

Android Studio Development Essentials

Android 6 Edition

Android Studio Development Essentials – Android 6 Edition

ISBN-13: 978-1519722089

© 2015 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 Download the eBook.....	2
1.3 Feedback	2
1.4 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 The Android Studio Setup Wizard	9
2.7 Installing Additional Android SDK Packages	10
2.8 Making the Android SDK Tools Command-line Accessible.....	13
2.8.1 Windows 7	14
2.8.2 Windows 8.1	15
2.8.3 Windows 10	16
2.8.4 Linux.....	16
2.8.5 Mac OS X.....	16
2.9 Updating the Android Studio and the SDK	16
2.10 Summary	17
3. Creating an Example Android App in Android Studio	19
3.1 Creating a New Android Project	19
3.2 Defining the Project and SDK Settings.....	20
3.3 Creating an Activity	21
3.4 Modifying the Example Application	23
3.5 Reviewing the Layout and Resource Files	28
3.6 Previewing the Layout.....	30
3.7 Summary	31
4. A Tour of the Android Studio User Interface	33

4.1 The Welcome Screen	33
4.2 The Main Window	34
4.3 The Tool Windows	35
4.4 Android Studio Keyboard Shortcuts	39
4.5 Switcher and Recent Files Navigation.....	39
4.6 Changing the Android Studio Theme.....	40
4.7 Summary.....	41
5. Creating an Android Virtual Device (AVD) in Android Studio	43
5.1 About Android Virtual Devices	43
5.2 Creating a New AVD	44
5.3 Starting the Emulator	45
5.4 Running the Application in the AVD	46
5.5 Run/Debug Configurations	48
5.6 Stopping a Running Application.....	49
5.7 AVD Command-line Creation.....	51
5.8 Android Virtual Device Configuration Files.....	53
5.9 Moving and Renaming an Android Virtual Device	54
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 6.0 based Devices	56
6.2.1 Mac OS X ADB Configuration	57
6.2.2 Windows ADB Configuration.....	57
6.2.3 Linux adb Configuration	59
6.3 Testing the adb Connection.....	60
6.4 Summary.....	61
7. The Basics of the Android Studio Code Editor	63
7.1 The Android Studio Editor	63
7.2 Splitting the Editor Window	66
7.3 Code Completion	67
7.4 Statement Completion	68
7.5 Parameter Information.....	69
7.6 Code Generation.....	69
7.7 Code Folding	70
7.8 Quick Documentation Lookup	72
7.9 Code Reformatting	73
7.10 Summary.....	73
8. An Overview of the Android Architecture.....	75

8.1 The Android Software Stack	75
8.2 The Linux Kernel	76
8.3 Android Runtime – ART	77
8.4 Android Libraries	77
8.4.1 C/C++ Libraries	78
8.5 Application Framework	78
8.6 Applications	79
8.7 Summary	79
9. The Anatomy of an Android Application	81
9.1 Android Activities	81
9.2 Android Intents.....	82
9.3 Broadcast Intents	82
9.4 Broadcast Receivers	82
9.5 Android Services.....	83
9.6 Content Providers.....	83
9.7 The Application Manifest	84
9.8 Application Resources	84
9.9 Application Context.....	84
9.10 Summary	84
10. Understanding Android Application and Activity Lifecycles	85
10.1 Android Applications and Resource Management.....	85
10.2 Android Process States.....	86
10.2.1 Foreground Process.....	86
10.2.2 Visible Process.....	86
10.2.3 Service Process	87
10.2.4 Background Process	87
10.2.5 Empty Process	87
10.3 Inter-Process Dependencies.....	87
10.4 The Activity Lifecycle	87
10.5 The Activity Stack	87
10.6 Activity States	89
10.7 Configuration Changes	89
10.8 Handling State Change	89
10.9 Summary	90
11. Handling Android Activity State Changes	91
11.1 The Activity Class	91
11.2 Dynamic State vs. Persistent State	94
11.3 The Android Activity Lifecycle Methods.....	95

11.4 Activity Lifetimes	96
11.5 Summary.....	97
12. Android Activity State Changes by Example.....	99
12.1 Creating the State Change Example Project	99
12.2 Designing the User Interface	100
12.3 Overriding the Activity Lifecycle Methods.....	102
12.4 Filtering the LogCat Panel.....	106
12.5 Running the Application	107
12.6 Experimenting with the Activity	108
12.7 Summary.....	109
13. Saving and Restoring the State of an Android Activity	111
13.1 Saving Dynamic State	111
13.2 Default Saving of User Interface State.....	111
13.3 The Bundle Class.....	113
13.4 Saving the State	113
13.5 Restoring the State	115
13.6 Testing the Application	116
13.7 Summary.....	116
14. Understanding Android Views, View Groups and Layouts	117
14.1 Designing for Different Android Devices	117
14.2 Views and View Groups	117
14.3 Android Layout Managers	118
14.4 The View Hierarchy.....	119
14.5 Creating User Interfaces	121
14.6 Summary.....	121
15. A Guide to the Android Studio Designer Tool	123
15.1 Blank vs. Empty Activity Templates	123
15.2 The Android Studio Designer.....	126
15.3 Design Mode.....	126
15.4 Text Mode.....	127
15.5 Setting Properties	128
15.6 Type Morphing	130
15.7 Creating a Custom Device Definition	130
15.8 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.....	139
16.8 Summary	143
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.....	163
18.7 Implementing Cell Row and Column Spanning	164
18.8 Changing the Gravity of a GridLayout Child	164
18.9 Summary	167
19. Working with the Android GridLayout using XML Layout Resources	169
19.1 GridLayouts in XML Resource Files.....	169
19.2 Adding Child Views to the GridLayout.....	170
19.3 Declaring Cell Spanning, Gravity and Margins.....	172
19.4 Summary	174
20. An Overview and Example of Android Event Handling	175
20.1 Understanding Android Events	175
20.2 Using the android:onClick Resource.....	176
20.3 Event Listeners and Callback Methods.....	176
20.4 An Event Handling Example.....	177
20.5 Designing the User Interface	177

20.6 The Event Listener and Callback Method	179
20.7 Consuming Events	181
20.8 Summary.....	183
21. Android Touch and Multi-touch Event Handling	185
21.1 Intercepting Touch Events	185
21.2 The MotionEvent Object.....	186
21.3 Understanding Touch Actions.....	186
21.4 Handling Multiple Touches	186
21.5 An Example Multi-Touch Application	187
21.6 Designing the Activity User Interface	187
21.7 Implementing the Touch Event Listener.....	189
21.8 Running the Example Application.....	193
21.9 Summary.....	193
22. Detecting Common Gestures using the Android Gesture Detector Class.....	195
22.1 Implementing Common Gesture Detection	195
22.2 Creating an Example Gesture Detection Project	196
22.3 Implementing the Listener Class	197
22.4 Creating the GestureDetectorCompat Instance	199
22.5 Implementing the onTouchEvent() Method	200
22.6 Testing the Application	201
22.7 Summary.....	202
23. Implementing Custom Gesture and Pinch Recognition on Android.....	203
23.1 The Android Gesture Builder Application	203
23.2 The GestureOverlayView Class	203
23.3 Detecting Gestures	203
23.4 Identifying Specific Gestures	204
23.5 Building and Running the Gesture Builder Application	204
23.6 Creating a Gestures File	206
23.7 Extracting the Gestures File from the SD Card	208
23.8 Creating the Example Project	208
23.9 Adding the Gestures File to the Project.....	209
23.10 Designing the User Interface	209
23.11 Loading the Gestures File	210
23.12 Registering the Event Listener	211
23.13 Implementing the onGesturePerformed Method	211
23.14 Testing the Application	213
23.15 Configuring the GestureOverlayView	214
23.16 Intercepting Gestures	214

23.17 Detecting Pinch Gestures	215
23.18 A Pinch Gesture Example Project	215
23.19 Summary	217
24. An Introduction to Android Fragments	219
24.1 What is a Fragment?	219
24.2 Creating a Fragment	220
24.3 Adding a Fragment to an Activity using the Layout XML File	221
24.4 Adding and Managing Fragments in Code	223
24.5 Handling Fragment Events	224
24.6 Implementing Fragment Communication	225
24.7 Summary	227
25. Using Fragments in Android Studio - An Example	229
25.1 About the Example Fragment Application	229
25.2 Creating the Example Project	229
25.3 Creating the First Fragment Layout.....	230
25.4 Creating the First Fragment Class.....	232
25.5 Creating the Second Fragment Layout	233
25.6 Adding the Fragments to the Activity.....	235
25.7 Making the Toolbar Fragment Talk to the Activity	238
25.8 Making the Activity Talk to the Text Fragment	242
25.9 Testing the Application.....	244
25.10 Summary	244
26. Creating and Managing Overflow Menus on Android	245
26.1 The Overflow Menu.....	245
26.2 Creating an Overflow Menu	246
26.3 Displaying an Overflow Menu	247
26.4 Responding to Menu Item Selections	248
26.5 Creating Checkable Item Groups.....	248
26.6 Creating the Example Project	250
26.7 Modifying the Menu Description	250
26.8 Modifying the onOptionsItemSelected() Method	251
26.9 Testing the Application.....	253
26.10 Summary	253
27. Animating User Interfaces with the Android Transitions Framework	255
27.1 Introducing Android Transitions and Scenes.....	255
27.2 Using Interpolators with Transitions	256
27.3 Working with Scene Transitions	257
27.4 Custom Transitions and TransitionSets in Code	258

27.5 Custom Transitions and TransitionSets in XML	259
27.6 Working with Interpolators	261
27.7 Creating a Custom Interpolator	263
27.8 Using the beginDelayedTransition Method	264
27.9 Summary.....	264
28. An Android Transition Tutorial using beginDelayedTransition	267
28.1 Creating the Android Studio TransitionDemo Project	267
28.2 Preparing the Project Files.....	267
28.3 Implementing beginDelayedTransition Animation.....	268
28.4 Customizing the Transition	271
28.5 Summary.....	272
29. Implementing Android Scene Transitions – A Tutorial	273
29.1 An Overview of the Scene Transition Project	273
29.2 Creating the Android Studio SceneTransitions Project.....	273
29.3 Identifying and Preparing the Root Container.....	273
29.4 Designing the First Scene	274
29.5 Designing the Second Scene	277
29.6 Entering the First Scene	279
29.7 Loading Scene 2	280
29.8 Implementing the Transitions.....	281
29.9 Adding the Transition File.....	281
29.10 Loading and Using the Transition Set	282
29.11 Configuring Additional Transitions	283
29.12 Summary.....	284
30. Working with the Floating Action Button and Snackbar	285
30.1 The Material Design.....	285
30.2 The Design Library	286
30.3 The Floating Action Button (FAB)	286
30.4 The Snackbar	287
30.5 Creating the Example Project	287
30.6 Reviewing the Project.....	288
30.7 Changing the Floating Action Button.....	290
30.8 Adding the ListView to the Content Layout.....	291
30.9 Adding Items to the ListView.....	292
30.10 Adding an Action to the Snackbar	295
30.11 Summary.....	297
31. Creating a Tabbed Interface using the TabLayout Component.....	299
31.1 An Introduction to the ViewPager	299

