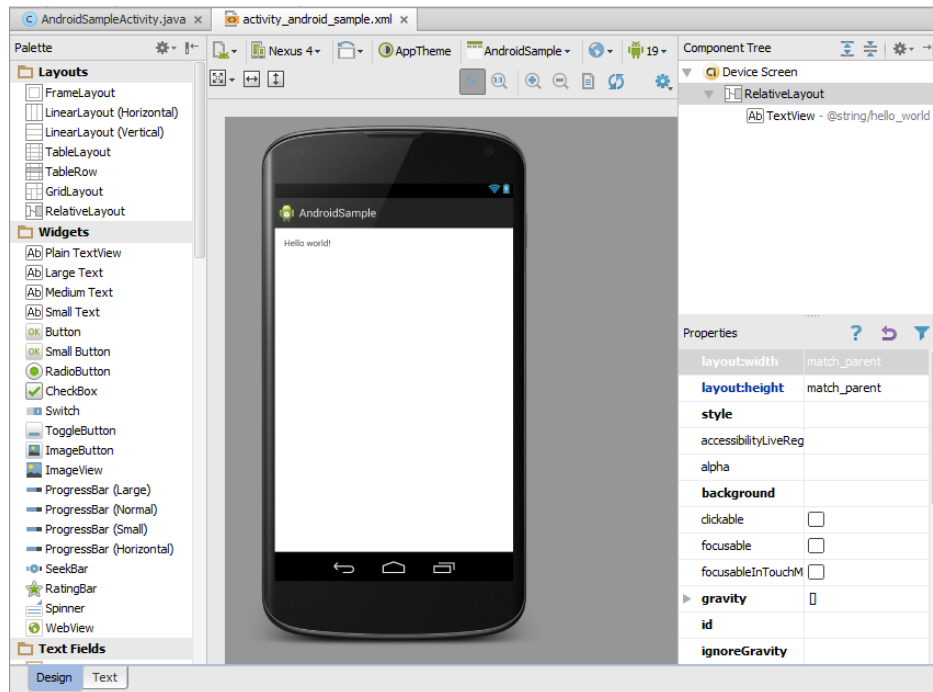



Figure 3-7

In the toolbar across the top of Designer window is a menu currently set to *Nexus 4* which is reflected in the visual representation of the device within the Designer panel. A wide range of other device options are available for selection by clicking on this menu.

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

As can be seen in the device screen, the layout already includes a label that displays a 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 *Component Tree* panel which, by default, is located in the upper right hand corner of the Designer panel and is shown in Figure 3-8:

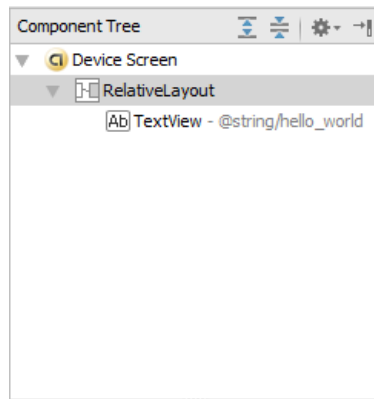


Figure 3-8

As we can see from the component tree hierarchy, the user interface consists of a `RelativeLayout` parent with a single child in the form of a `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 to remove the object from the layout.

In the Palette panel, locate the *Widgets* category. Click and drag the *Large Text* object and drop it in the center of the user interface design when the green marker lines appear to indicate the center of the display:

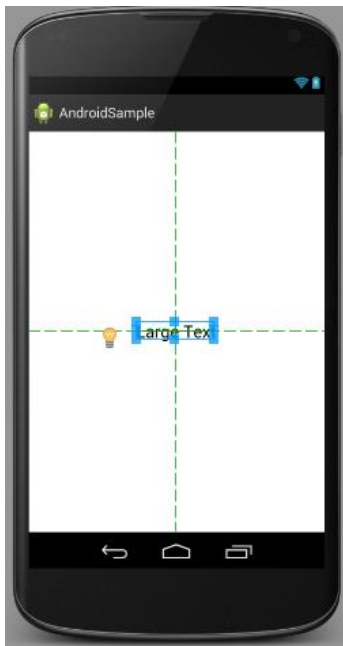


Figure 3-9

The Android Studio Designer tool also provides an alternative to dragging and dropping components from the palette on to the design layout. Components may also be added by selecting the required object from the palette and then simply clicking on the layout at the location where the component is to be placed.

The next step is to change the text that is currently displayed by the TextView component. Double click on the object in the design layout to display the text and id editing panel as illustrated in Figure 3-10. Within the panel, change the text property from “Large Text” to “Welcome to Android Studio”.

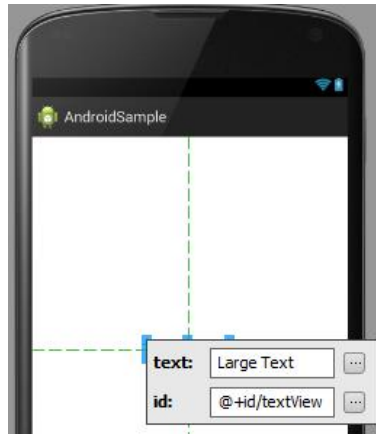


Figure 3-10

At this point it is important to explain the light bulb next to the TextView object in the layout. This indicates a possible problem and provides some recommended solutions. Clicking on the icon in this instance informs us that the problem is as follows:

```
[I18N] Hardcoded string "Welcome to Android Studio", should use @string resource
```

This I18N message is informing us that a potential issue exists with regard to the future internationalization of the project ("I18N" comes from the fact that the word "internationalization" begins with an "I", ends with an "N" and has 18 letters in between). The warning is reminding us that when developing Android 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 translation work 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 to it the string "Welcome to Android Studio".

Click on the arrow to the right of the warning message to display the menu of possible solutions (Figure 3-11).