31.2 An Overview of the TabLayout Component	299
31.3 Creating the TabLayoutDemo Project	300
31.4 Creating the First Fragment.....	300
31.5 Duplicating the Fragments	302
31.6 Adding the TabLayout and ViewPager	303
31.7 Creating the Pager Adapter.....	304
31.8 Performing the Initialization Tasks	306
31.9 Testing the Application.....	308
31.10 Customizing the TabLayout	309
31.11 Displaying Icon Tab Items	311
31.12 Summary	312
32. Working with the RecyclerView and CardView Widgets	313
32.1 An Overview of the RecyclerView	313
32.2 An Overview of the CardView	316
32.3 Adding the Libraries to the Project	317
32.4 Summary	318
33. An Android RecyclerView and CardView Tutorial.....	319
33.1 Creating the CardDemo Project	319
33.2 Removing the Floating Action Button	319
33.3 Adding the RecyclerView and CardView Libraries.....	320
33.4 Designing the CardView Layout.....	320
33.5 Adding the RecyclerView.....	322
33.6 Creating the RecyclerView Adapter	322
33.7 Adding the Image Files	325
33.8 Initializing the RecyclerView Component.....	326
33.9 Testing the Application.....	327
33.10 Responding to Card Selections	328
33.11 Summary	329
34. Working with the AppBar and Collapsing Toolbar Layouts.....	331
34.1 The Anatomy of an AppBar	331
34.2 The Example Project.....	332
34.3 Coordinating the RecyclerView and Toolbar	333
34.4 Introducing the Collapsing Toolbar Layout	335
34.5 Changing the Title and Scrim Color	338
34.6 Summary	339
35. Implementing an Android Navigation Drawer.....	341
35.1 An Overview of the Navigation Drawer.....	341
35.2 Opening and Closing the Drawer.....	343

35.3 Responding to Drawer Item Selections	343
35.4 Using the Navigation Drawer Activity Template.....	344
35.5 Creating the Navigation Drawer Template Project.....	345
35.6 The Template Layout Resource Files	345
35.7 The Header Coloring Resource File.....	345
35.8 The Template Menu Resource File	345
35.9 The Template Code.....	346
35.10 Running the App	347
35.11 Summary.....	348
36. An Android Studio Master/Detail Flow Tutorial.....	349
36.1 The Master/Detail Flow	349
36.2 Creating a Master/Detail Flow Activity	351
36.3 The Anatomy of the Master/Detail Flow Template	352
36.4 Modifying the Master/Detail Flow Template	353
36.5 Changing the Content Model.....	354
36.6 Changing the Detail Pane.....	355
36.7 Modifying the WebsiteDetailFragment Class	357
36.8 Modifying the WebsiteListActivity Class.....	358
36.9 Adding Manifest Permissions	359
36.10 Running the Application	359
36.11 Summary.....	360
37. An Overview of Android Intents	361
37.1 An Overview of Intents	361
37.2 Explicit Intents	362
37.3 Returning Data from an Activity	363
37.4 Implicit Intents.....	364
37.5 Using Intent Filters.....	365
37.6 Checking Intent Availability	366
37.7 Summary.....	366
38. Android Explicit Intents – A Worked Example.....	367
38.1 Creating the Explicit Intent Example Application	367
38.2 Designing the User Interface Layout for ActivityA.....	367
38.3 Creating the Second Activity Class.....	369
38.4 Designing the User Interface Layout for ActivityB	370
38.5 Reviewing the Application Manifest File	372
38.6 Creating the Intent	373
38.7 Extracting Intent Data.....	374
38.8 Launching ActivityB as a Sub-Activity	375

38.9 Returning Data from a Sub-Activity.....	376
38.10 Testing the Application.....	377
38.11 Summary	377
39. Android Implicit Intents – A Worked Example	379
39.1 Creating the Android Studio Implicit Intent Example Project	379
39.2 Designing the User Interface	379
39.3 Creating the Implicit Intent	381
39.4 Adding a Second Matching Activity	382
39.5 Adding the Web View to the UI.....	382
39.6 Obtaining the Intent URL.....	383
39.7 Modifying the MyWebView Project Manifest File	385
39.8 Installing the MyWebView Package on a Device	386
39.9 Testing the Application.....	387
39.10 Summary	388
40. Android Broadcast Intents and Broadcast Receivers	389
40.1 An Overview of Broadcast Intents.....	389
40.2 An Overview of Broadcast Receivers.....	390
40.3 Obtaining Results from a Broadcast	392
40.4 Sticky Broadcast Intents	392
40.5 The Broadcast Intent Example	393
40.6 Creating the Example Application	393
40.7 Creating and Sending the Broadcast Intent.....	393
40.8 Creating the Broadcast Receiver	394
40.9 Configuring a Broadcast Receiver in the Manifest File.....	396
40.10 Testing the Broadcast Example	397
40.11 Listening for System Broadcasts.....	397
40.12 Summary	398
41. A Basic Overview of Android Threads and Thread Handlers.....	401
41.1 An Overview of Threads	401
41.2 The Application Main Thread	401
41.3 Thread Handlers	401
41.4 A Basic Threading Example.....	402
41.5 Creating a New Thread.....	405
41.6 Implementing a Thread Handler	406
41.7 Passing a Message to the Handler.....	408
41.8 Summary	410
42. An Overview of Android Started and Bound Services.....	411
42.1 Started Services	411

42.2 Intent Service.....	412
42.3 Bound Service	412
42.4 The Anatomy of a Service	413
42.5 Controlling Destroyed Service Restart Options	414
42.6 Declaring a Service in the Manifest File.....	414
42.7 Starting a Service Running on System Startup.....	415
42.8 Summary.....	415
43. Implementing an Android Started Service – A Worked Example.....	417
43.1 Creating the Example Project	417
43.2 Creating the Service Class.....	417
43.3 Adding the Service to the Manifest File.....	419
43.4 Starting the Service.....	420
43.5 Testing the IntentService Example	420
43.6 Using the Service Class	421
43.7 Creating the New Service	421
43.8 Modifying the User Interface.....	423
43.9 Running the Application	425
43.10 Creating a New Thread for Service Tasks	425
43.11 Summary.....	427
44. Android Local Bound Services – A Worked Example	429
44.1 Understanding Bound Services.....	429
44.2 Bound Service Interaction Options.....	429
44.3 An Android Studio Local Bound Service Example	430
44.4 Adding a Bound Service to the Project	430
44.5 Implementing the Binder.....	431
44.6 Binding the Client to the Service	434
44.7 Completing the Example.....	435
44.8 Testing the Application	438
44.9 Summary.....	438
45. Android Remote Bound Services – A Worked Example.....	439
45.1 Client to Remote Service Communication	439
45.2 Creating the Example Application	439
45.3 Designing the User Interface	440
45.4 Implementing the Remote Bound Service	440
45.5 Configuring a Remote Service in the Manifest File.....	442
45.6 Launching and Binding to the Remote Service	443
45.7 Sending a Message to the Remote Service.....	445
45.8 Summary.....	445

46. An Overview of Android SQLite Databases	447
46.1 Understanding Database Tables.....	447
46.2 Introducing Database Schema.....	448
46.3 Columns and Data Types	448
46.4 Database Rows	448
46.5 Introducing Primary Keys	448
46.6 What is SQLite?	449
46.7 Structured Query Language (SQL)	449
46.8 Trying SQLite on an Android Virtual Device (AVD)	450
46.9 Android SQLite Java Classes	452
46.9.1 <i>Cursor</i>	452
46.9.2 <i>SQLiteDatabase</i>	453
46.9.3 <i>SQLiteOpenHelper</i>	453
46.9.4 <i>ContentValues</i>	454
46.10 Summary	454
47. An Android TableLayout and TableRow Tutorial	455
47.1 The TableLayout and TableRow Layout Views	455
47.2 Creating the Database Project.....	457
47.3 Adding the TableLayout to the User Interface	457
47.4 Adding and Configuring the TableRows	458
47.5 Adding the Button Bar to the Layout	459
47.6 Adjusting the Layout Margins.....	461
47.7 Summary	463
48. An Android SQLite Database Tutorial.....	465
48.1 About the Database Example	465
48.2 Creating the Data Model	466
48.3 Implementing the Data Handler.....	467
48.3.1 <i>The Add Handler Method</i>	470
48.3.2 <i>The Query Handler Method</i>	470
48.3.3 <i>The Delete Handler Method</i>	471
48.4 Implementing the Activity Event Methods	471
48.5 Testing the Application.....	474
48.6 Summary	474
49. Understanding Android Content Providers	475
49.1 What is a Content Provider?	475
49.2 The Content Provider	475
49.2.1 <i>onCreate()</i>	476
49.2.2 <i>query()</i>	476

49.2.3 <i>insert()</i>	476
49.2.4 <i>update()</i>	476
49.2.5 <i>delete()</i>	476
49.2.6 <i>getType()</i>	476
49.3 The Content URI	476
49.4 The Content Resolver	477
49.5 The <provider> Manifest Element	477
49.6 Summary	478
50. Implementing an Android Content Provider in Android Studio	479
50.1 Copying the Database Project	479
50.2 Adding the Content Provider Package	479
50.3 Creating the Content Provider Class	480
50.4 Constructing the Authority and Content URI	482
50.5 Implementing URI Matching in the Content Provider	483
50.6 Implementing the Content Provider onCreate() Method	485
50.7 Implementing the Content Provider insert() Method	485
50.8 Implementing the Content Provider query() Method	486
50.9 Implementing the Content Provider update() Method	488
50.10 Implementing the Content Provider delete() Method	489
50.11 Declaring the Content Provider in the Manifest File	491
50.12 Modifying the Database Handler	492
50.13 Summary	494
51. Accessing Cloud Storage using the Android Storage Access Framework	495
51.1 The Storage Access Framework	495
51.2 Working with the Storage Access Framework	497
51.3 Filtering Picker File Listings	497
51.4 Handling Intent Results	499
51.5 Reading the Content of a File	499
51.6 Writing Content to a File	500
51.7 Deleting a File	501
51.8 Gaining Persistent Access to a File	501
51.9 Summary	502
52. An Android Storage Access Framework Example	503
52.1 About the Storage Access Framework Example	503
52.2 Creating the Storage Access Framework Example	503
52.3 Designing the User Interface	504
52.4 Declaring Request Codes	506
52.5 Creating a New Storage File	507

52.6 The onActivityResult() Method	508
52.7 Saving to a Storage File	510
52.8 Opening and Reading a Storage File	513
52.9 Testing the Storage Access Application	515
52.10 Summary	516
53. Implementing Video Playback on Android using the VideoView and MediaController Classes	517
53.1 Introducing the Android VideoView Class	517
53.2 Introducing the Android MediaController Class	518
53.3 Testing Video Playback	518
53.4 Creating the Video Playback Example	519
53.5 Designing the VideoPlayer Layout	519
53.6 Configuring the VideoView	520
53.7 Adding Internet Permission	521
53.8 Adding the MediaController to the Video View	522
53.9 Setting up the onPreparedListener	523
53.10 Summary	525
54. Video Recording and Image Capture on Android using Camera Intents	527
54.1 Checking for Camera Support	527
54.2 Calling the Video Capture Intent	528
54.3 Calling the Image Capture Intent	529
54.4 Creating an Android Studio Video Recording Project	530
54.5 Designing the User Interface Layout	530
54.6 Checking for the Camera	532
54.7 Launching the Video Capture Intent	533
54.8 Handling the Intent Return	534
54.9 Testing the Application	535
54.10 Summary	535
55. Making Runtime Permission Requests in Android 6.0	537
55.1 Understanding Normal and Dangerous Permissions	537
55.2 Creating the Permissions Example Project	539
55.3 Checking for a Permission	539
55.4 Requesting Permission at Runtime	541
55.5 Providing a Rationale for the Permission Request	543
55.6 Testing the Permissions App	545
55.7 Summary	546
56. Android Audio Recording and Playback using MediaPlayer and MediaRecorder	547
56.1 Playing Audio	547
56.2 Recording Audio and Video using the MediaRecorder Class	548