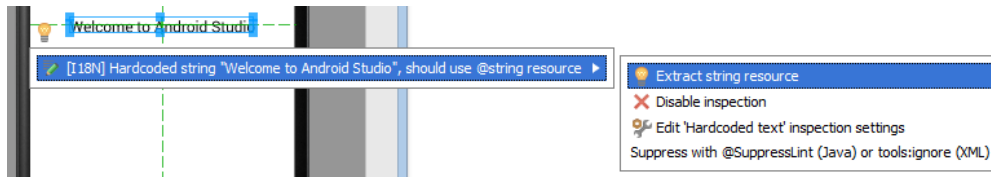


Figure 3-11

From the menu, select the *Extract string resource* option to display the *Extract Resource* dialog. In this dialog, enter *welcomestring* into the *Resource name:* field before clicking on *OK*. The string is now stored as a resource in the *src -> res -> values -> strings.xml* file.

3.5 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 design and resource handling. In the previous section, we made some changes to the user interface by modifying the *activity_android_sample.xml* file using the UI Designer tool. In fact, all that the Designer 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 may actually be quicker than using the Designer tool. At the bottom of the Designer panel are two tabs labeled *Design* and *Text* respectively. To switch to the XML view simply select the *Text* tab as shown in Figure 3-12:

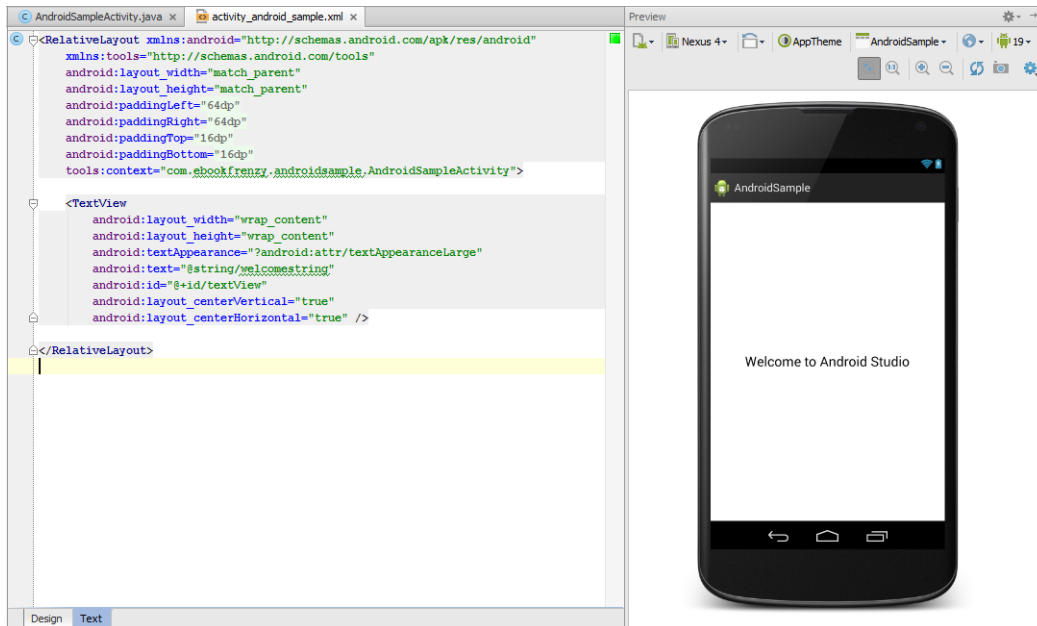


Figure 3-12

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.

One of the more powerful features of Android Studio can be found to the right hand side of the XML editing panel. This is the Preview panel and shows the current visual state of the layout. As changes are made to the XML layout, these will be reflected in the preview panel. To see this in action, modify the XML layout to change the background color of the `RelativeLayout` to a shade of red as follows:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".AndroidSampleActivity"
    android:background="#ff2438">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/welcomestring"
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Note that the color of the preview changes in real-time to match the new setting in the XML file. Note also that a small red square appears in the left hand margin (also referred to as the *gutter*) of the XML editor next to the line containing the color setting. This is a visual cue to the fact that the color red has been set on a property. Change the color value to `#a0ff28` and note that both the small square in the margin and the preview change to green.

Finally, use the Project view to locate the `app -> src -> res -> values -> strings.xml` file and double click on it to load it into the editor. Currently the XML should read as follows:

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

```
<resources>

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

</resources>
```

As a demonstration of resources in action, change the string value currently assigned to the *welcomestring* resource and then return to the Designer by selecting the tab for the layout file in the editor panel. Note that the layout has picked up the new resource value for the welcome string.

There is also a quick way to access the value of a resource referenced in an XML file. With the Designer tool in Text mode, click on the “@string/welcomestring” property setting so that it highlights and then press Ctrl+B on the keyboard. Android Studio will subsequently open the *strings.xml* file and take you to the line in that file where this resource is declared. Use this opportunity to revert the string resource back to the original “Welcome to Android Studio” text.

3.6 Previewing the Layout

So far in this chapter, the layout has only been previewed on a representation of the Nexus 4 device. As previously discussed, the layout can be tested for other devices by making selections from the device menu in the toolbar across the top edge of the Designer panel. Another useful option provided by this menu is *Preview All Screen Sizes* which, when selected, shows the layout in all currently configured device configurations as demonstrated in Figure 3-13:

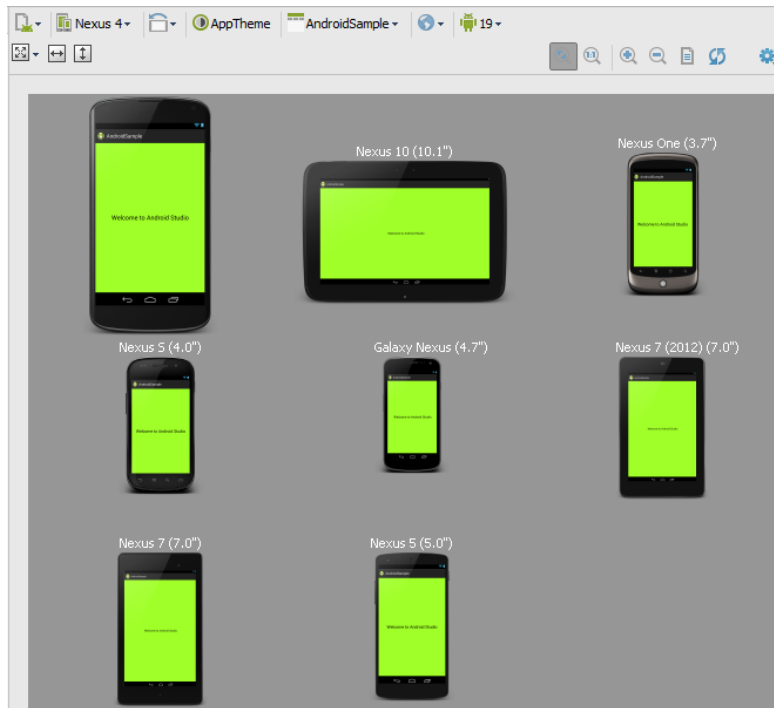


Figure 3-13

To revert to a single preview layout, select the device menu once again, this time choosing the *Remove Previews* option.

3.7 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 a simple application and then used the Android Studio UI Designer to modify the user interface layout. In doing so, 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.

Whilst it is useful to be able to preview a layout from within the Android Studio Designer tool, there is no substitute for testing an application by compiling and running it. In a later chapter entitled *Creating an Android Virtual Device (AVD) in Android Studio*, the steps necessary to set up an emulator for testing purposes will be covered in detail. Before running the application, however, the next chapter will take a small detour to provide a guided tour of the Android Studio user interface.

4. A Tour of the Android Studio User Interface

Whilst it is tempting to plunge into running the example application created in the previous chapter, doing so involves using aspects of the Android Studio user interface which are best described in advance.

Android Studio is a powerful and feature rich development environment that is, to a large extent, intuitive to use. That being said, taking the time now to gain familiarity with the layout and organization of the Android Studio user interface will considerably shorten the learning curve in later chapters of the book. With this in mind, this chapter will provide an initial overview of the various areas and components that make up the Android Studio environment.

4.1 The Welcome Screen

The welcome screen (Figure 4-1) is displayed any time that Android Studio is running with no projects currently open (open projects can be closed at any time by selecting the *File -> Close Project* menu option). If Android Studio was previously exited while a project was still open, the tool will by-pass the welcome screen next time it is launched, automatically opening the previously active project.

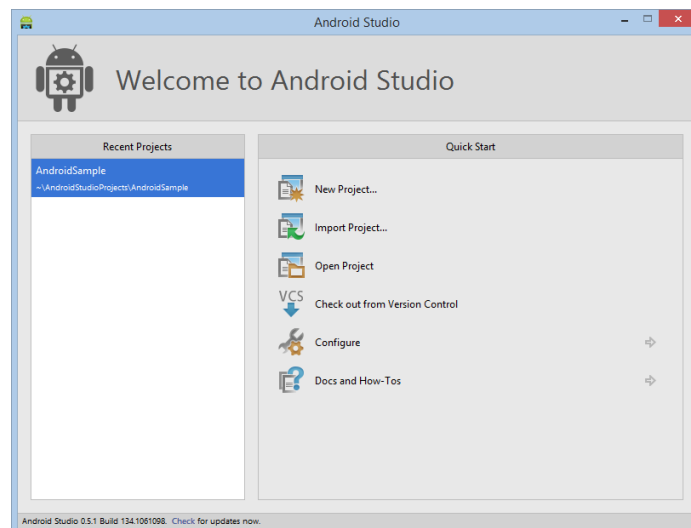


Figure 4-1

A Tour of the Android Studio User Interface

In addition to a list of recent projects, the Quick Start menu provides a range of options for performing tasks such as opening, creating and importing projects along with access to projects currently under version control. In addition, the *Configure* option provides access to the SDK Manager along with a vast array of settings and configuration options. A review of these options will quickly reveal that there is almost no aspect of Android Studio that cannot be configured and tailored to your specific needs.

Finally, the status bar along the bottom edge of the window provides information about the version of Android Studio currently running, along with a link to check if updates are available for download.

4.2 The Main Window

When a new project is created, or an existing one opened, the Android Studio *main window* will appear. When multiple projects are open simultaneously, each will be assigned its own main window. The precise configuration of the window will vary depending on which tools and panels were displayed the last time the project was open, but will typically resemble that of Figure 4-2.