56.3 About the Example Project	549
56.4 Creating the AudioApp Project	550
56.5 Designing the User Interface	550
56.6 Checking for Microphone Availability	551
56.7 Performing the Activity Initialization	552
56.8 Implementing the recordAudio() Method	553
56.9 Implementing the stopAudio() Method	554
56.10 Implementing the playAudio() method	555
56.11 Configuring and Requesting Permissions	555
56.12 Testing the Application	559
56.13 Summary	559
57. Working with the Google Maps Android API in Android Studio	561
57.1 The Elements of the Google Maps Android API	561
57.2 Creating the Google Maps Project	562
57.3 Obtaining Your Developer Signature	562
57.4 Testing the Application	564
57.5 Understanding Geocoding and Reverse Geocoding	564
57.6 Adding a Map to an Application	567
57.7 Requesting Current Location Permission	567
57.8 Displaying the User's Current Location	570
57.9 Changing the Map Type	570
57.10 Displaying Map Controls to the User	571
57.11 Handling Map Gesture Interaction	572
57.11.1 Map Zooming Gestures	572
57.11.2 Map Scrolling/Panning Gestures	572
57.11.3 Map Tilt Gestures	573
57.11.4 Map Rotation Gestures	573
57.12 Creating Map Markers	573
57.13 Controlling the Map Camera	575
57.14 Summary	576
58. Printing with the Android Printing Framework	577
58.1 The Android Printing Architecture	577
58.2 The Print Service Plugins	577
58.3 Google Cloud Print	578
58.4 Printing to Google Drive	579
58.5 Save as PDF	579
58.6 Printing from Android Devices	579
58.7 Options for Building Print Support into Android Apps	581
58.7.1 Image Printing	581

58.7.2 Creating and Printing HTML Content	582
58.7.3 Printing a Web Page	584
58.7.4 Printing a Custom Document	585
58.8 Summary	585
59. An Android HTML and Web Content Printing Example	587
59.1 Creating the HTML Printing Example Application	587
59.2 Printing Dynamic HTML Content	587
59.3 Creating the Web Page Printing Example.....	591
59.4 Removing the Floating Action Button	591
59.5 Designing the User Interface Layout	592
59.6 Loading the Web Page into the WebView.....	594
59.7 Adding the Print Menu Option	595
59.8 Summary	598
60. A Guide to Android Custom Document Printing	599
60.1 An Overview of Android Custom Document Printing.....	599
60.1.1 Custom Print Adapters	600
60.2 Preparing the Custom Document Printing Project	601
60.3 Creating the Custom Print Adapter	602
60.4 Implementing the onLayout() Callback Method	604
60.5 Implementing the onWrite() Callback Method	607
60.6 Checking a Page is in Range.....	610
60.7 Drawing the Content on the Page Canvas	611
60.8 Starting the Print Job.....	614
60.9 Testing the Application.....	615
60.10 Summary	616
61. Handling Different Android Devices and Displays	617
61.1 Handling Different Device Displays	617
61.2 Creating a Layout for each Display Size	617
61.3 Providing Different Images.....	618
61.4 Checking for Hardware Support	619
61.5 Providing Device Specific Application Binaries.....	620
61.6 Summary	620
62. Signing and Preparing an Android Application for Release	621
62.1 The Release Preparation Process	621
62.2 Changing the Build Variant.....	621
62.3 Enabling ProGuard.....	622
62.4 Creating a Keystore File.....	623
62.5 Generating a Private Key	624

62.6 Creating the Application APK File	625
62.7 Register for a Google Play Developer Console Account	627
62.8 Uploading New APK Versions to the Google Play Developer Console.....	627
62.9 Summary.....	629
63. Integrating Google Play In-app Billing into an Android Application.....	631
63.1 Installing the Google Play Billing Library.....	631
63.2 Creating the Example In-app Billing Project	632
63.3 Adding Billing Permission to the Manifest File	633
63.4 Adding the IInAppBillingService.aidl File to the Project	633
63.5 Adding the Utility Classes to the Project	635
63.6 Designing the User Interface	636
63.7 Implementing the “Click Me” Button	637
63.8 Google Play Developer Console and Google Wallet Accounts	638
63.9 Obtaining the Public License Key for the Application	639
63.10 Setting Up Google Play Billing in the Application	640
63.11 Initiating a Google Play In-app Billing Purchase	642
63.12 Implementing the onActivityResult Method	643
63.13 Implementing the Purchase Finished Listener	643
63.14 Consuming the Purchased Item.....	644
63.15 Releasing the IabHelper Instance	645
63.16 Modifying the Security.java File	646
63.17 Testing the In-app Billing Application	647
63.18 Building a Release APK.....	648
63.19 Creating a New In-app Product.....	649
63.20 Publishing the Application to the Alpha Distribution Channel	650
63.21 Adding In-app Billing Test Accounts	650
63.22 Configuring Group Testing.....	651
63.23 Resolving Problems with In-App Purchasing	652
63.24 Summary.....	653
64. An Overview of Gradle in Android Studio	655
64.1 An Overview of Gradle.....	655
64.2 Gradle and Android Studio	655
64.2.1 Sensible Defaults	656
64.2.2 Dependencies	656
64.2.3 Build Variants	656
64.2.4 Manifest Entries	657
64.2.5 APK Signing	657
64.2.6 ProGuard Support.....	657
64.3 The Top-level Gradle Build File	657

64.4 Module Level Gradle Build Files	658
64.5 Configuring Signing Settings in the Build File	661
64.6 Running Gradle Tasks from the Command-line.....	662
64.7 Summary	663
65. An Android Studio Gradle Build Variants Example	665
65.1 Creating the Build Variant Example Project	665
65.2 Extracting the Hello World String Resource	666
65.3 Adding the Build Flavors to the Module Build File	666
65.4 Adding the Flavors to the Project Structure	669
65.5 Adding Resource Files to the Flavors.....	670
65.6 Testing the Build Flavors	671
65.7 Build Variants and Class Files	671
65.8 Adding Packages to the Build Flavors.....	671
65.9 Customizing the Activity Classes	672
65.10 Summary	673
Index.....	675

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 6 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 printing, transitions and cloud-based file storage.

The concepts of material design are also covered in detail, including the use of floating action buttons, Snackbars, tabbed interfaces, card views, navigation drawers and collapsing toolbars.

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/androidstudioA6/index.php>

The steps to load a project from the code samples into Android Studio are as follows:

1. From the *Welcome to Android Studio* dialog, select the *Open an existing Android Studio* project option.
2. In the project selection dialog, navigate to and select the folder containing the project to be imported and click on OK.

1.2 Download the eBook

Thank you for purchasing the print edition of this book. If you would like to download the eBook version of this book, please email proof of purchase to feedback@ebookfrenzy.com and we will provide you with a download link for the book in PDF, ePub and MOBI formats.

1.3 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.

1.4 Errata

While 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/androidstudioA6.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 Vista (32-bit or 64-bit)
- Windows 7 (32-bit or 64-bit)
- Windows 8 / Windows 8.1 or later
- Mac OS X 10.8.5 or later (Intel based systems only)
- Linux systems with version 2.15 or later of GNU C Library (glibc)
- Minimum of 2GB of RAM (4GB is preferred)
- Approximately 4.5GB of available disk space

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 version 7 of the Standard Edition of the Java Platform Development Kit. 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/jdk7-downloads-1880260.html>

Assuming that a suitable JDK is not already installed on your system, download version 7 of the 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

Java is not installed by default on recent versions of Mac OS X. To confirm the presence or otherwise of Java, 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 "1.7.0_79-b15"  
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)  
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, 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 a More Info button which, when clicked will display the Oracle Java web page:

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

On the Oracle Java web page, locate and download the Java SE 7 JDK installation package for Mac OS X.

Open the downloaded disk image (.dmg file) and double-click on the icon to install the Java package (Figure 2-1):



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

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

```
sudo apt-get install lib32stdc++6
```

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/jdk7-downloads-1880260.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-7u79-linux-x64.rpm*, the commands to perform the installation would read as follows:

```
su  
rpm -ihv jdk-7u79-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*).

Setting up an Android Studio Development Environment

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_79* the following would need to be added to your *\$PATH* environment variable:

```
/home/demo/java/jdk1.7.0_79/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_79/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. The content and examples in this book were created based on Android Studio version 1.5.

Android Studio is subject to frequent updates and it is possible, therefore, that a more recent release of Android Studio is now available. For the purposes of compatibility with the tutorials and examples, however, it is recommended that this book be used with Android Studio version 1.5 which may be downloaded from the following web page:

<http://tools.android.com/download/studio/builds/1-5>

From this page, 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. When prompted to select the components to install, make sure that the *Android Studio*, *Android SDK* and *Android Virtual Device* options are all selected.

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* and that the Android SDK packages have been installed into the user's *AppData\Local\Android\sdk* sub-folder. 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. The executable may 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-ide-<version>.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 folder. 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. When attempting to launch Android Studio, an error dialog may appear indicating that the JVM cannot be found. If this error occurs, it will be necessary to download and install the Mac OS X Java 6 JRE package on the system. This can be downloaded from Apple using the following link:

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

Once the Java for OS X package has been installed, Android Studio should launch without any problems.

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:

```
unzip /<path to package>/android-studio-ide-<version>-linux.zip
```