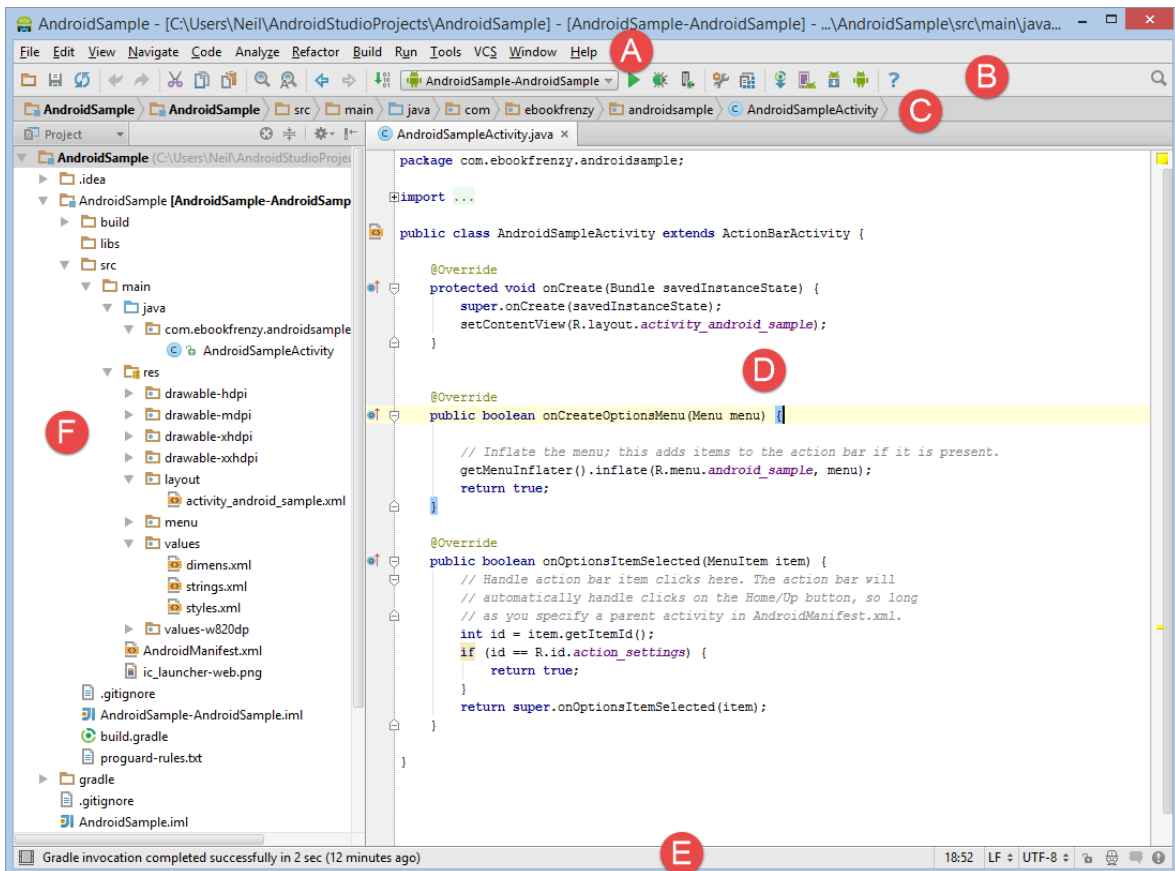


Figure 4-2

The various elements of the main window can be summarized as follows:

A – Menu Bar – Contains a range of menus for performing tasks within the Android Studio environment.

B – Toolbar – A selection of shortcuts to frequently performed actions. The toolbar buttons provide quicker access to a select group of menu bar actions. The toolbar can be customized by right-clicking on the bar and selecting the *Customize Menus and Toolbars...* menu option.

C – Navigation Bar – The navigation bar provides a convenient way to move around the files and folders that make up the project. Clicking on an element in the navigation bar will drop down a menu listing the subfolders and files at that location ready for selection. This provides an alternative to the Project tool window.

D – Editor Window – The editor window displays the content of the file on which the developer is currently working. What gets displayed in this location, however, is subject to context. When editing code, for example, the code editor will appear. When working on a user interface layout file, on the other hand, the user interface Designer tool will appear. When multiple files are open, each file is represented by a tab located along the top edge of the editor as shown in Figure 4-3.

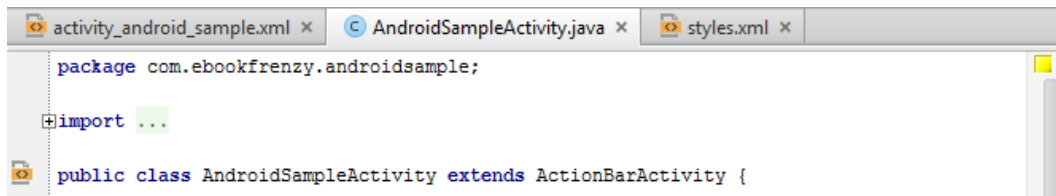


Figure 4-3

E – Status Bar – The status bar displays informational messages about the project and the activities of Android Studio together with the tools menu button located in the far left corner. Hovering over items in the status bar will provide a description of that field. Many fields are interactive, allowing the user to click to perform tasks or obtain more detailed status information.

F – Project Tool Window – The project tool window provides a hierarchical overview of the project file structure allowing navigation to specific files and folders to be performed.

The project tool window is just one of a number of tool windows available within the Android Studio environment.

4.3 The Tool Windows

In addition to the project view tool window, Android Studio also includes a number of other windows which, when enabled, are displayed along the bottom and sides of the main window. The tool window

A Tour of the Android Studio User Interface

quick access menu can be accessed by hovering the mouse pointer over the button located in the far left hand corner of the status bar (Figure 4-4) without clicking the mouse button.

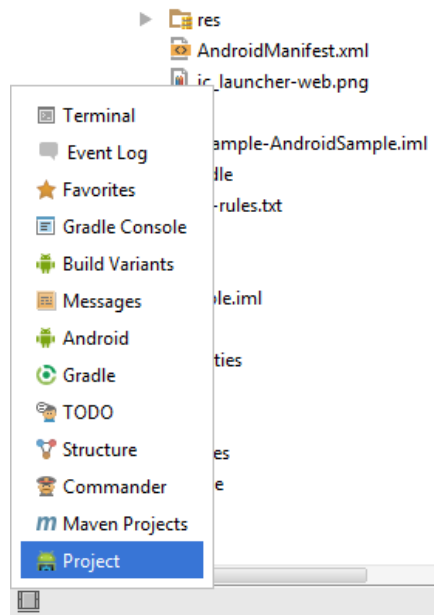


Figure 4-4

Selecting an item from the quick access menu will cause the corresponding tool window to appear within the main window.

Alternatively, a set of *tool window bars* can be displayed by clicking on the quick access menu icon in the status bar. These bars appear along the left, right and bottom edges of the main window (as indicated by the arrows in Figure 4-5) and contain buttons for showing and hiding each of the tool windows. When the tool window bars are displayed, a second click on the button in the status bar will hide them.