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
```

On Linux it may also be necessary to specify the location of the Java Development Kit using the following steps:

1. Launch Android Studio and create a new project.
2. Select the *File -> Other Settings -> Default Project Structure...* menu option.
3. Enter the full path to the directory containing the JDK into the *JDK Location* field.
4. Click *Apply* followed by *OK*.

2.6 The Android Studio Setup Wizard

The first time that Android Studio is launched after being installed, a dialog will appear providing the option to import settings from a previous Android Studio version. If you have settings from a previous version and would like to import them into the latest installation, select the appropriate option and location. Alternatively, indicate that you do not need to import any previous settings and click on the OK button to proceed.

Next, the setup wizard may appear as shown in Figure 2-3 though this dialog does not appear on all platforms:

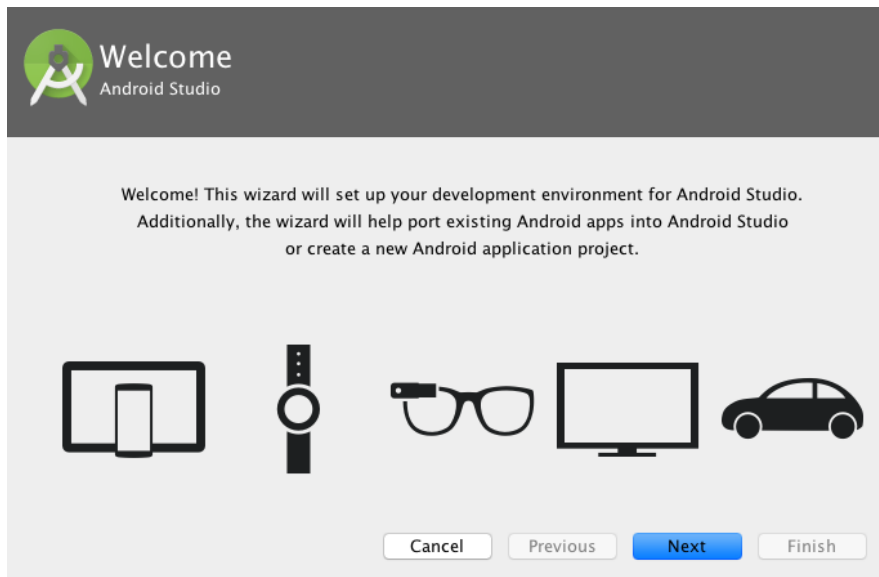


Figure 2-3

Setting up an Android Studio Development Environment

If the wizard appears, click on the Next button, choose the Standard installation option and click on Next once again.

Android Studio will proceed to download and configure the latest Android SDK and some additional components and packages. Once this process has completed, click on the *Finish* button in the *Downloading Components* dialog at which point the Welcome to Android Studio screen should then appear:

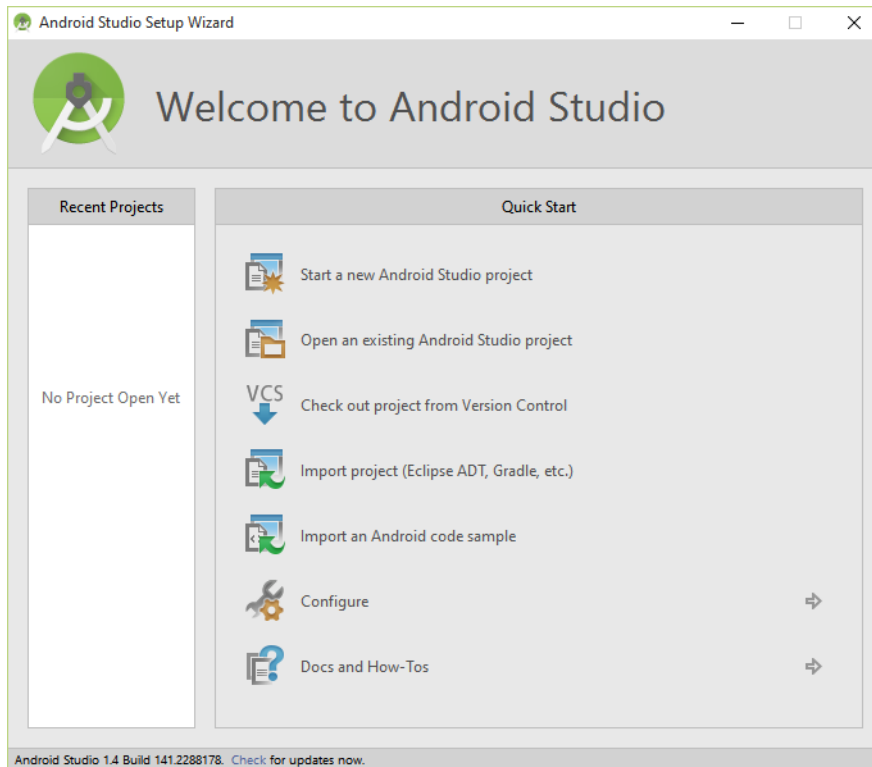


Figure 2-4

2.7 Installing Additional 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 or updated packages.

This task can be performed using the *Android SDK Settings* screen, 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 *Android SDK* screen of the default settings dialog will appear as shown in Figure 2-5:

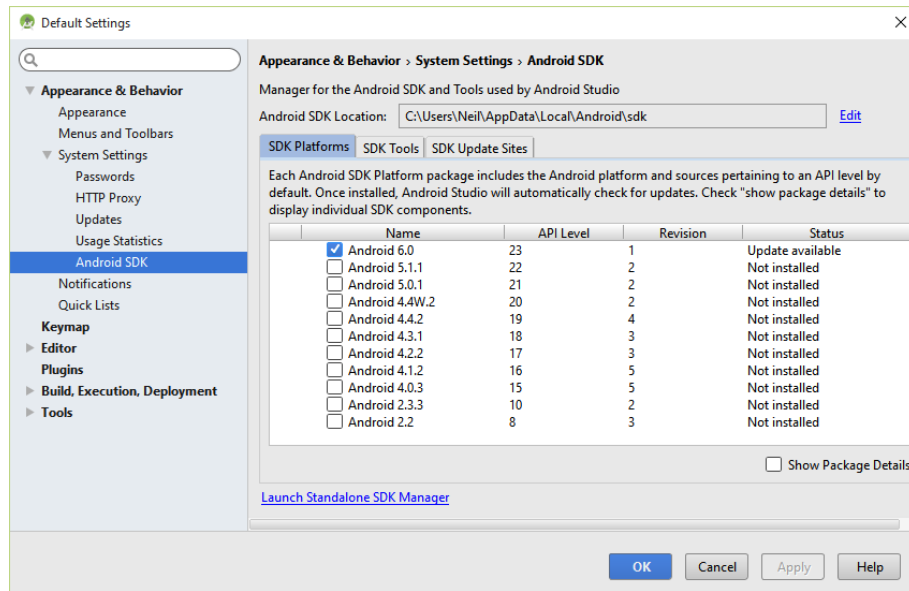


Figure 2-5

Immediately after installing Android Studio for the first time it is likely that only the latest version of the Android SDK has been installed. To install older versions of the Android SDK simply select the checkboxes corresponding to the versions and click on the *Apply* button.

It is also possible that updates will be listed as being available for the latest SDK. To access detailed information about the packages that are available for update, enable the *Show Package Details* option located in the lower right hand corner of the screen. This will display information similar to that shown in Figure 2-6:

Name	API Level	Revision	Status
Android 6.0			
<input checked="" type="checkbox"/> Android 6.0 Platform	23	1	Installed
<input type="checkbox"/> Android TV ARM EABI v7a System Image	23	2	Not installed
<input type="checkbox"/> Android TV Intel x86 Atom System Image	23	2	Not installed
<input type="checkbox"/> ARM EABI v7a System Image	23	3	Not installed
<input type="checkbox"/> Intel x86 Atom System Image	23	4	Not installed
<input type="checkbox"/> Intel x86 Atom_64 System Image	23	4	Not installed
<input type="checkbox"/> Google APIs, Android 23	23	1	Update Available: 1
<input type="checkbox"/> Google APIs ARM EABI v7a System Image	23	7	Not installed
<input checked="" type="checkbox"/> Google APIs Intel x86 Atom System Image	23	8	Installed
<input type="checkbox"/> Google APIs Intel x86 Atom_64 System Image	23	8	Not installed
<input checked="" type="checkbox"/> Sources for Android 23	23	1	Installed

Figure 2-6

The above figure highlights the availability of an update. To install the updates, enable the checkbox to the left of the item name and click on the *Apply* button.

Setting up an Android Studio Development Environment

In addition to the Android SDK packages, a number of tools are also installed for building Android applications. To view the currently installed packages and check for updates, remain within the SDK settings screen and select the SDK Tools tab as shown in Figure 2-7:

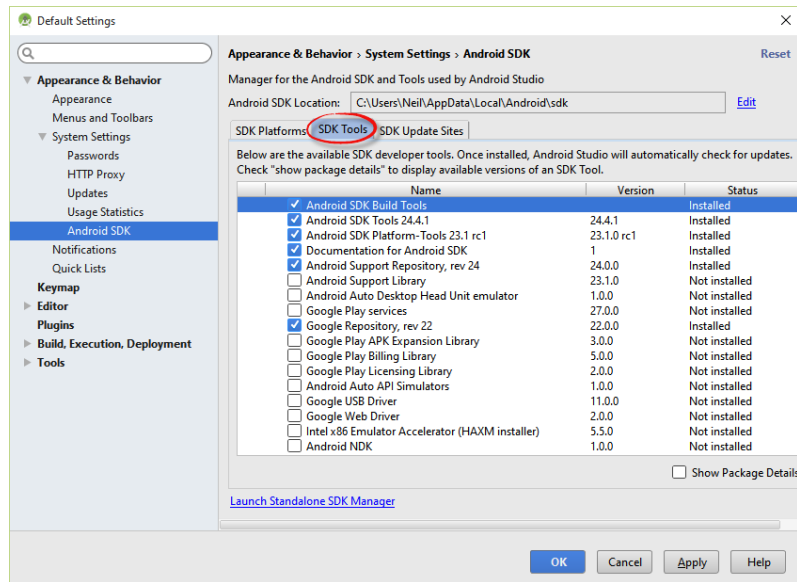


Figure 2-7

Within the Android SDK Tools screen, make sure that the following packages are listed as *Installed* in the Status column:

- Android SDK Build-tools
- Android SDK Tools
- Android SDK Platform-tools
- Android Support Repository
- Android Support Library
- Google Repository
- Google USB Driver (Windows only)
- Intel x86 Emulator Accelerator (HAXM installer)

In the event that any of the above packages are listed as *Not Installed* or requiring an update, simply select the checkboxes next to those packages and click on the *Apply* button to initiate the installation process.

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.

An alternative to using the Android SDK settings panel is to access the *Standalone SDK Manager* which can be launched using the link in the lower left hand corner of the settings screen. The Standalone SDK Manager (Figure 2-8) provides a similar list of packages together with options to perform update and installation tasks:

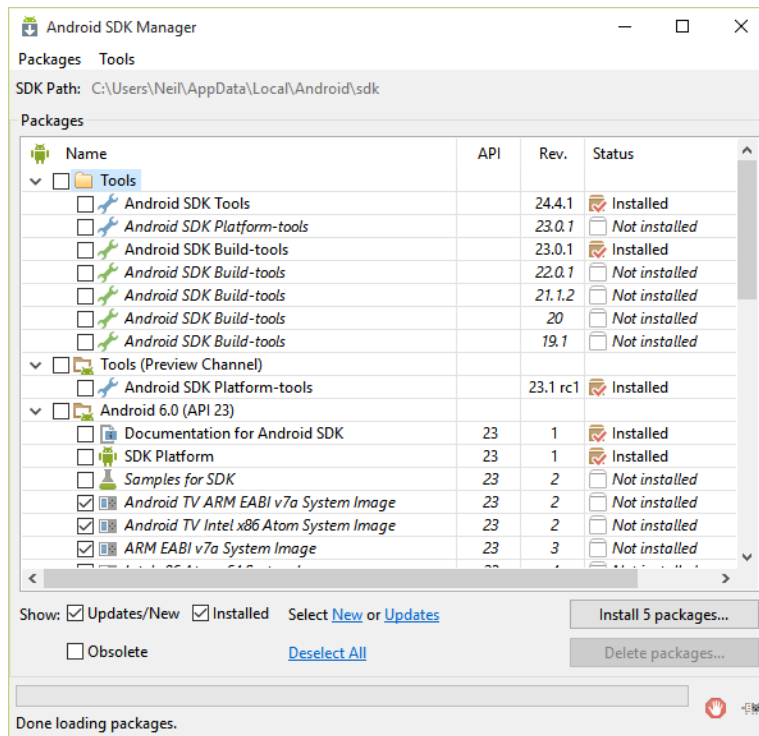


Figure 2-8

2.8 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_sdk_installation>* represents the file system location into which the Android SDK was installed):

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

Setting up an Android Studio Development Environment

The location of the SDK on your system can be identified by launching the Standalone SDK Manager and referring to the *Android SDK Location*: field located at the top of the settings panel as highlighted in Figure 2-9:

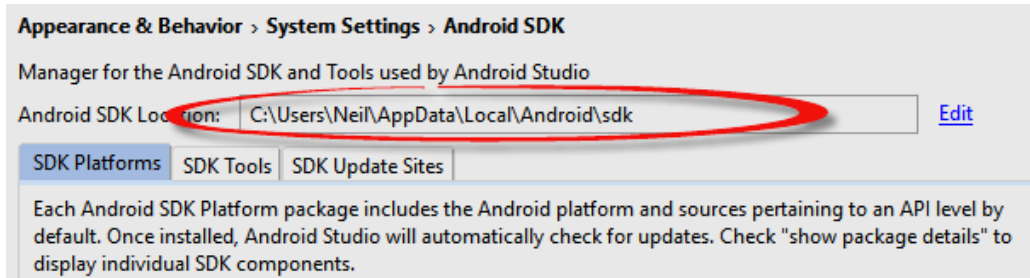


Figure 2-9

Once the location of the SDK has been identified, the steps to add this to the PATH variable are operating system dependent:

2.8.1 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 Android SDK was installed into *C:\Users\demo\AppData\Local\Android\sdk*, the following would be appended to the end of the current Path value:

```
;C:\Users\demo\AppData\Local\Android\sdk\platform-  
tools;C:\Users\demo\AppData\Local\Android\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.
```

2.8.2 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.

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.
```

2.8.3 Windows 10

Right-click on the Start menu, select *System* from the resulting menu and click on the *Advanced system settings* option in the System window. Follow the steps outlined for Windows 7 starting from step 2 through to step 4.

2.8.4 Linux

On Linux this will involve once again editing the *.bashrc* file. Assuming that the Android SDK bundle package was installed into */home/demo/Android/sdk*, 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/sdk/platform-  
tools:/home/demo/Android/sdk/tools:/home/demo/android-studio/bin:$PATH
```

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

2.8.5 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 Android SDK installation location of */Users/demo/Library/Android/sdk*, 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/Library/Android/sdk/tools  
/Users/demo/Library/Android/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
```

2.9 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.

2.10 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, 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:

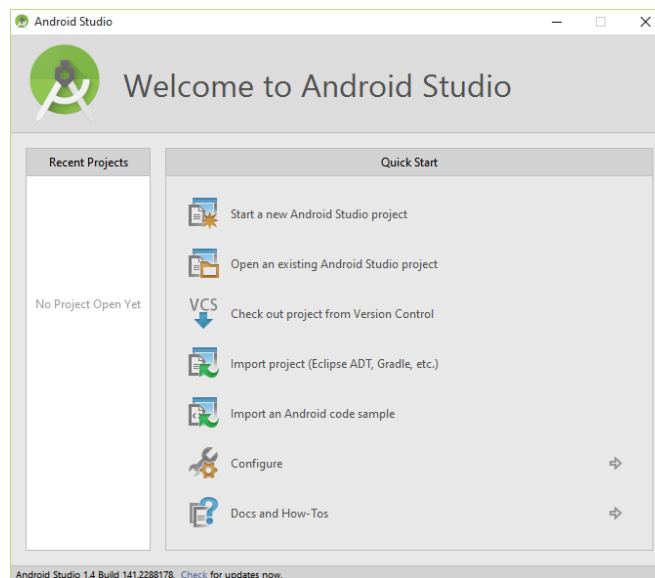


Figure 3-1

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 *Start a new Android Studio project* option to display the first screen of the *New Project* wizard as shown in Figure 3-2:

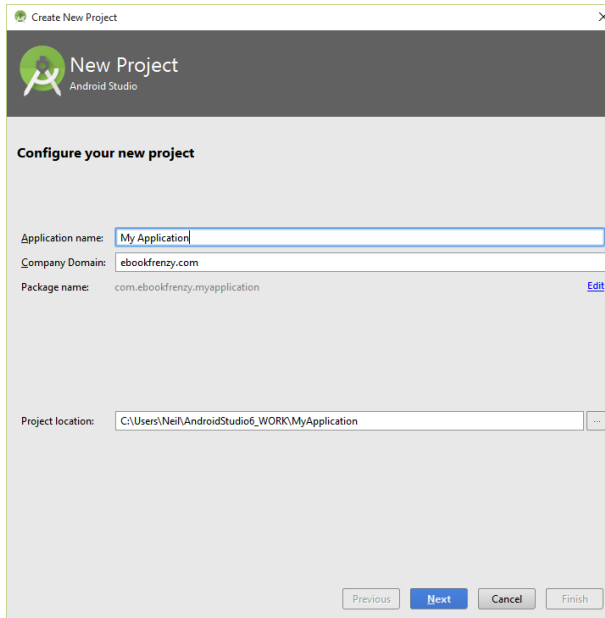


Figure 3-2

3.2 Defining the Project and SDK Settings

In the *New Project* window, set 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). The reason for selecting an older SDK release is that this ensures that the finished application will be able to run on the widest possible range of Android devices. The higher the minimum SDK selection, the more the application will be restricted to newer Android devices. A useful chart (Figure 3-3) can be viewed by clicking on the *Help me choose* link. This outlines the various SDK versions and API levels available for use and the percentage of Android devices in the marketplace on which the application will run if that SDK is used as the minimum level. In general it should only be necessary to select a more recent SDK when that release contains a specific feature that is required for your application. To help in the decision process, selecting an API level from the chart will display the features that are supported at that level.

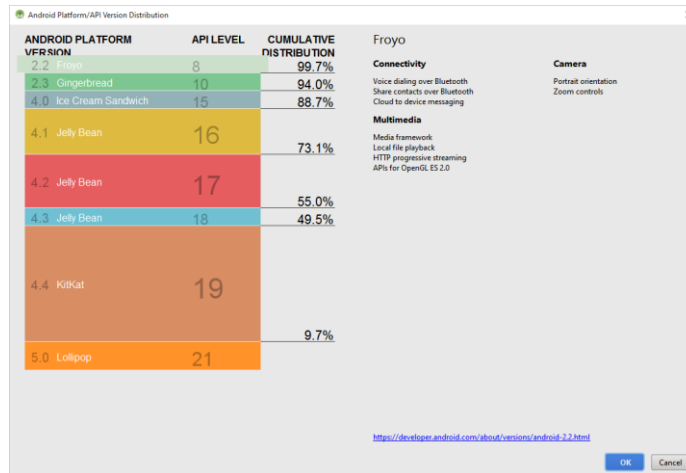


Figure 3-3

Since the project is not intended for Google TV, Android Auto or wearable devices, leave the remaining options disabled before clicking *Next*.

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 *Empty*, *Master/Detail Flow*, *Google Maps* and *Navigation Drawer* options will be covered extensively in later chapters. For the purposes of this example, however, simply select the option to create a *Blank Activity*. The blank activity option creates a template user interface consisting of an app bar, menu, content area and a single floating action button.

Creating an Example Android App in Android Studio

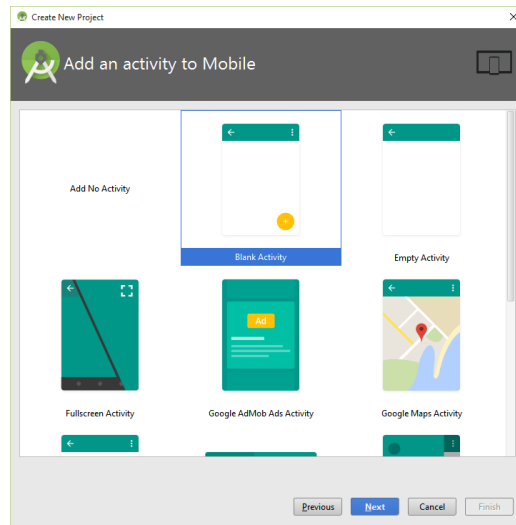


Figure 3-4

With the Blank Activity option selected, click *Next*. On the final screen (Figure 3-5) 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 3-5 and with a menu resource named *menu_android_sample*:

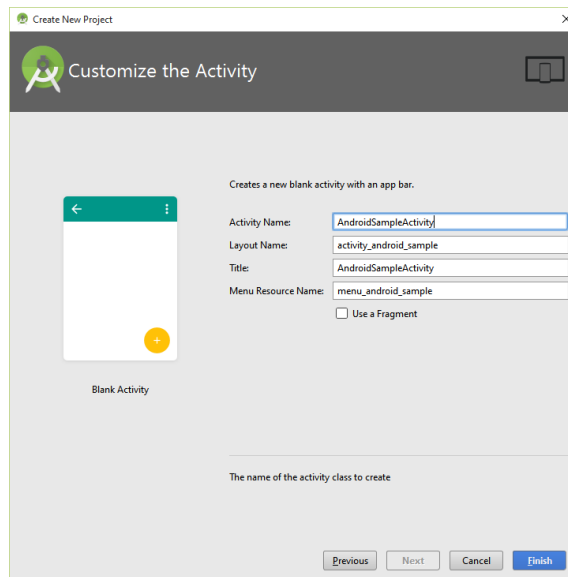


Figure 3-5

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.

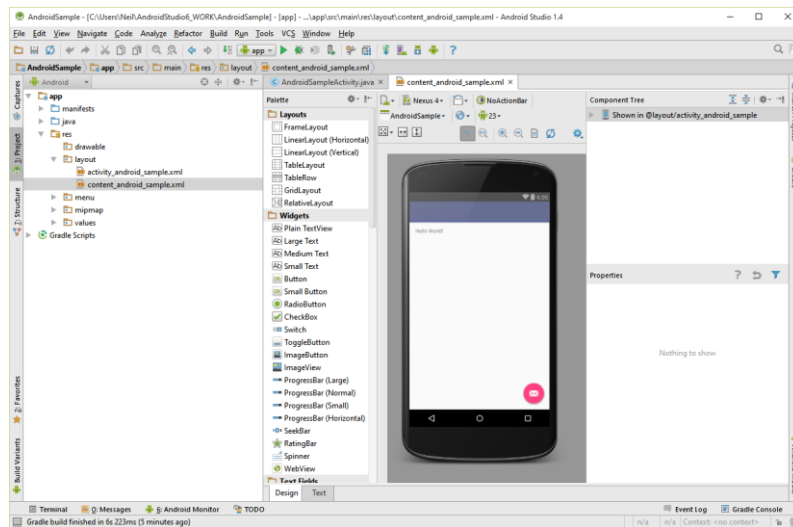


Figure 3-6

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. The Project tool window has a number of modes in which information can be displayed. By default, this panel will be in *Android* mode. This setting is controlled by the drop down menu at the top of the panel as highlighted in Figure 3-6. If the panel is not currently in Android mode, click on this menu and switch to Android mode:

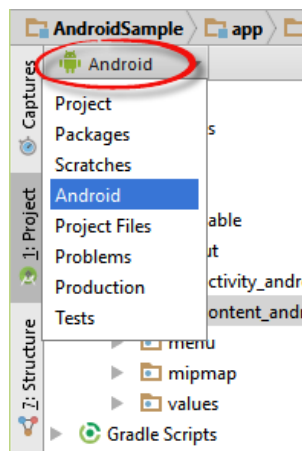


Figure 3-7

Creating an Example Android App in Android Studio

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 *app -> res -> layout* in the project file hierarchy. This layout file includes the app bar (also known as an action bar) that appears across the top of the device screen (marked A in Figure 3-8) and the floating action button (the email button marked B). In addition to these items, the *activity_android_sample.xml* layout file contains a reference to a second file containing the content layout (marked C):