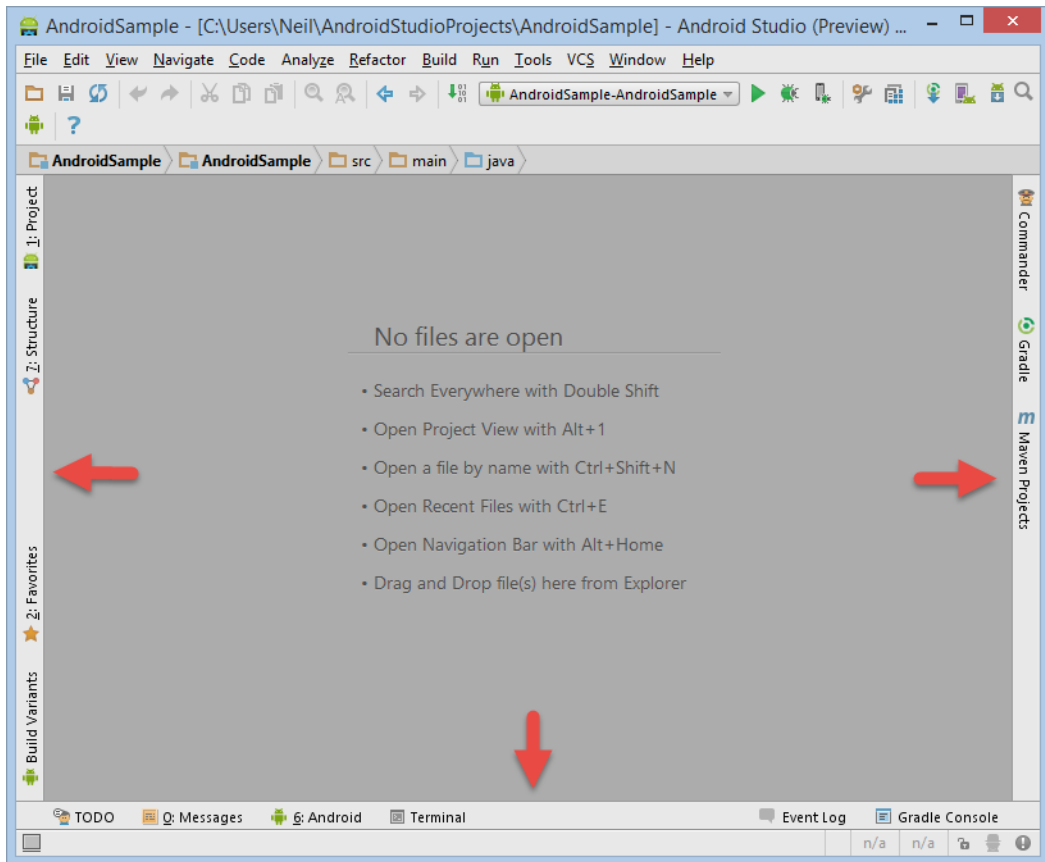


Figure 4-5

Clicking on a button will display the corresponding tool window whilst a second click will hide the window. Buttons prefixed with a number (for example 1: Project) indicate that the tool window may also be displayed by pressing the Alt key on the keyboard (or the Command key for Mac OS X) together with the corresponding number.

The location of a button in a tool window bar indicates the side of the window against which the window will appear when displayed. These positions can be changed by clicking and dragging the buttons to different locations in other window tool bars.

Each tool window has its own toolbar along the top edge. The buttons within these toolbars vary from one tool to the next, though all tool windows contain a settings option, represented by the cog icon, which allows various aspects of the window to be changed. Figure 4-6 shows the settings menu for the project view tool window. Options are available, for example, to undock a window and to allow it to float outside of the boundaries of the Android Studio main window.

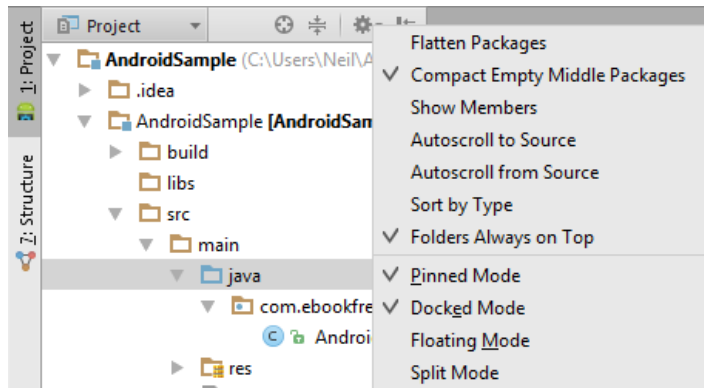


Figure 4-6

All of the windows also include a far right button on the toolbar providing an additional way to hide the tool window from view. A search of the items within a tool window can be performed simply by giving that window focus by clicking in it and then typing the search term (for example the name of a file in the Project tool window). A search box will appear in the window's tool bar and items matching the search highlighted.

Android Studio offers a wide range of window tool windows, the most commonly used of which are as follows:

Project – The project view provides an overview of the file structure that makes up the project allowing for quick navigation between files. Generally, double clicking on a file in the project view will cause that file to be loaded into the appropriate editing tool.

Structure – The structure tool provides a high level view of the structure of the source file currently displayed in the editor. This information includes a list of items such as classes, methods and variables in the file. Selecting an item from the structure list will take you to that location in the source file in the editor window.

Favorites – A variety of project items can be added to the favorites list. Right-clicking on a file in the project view, for example, provides access to an *Add to Favorites* menu option. Similarly, a method in a source file can be added as a favorite by right-clicking on it in the Structure tool window. Anything added to a Favorites list can be accessed through this Favorites tool window.

Build Variants – The build variants tool window provides a quick way to configure different build targets for the current application project (for example different builds for debugging and release versions of the application, or multiple builds to target different device categories).

TODO – As the name suggests, this tool provides a place to review items that have yet to be completed on the project. Android Studio compiles this list by scanning the source files that make up the project to

look for comments that match specified TODO patterns. These patterns can be reviewed and changed by selecting the *File -> Settings...* menu option and navigating to the *TODO* page listed under *IDE Settings*.

Messages – The messages tool window records output from the Gradle build system (Gradle is the underlying system used by Android Studio for building the various parts of projects into a runnable applications) and can be useful for identifying the causes of build problems when compiling application projects.

Android – The Android tool window provides access to the Android debugging system. Within this window tasks such as monitoring log output from a running application, taking screenshots and videos of the application, stopping a process and performing basic debugging tasks can be performed.

Terminal – Provides access to a terminal window on the system on which Android Studio is running. On Windows systems this is the Command Prompt interface, whilst on Linux and Mac OS X systems this takes the form of a Terminal prompt.

Run – The run tool window becomes available when an application is currently running and provides a view of the results of the run together with options to stop or restart a running process. If an application is failing to install and run on a device or emulator, this window will typically provide diagnostic information relating to the problem.

Event Log – The event log window displays messages relating to events and activities performed within Android Studio. The successful build of a project, for example, or the fact that an application is now running will be reported within this window tool.

Gradle Console – The Gradle console is used to display all output from the Gradle system as projects are built from within Android Studio. This will include information about the success or otherwise of the build process together with details of any errors or warnings.

Maven Projects – Maven is a project management and build system designed to ease the development of complex Java based projects and overlaps in many areas with the functionality provided by Gradle. Google has chosen Gradle as the underlying build system for Android development, so unless you are already familiar with Maven or have existing Maven projects to import, your time will be better spent learning and adopting Gradle for your projects. The Maven projects tool window can be used to add, manage and import Maven based projects within Android Studio.

Gradle – The Gradle tool window provides a view onto the Gradle tasks that make up the project build configuration. The window lists the tasks that are involved in compiling the various elements of the project into an executable application. Right-click on a top level Gradle task and select the *Open Gradle Config* menu option to load the Gradle build file for the current project into the editor. Gradle will be covered in greater detail later in this book.

Commander – The Commander window tool can best be described as a combination of the Project and Structure tool windows, allowing the file hierarchy of the project to be traversed and for the various elements that make up classes to be inspected and loaded into the editor or designer windows.

Designer – Available when the UI Designer is active, this tool window provides access to the designer’s Component Tree and Properties panels.

4.4 Android Studio Keyboard Shortcuts

Android Studio includes an abundance of keyboard shortcuts designed to save time when performing common tasks. A full keyboard shortcut keymap listing can be viewed and printed from within the Android Studio project window by selecting the *Help -> Default Keymap Reference* menu option.

4.5 Switcher and Recent Files Navigation

Another useful mechanism for navigating within the Android Studio main window involves the use of the *Switcher*. Accessed via the *Ctrl-Tab* keyboard shortcut, the switcher appears as a panel listing both the tool windows and currently open files (Figure 4-7).

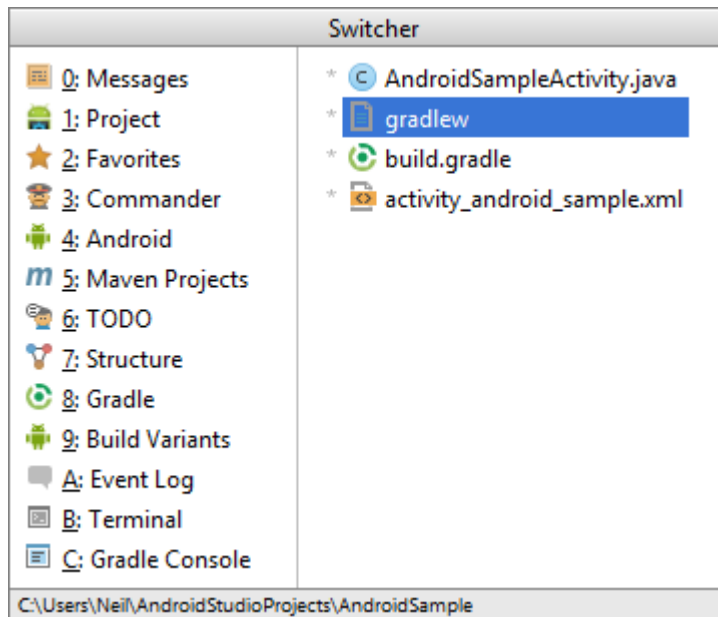


Figure 4-7

Once displayed, the switcher will remain visible for as long the Ctrl key remains depressed. Repeatedly tapping the Tab key whilst holding down the Ctrl key will cycle through the various selection options, whilst releasing the Ctrl key causes the currently highlighted item to be selected and displayed within the main window.

In addition to the switcher, navigation to recently opened files is provided by the Recent Files panel (Figure 4-8). This can be accessed using the Ctrl-E keyboard shortcut (Cmd-E on Mac OS X). Once displayed, either the mouse pointer can be used to select an option or, alternatively, the keyboard arrow keys can be used to scroll through the file name and tool window options. Pressing the Enter key will select the currently highlighted item.

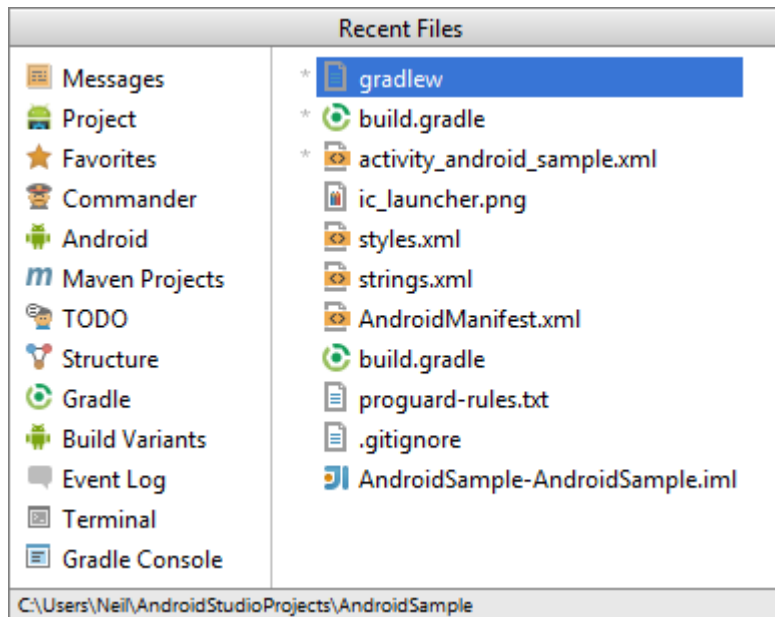


Figure 4-8

4.6 Changing the Android Studio Theme

The overall theme of the Android Studio environment may be changed either from the welcome screen using the *Configure -> Settings* option, or via the *File -> Settings...* menu option of the main window.