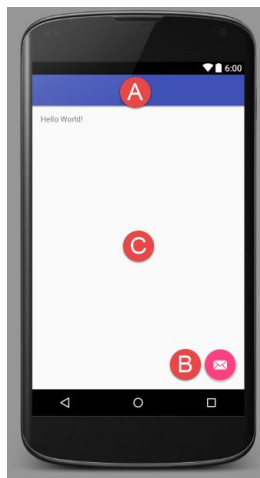


Figure 3-8

By default, the content layout is contained within a file named *content_android_studio.xml* and it is within this file that changes to the layout of the activity are made. Using the Project tool window, locate this file as illustrated in Figure 3-9:

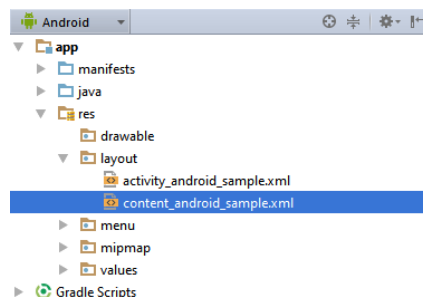


Figure 3-9

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:

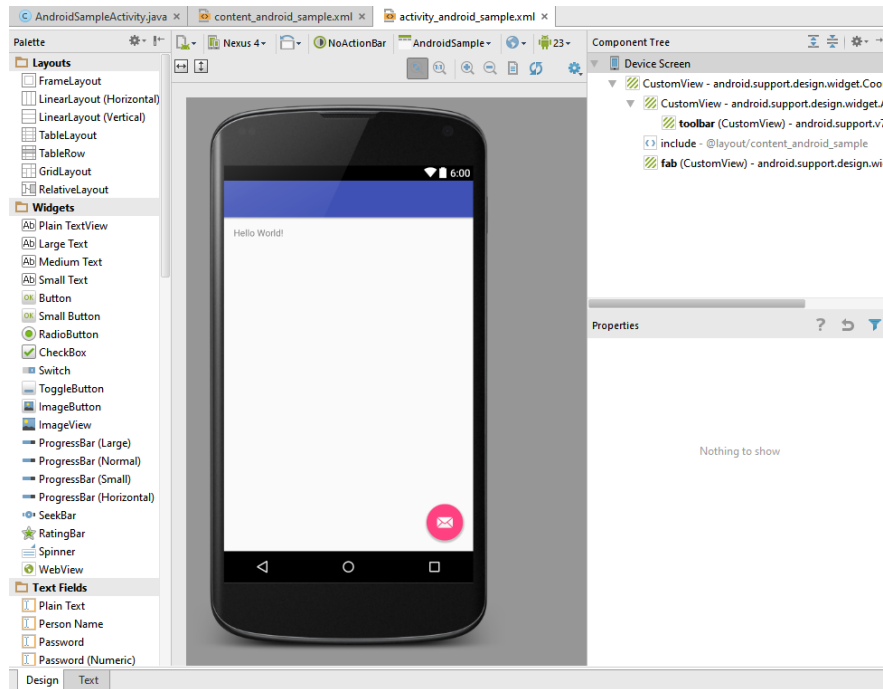



Figure 3-10

In the toolbar across the top of the 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 content 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 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 3-11:

Creating an Example Android App in Android Studio

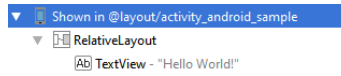


Figure 3-11

As we can see from the component tree hierarchy, the user interface layout is embedded in the *activity_android_sample.xml* layout file and 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:

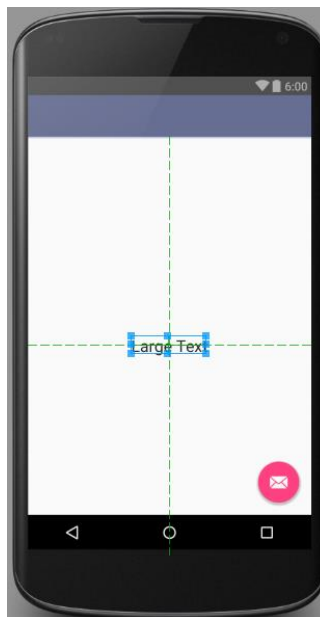


Figure 3-12

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 3-13. Within the panel, change the text property from “Large Text” to “Welcome to Android Studio”.

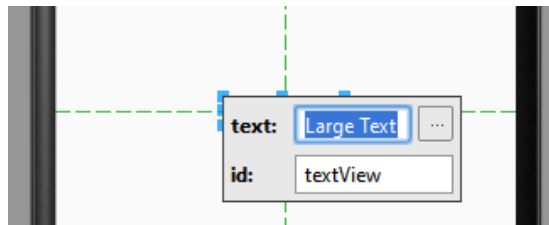


Figure 3-13

At this point it is important to explain the light bulb that has now appeared 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 3-14).

Creating an Example Android App in Android Studio

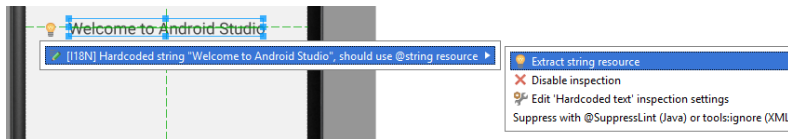


Figure 3-14

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 *app -> res -> values -> strings.xml* file. If the layout displays the name of the string resource instead of the “Welcome to Android Studio” text, click on the refresh button located in the toolbar as highlighted in Figure 3-15:

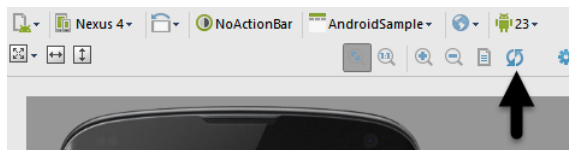


Figure 3-15

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 *content_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 3-16:

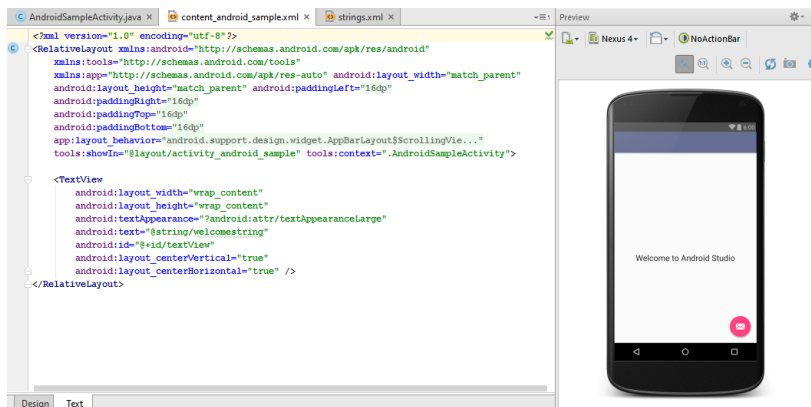


Figure 3-16

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. If the panel is not visible, display it by selecting the *Preview* button located along the right hand edge of the Android Studio window. 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:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    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"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_android_sample"
    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.

Creating an Example Android App in Android Studio

Finally, use the Project view to locate the *app* -> *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="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 tool 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 3-17.

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

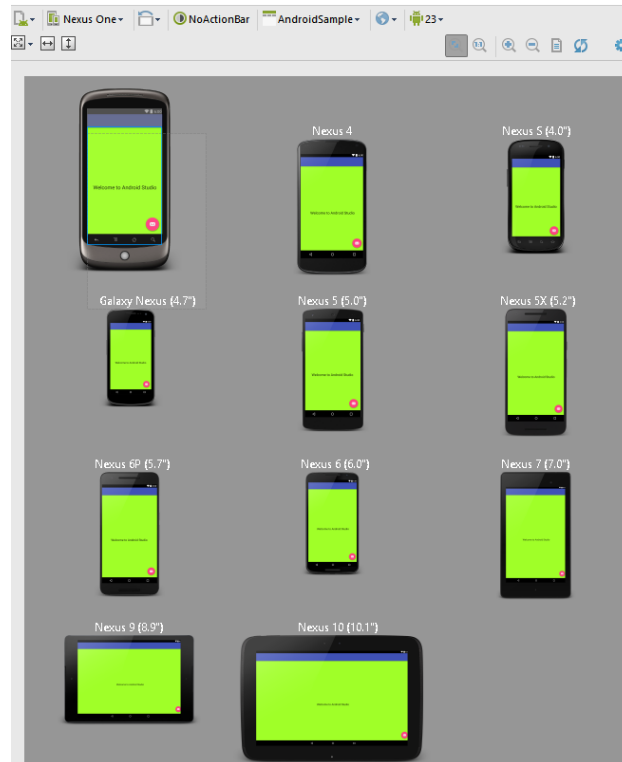


Figure 3-17

3.7 Summary

While 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 Designer tool 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.

While 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.

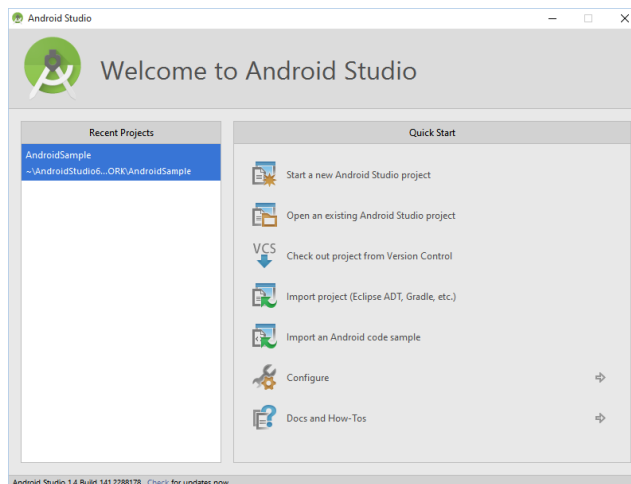


Figure 4-1

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.

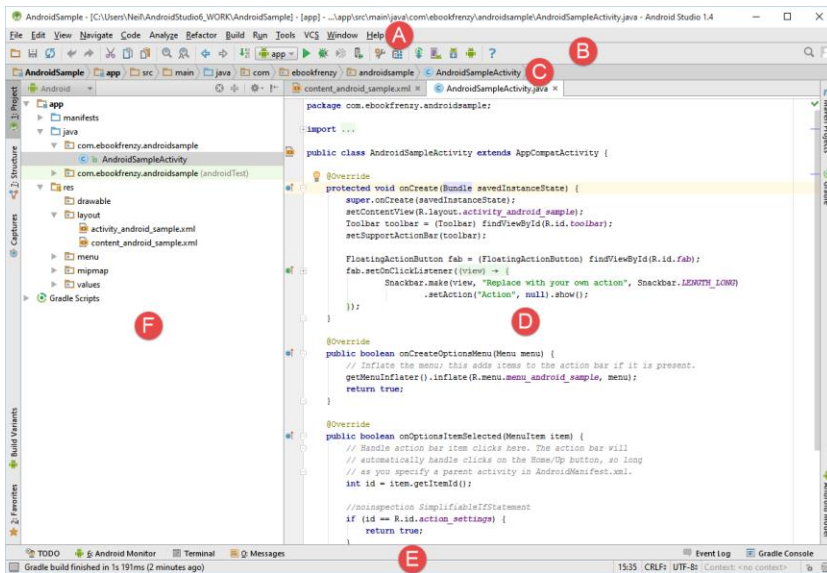


Figure 4-2

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.