Once the settings dialog is displayed, select the *Appearance* option in the left hand panel and then change the setting of the *Theme* menu before clicking on the *Apply* button. The themes currently available consist of IntelliJ, Windows and Darcula. Figure 4-9 shows an example of the main window with the Darcula theme selected:

A Tour of the Android Studio User Interface

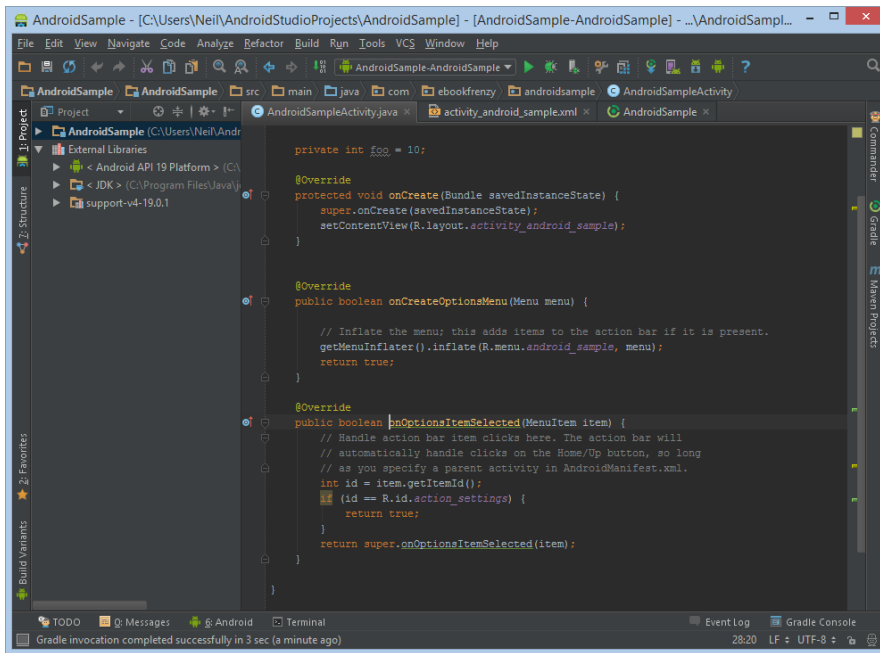


Figure 4-9

4.7 Summary

The primary elements of the Android Studio environment consist of the welcome screen and main window. Each open project is assigned its own main window which, in turn, consists of a menu bar, toolbar, editing and design area, status bar and a collection of tool windows. Tool windows appear on the sides and bottom edges of the main window and can be accessed either using the quick access menu located in the status bar, or via the optional tool window bars.

There are very few actions within Android Studio which cannot be triggered via a keyboard shortcut. A keymap of default keyboard shortcuts can be accessed at any time from within the Android Studio main window.

5. Creating an Android Virtual Device (AVD) in Android Studio

In the course of developing Android apps in Android Studio 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.

5.1 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 standard Android Studio installation, a number of emulator templates are 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 processor type, memory capacity and the size and pixel density of the screen. Check the online developer documentation for your device to find out if emulator definitions are available for download and installation into the AVD environment.

When launched, an AVD will appear as a window containing an emulated Android device environment. Figure 5-1, for example, shows an AVD session configured to emulate the Google Nexus 7 device.

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

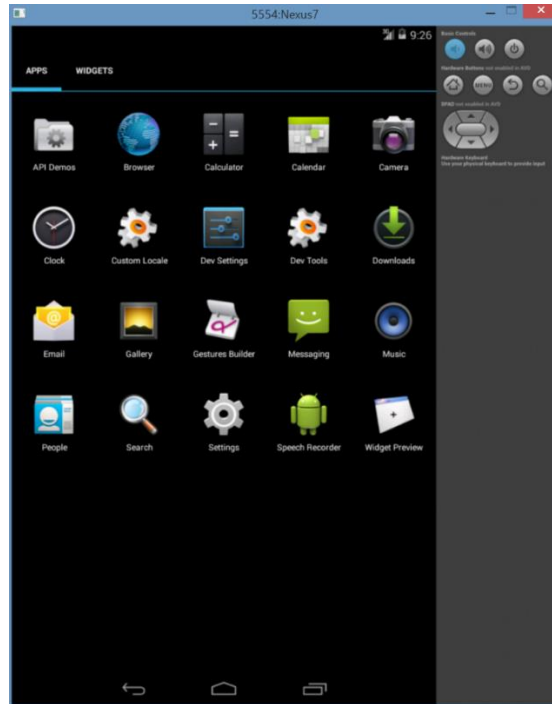


Figure 5-1

5.2 Creating a New AVD

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

To create a new AVD, the first step is to launch the AVD Manager. This can be achieved from within the Android Studio environment by selecting the *Tools -> Android -> AVD Manager* menu option from within the main window. Alternatively, the tool may be launched from a terminal or command-line prompt using the following command:

```
android avd
```

Once launched, the tool will appear as outlined in Figure 5-2. Assuming a new Android SDK installation, no AVDs will currently be listed:

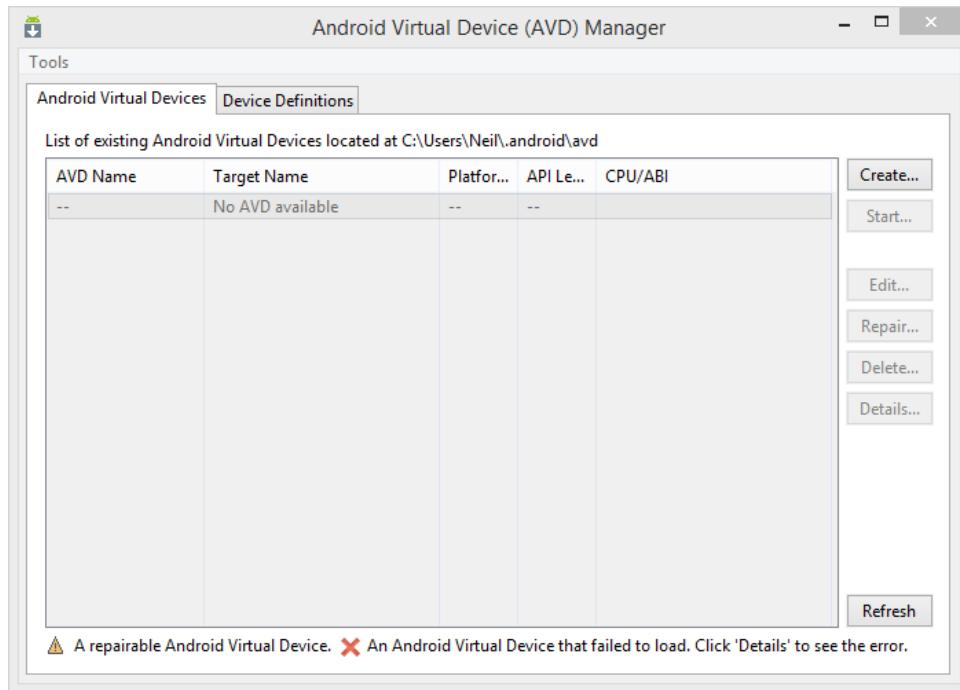


Figure 5-2

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 (2012) (7.0", 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. Set the *Skin* menu to "no skin" so that the device is displayed without simulated hardware controls.
6. 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.
7. If the host computer contains a web cam the *Front Camera*: emulation may be configured to use this camera. Alternatively, an emulated camera may be selected. If camera functionality is not required by the application, simply leave this set to *None*.

Whether or not you enable the *Hardware keyboard present* option is a matter of personal preference. When the hardware keyboard option is selected, it will be possible to use the physical keyboard on the

Creating an Android Virtual Device (AVD) in Android Studio

system on which the emulator is running. As such, the Android software keyboard will not appear within the emulator when text input is required by a running application.

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 5-3 illustrates the dialog with the appropriate settings implemented for a Nexus 7 emulator. Once the configuration settings are complete, click on the *OK* button.

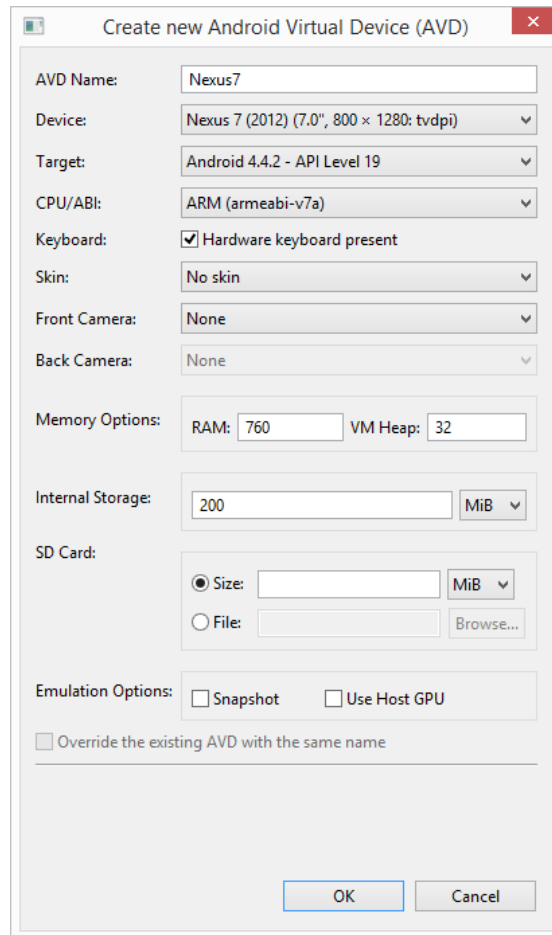


Figure 5-3

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.

5.3 Starting the Emulator

To perform a test run of the newly created AVD emulator, simply select the emulator from the AVD 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 system will detect that it is already running and attach to it when applications are launched, thereby saving considerable amounts of startup time.

Another option when using the emulator is to enable the *Snapshot* option in the AVD settings screen. This option, which can only be used when the *Use Host GPU* option is disabled, enables the state of an AVD instance to be saved and reloaded next time it is launched. This can result in an emulator startup time of just a few seconds.

When using snapshots, make sure that the *Launch from snapshot* and *Save to snapshot* options are both enabled in the Launch Options dialog. The first time an emulator instance is launched with snapshots enabled, the AVD will start in the normal manner. On closing the emulator window, a slight delay will occur while the state of the AVD is saved. Next time the emulator is launched it will load more quickly and revert to the previously saved state.

To revert the AVD to its original, pre-snapshot state, simply launch it with the *Launch from snapshot* option disabled.

To save time in the next section of this chapter, leave the emulator running before proceeding.

5.4 Running the Application in the AVD

With an AVD emulator configured, the example *AndroidSample* application created in the earlier chapter now can be compiled and run. With the *AndroidSample* project loaded into Android Studio, simply click on the run button represented by a green triangle located in the Android Studio toolbar as shown in Figure 5-4 below, select the *Run -> Run...* menu option or use the Shift+F10 keyboard shortcut:

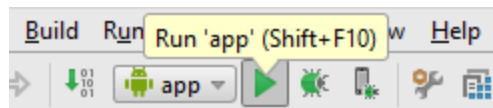


Figure 5-4

By default, Android Studio will respond to the run request by displaying the *Choose Device* dialog. This provides the option to execute the application on an AVD instance that is already running, or to launch a new AVD session specifically for this application. Figure 5-5 lists the previously created Nexus7 AVD as