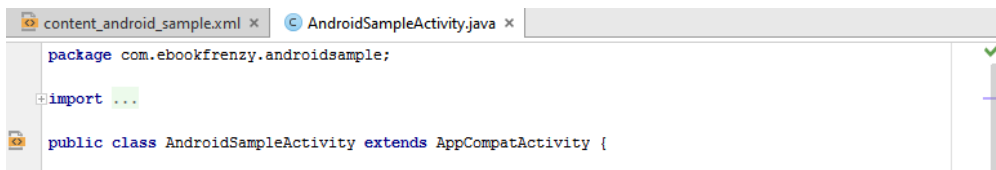


Figure 4-3

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 drop-down menu in the toolbar can be used to display the project in a number of different ways. The default setting is the *Android* view which is the mode primarily used in the remainder of this book.

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 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.

A Tour of the Android Studio User Interface

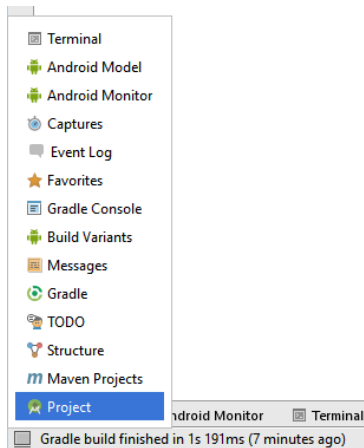


Figure 4-4

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.

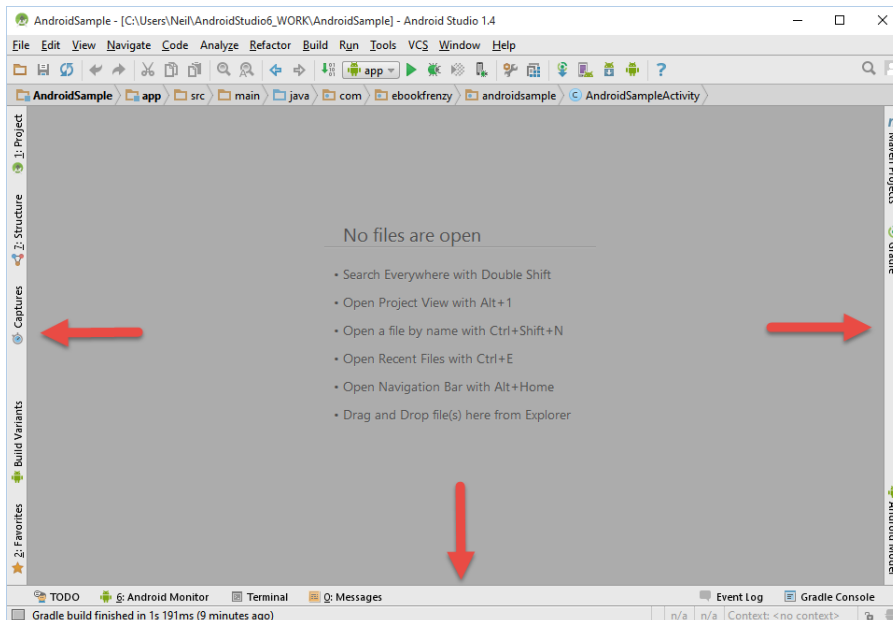


Figure 4-5

Clicking on a button will display the corresponding tool window while 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 and to move and resize the tool panel.

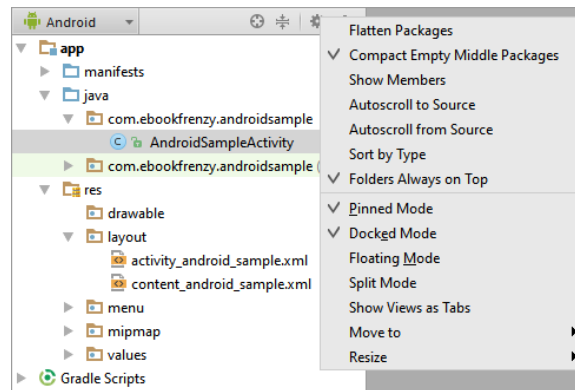


Figure 4-6

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.

Captures – The captures tool window provides access to performance data files that have been generated by the monitoring tools contained within the Android Monitor tool 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 *Editor*.

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 runnable applications) and can be useful for identifying the causes of build problems when compiling application projects.

Android Monitor – The Android Monitor 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. The tool also includes real-time GPU, networking, memory and CPU usage monitors.

Android Model – The Android Model tool window provides a single location in which to view an exhaustive list of the different options and settings configured within the project. These can range from the more obvious settings such as the target Android SDK version to more obscure values such as build configuration rules.

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, while 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 tool window.

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.

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.

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).

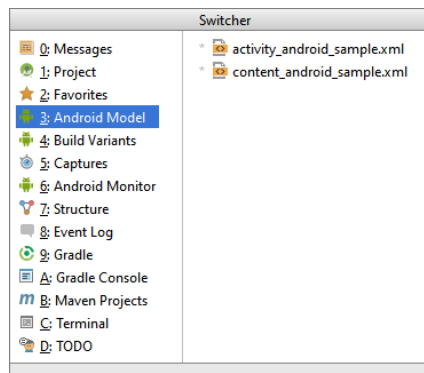


Figure 4-7

A Tour of the Android Studio User Interface

Once displayed, the switcher will remain visible for as long as the Ctrl key remains depressed. Repeatedly tapping the Tab key while holding down the Ctrl key will cycle through the various selection options, while 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.

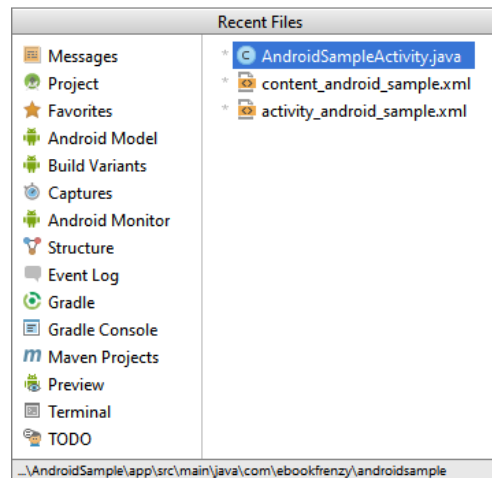


Figure 4-8

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:

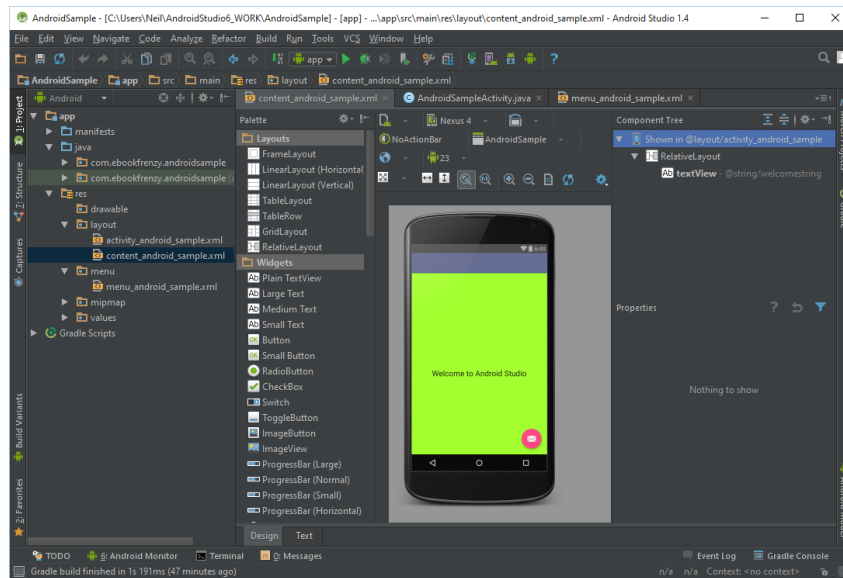


Figure 4-9

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 9 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 9 model.

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.

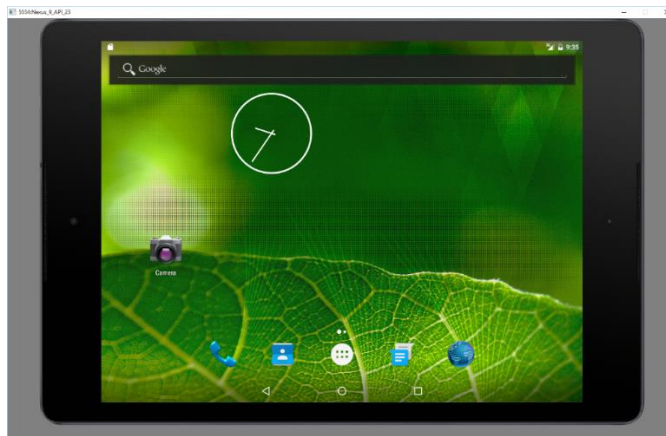


Figure 5-1

5.2 Creating a New AVD

In order to test the behavior of an application in the absence of a physical device, 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 Studio installation, only a Nexus 5 AVD will currently be listed:

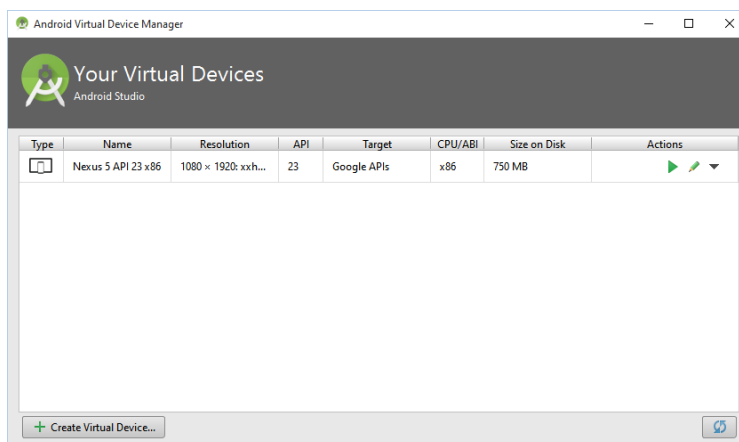


Figure 5-2

To add an additional AVD, begin by clicking on the *Create Virtual Device* button in order to invoke the *Virtual Device Configuration* dialog:

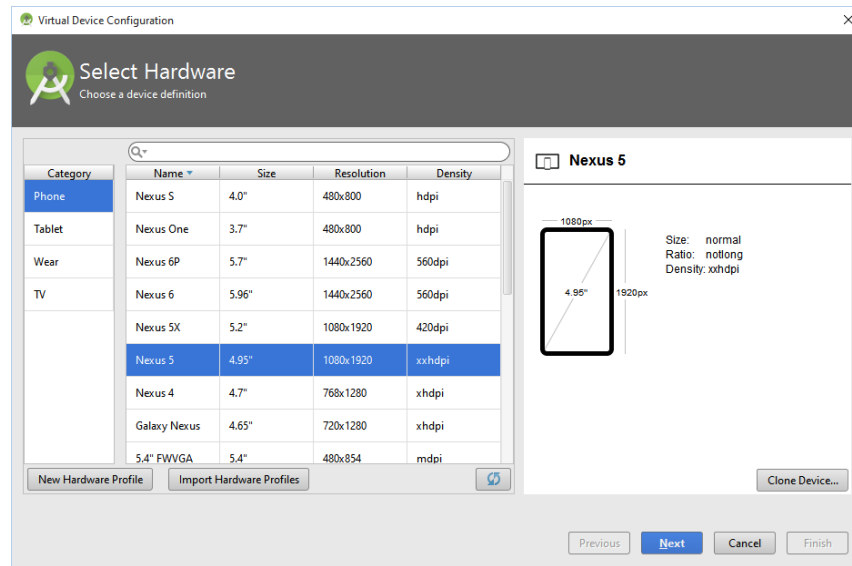


Figure 5-3

Within the dialog, perform the following steps to create a Nexus 9 compatible emulator:

1. From the *Category* panel, select the *Tablet* option to display the list of available Android tablet AVD templates.
2. Select the *Nexus 9* device option and click *Next*.
3. On the *System Image* screen, select the latest version of Android (at time of writing this is Marshmallow, API level 23, Android 6.0) for the *x86* ABI. Note that if the system image has not yet been installed a *Download* link will be provided next to the Release Name. Click this link to download and install the system image before selecting it.
4. Click *Next* to proceed and enter a descriptive name (for example *Nexus 9*) into the name field or simply accept the default name.
5. Click *Finish* to create the AVD.

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 pencil icon in the *Actions* column of the device row in the AVD Manager.

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 launch button (the green triangle in the *Actions* column). The emulator will appear in a new window and, after a short period of time, the “android” logo will appear in the center

Creating an Android Virtual Device (AVD) in Android Studio

of the screen. The amount of time it takes for the emulator to start will depend on the configuration of both the AVD and the system on which it is running. 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.

The emulator probably defaulted to appearing full size and in landscape orientation. It is useful to be aware that these default options can be changed. Within the AVD Manager, select the new Nexus 9 entry and click on the pencil icon in the *Actions* column of the device row. In the configuration screen locate the *Startup size and orientation* section and change the *Scale* menu to *2dp on device = 1px on screen* to reduce the size of the emulator on the screen. Exit and restart the emulator session to see this change take effect.

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:

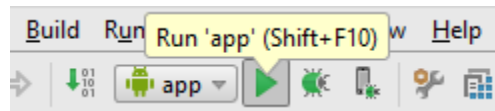


Figure 5-4

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 Nexus 9 AVD as a running device as a result of the steps performed in the preceding section. With this device selected in the dialog, click on *OK* to install and run the application on the emulator.

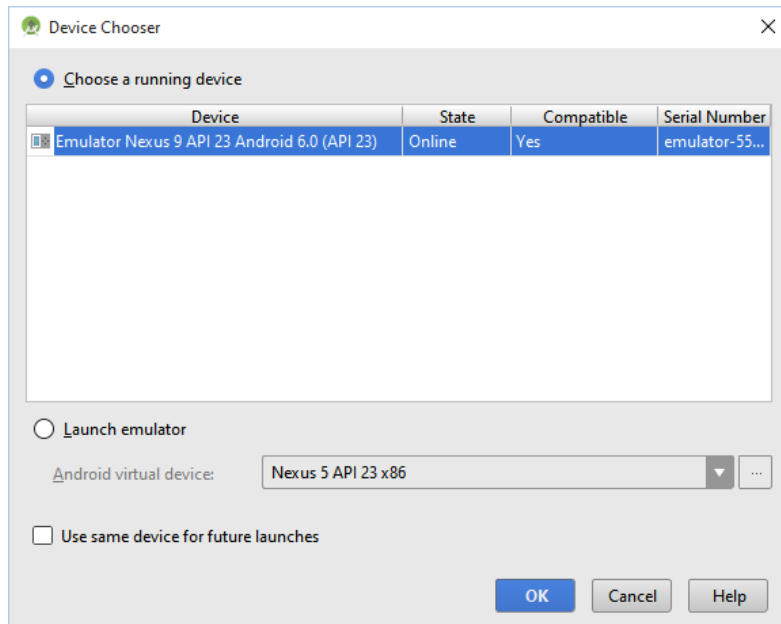


Figure 5-5

Once the application is installed and running, the user interface for the `AndroidSampleActivity` class will appear within the emulator:

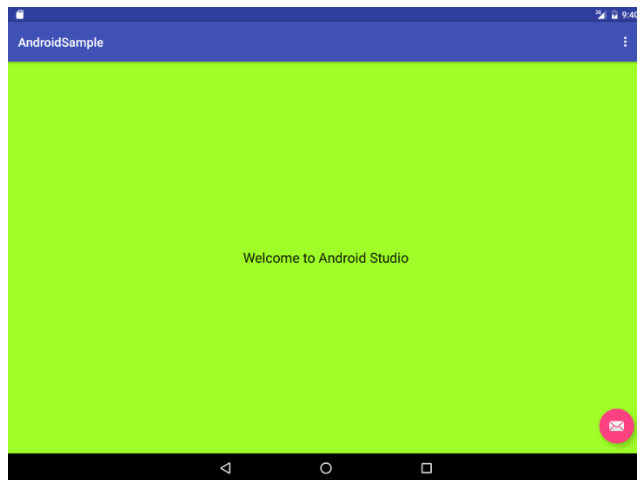


Figure 5-6

In the event that the activity does not automatically launch, check to see if the launch icon has appeared among the apps on the emulator. If it has, simply click on it to launch the application. Once the run process begins, the Run and Android tool windows will become available. The Run tool window

Creating an Android Virtual Device (AVD) in Android Studio

will display diagnostic information as the application package is installed and launched. Figure 5-7 shows the Run tool window output from a successful application launch:

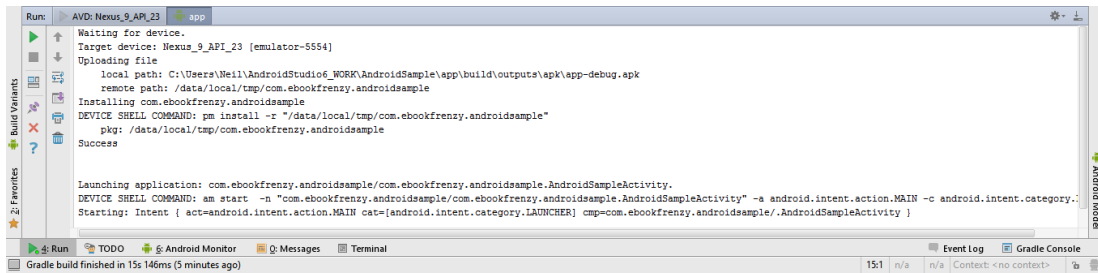


Figure 5-7

If problems are encountered during the launch process, the Run tool will provide information that will hopefully help to isolate the cause of the problem.

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.

5.5 Run/Debug Configurations

A particular project can be configured such that a specific device or emulator is used automatically each time it is run from within Android Studio. This avoids the necessity to make a selection from the device chooser each time the application is executed. To review and modify the Run/Debug configuration, click on the button to the left of the run button in the Android Studio toolbar and select the *Edit Configurations...* option from the resulting menu:

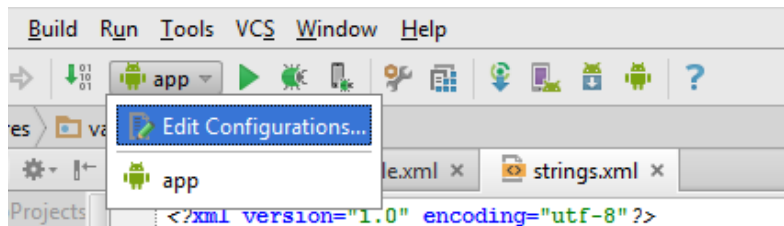


Figure 5-8

In the *Run/Debug Configurations* dialog, the application may be configured to always use a preferred emulator by enabling the *Emulator* option listed in the *Target Device* section and selecting the emulator from the drop down menu. Figure 5-9, for example, shows the AndroidSample application configured to run by default on the previously created Nexus 9 emulator:

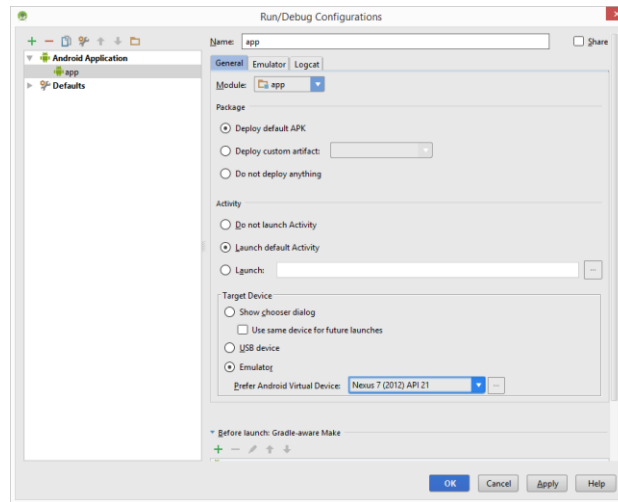


Figure 5-9

Be sure to switch the Target Device setting back to "Show chooser dialog" mode before moving on to the next chapter of the book.

5.6 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, however, to manually stop a running application from within Android Studio.

To stop a running application, begin by displaying the *Android Monitor* tool window either using the window bar button, or via the quick access menu (invoked by moving the mouse pointer over the button in the left hand corner of the status bar as shown in Figure 5-10).

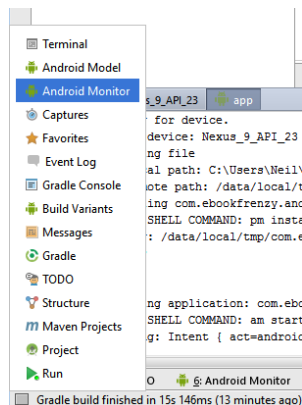


Figure 5-10

Creating an Android Virtual Device (AVD) in Android Studio

Once the Android tool window appears, select the *androidsample* app menu highlighted in Figure 5-11 below:

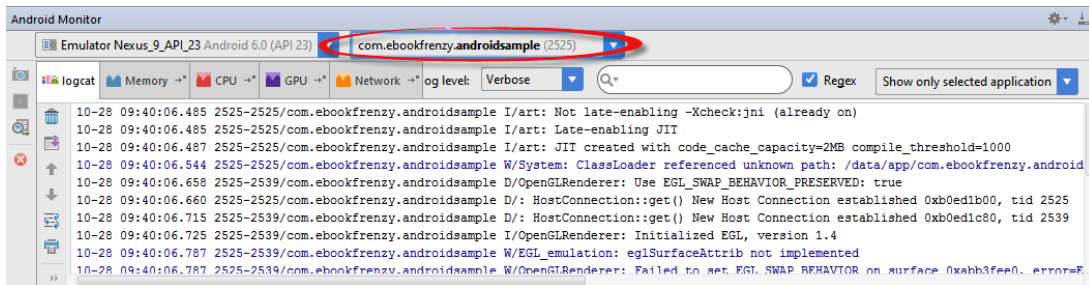


Figure 5-11

With the process selected, stop it by clicking on the red *Terminate Application* button in the vertical toolbar to the left of the process list:

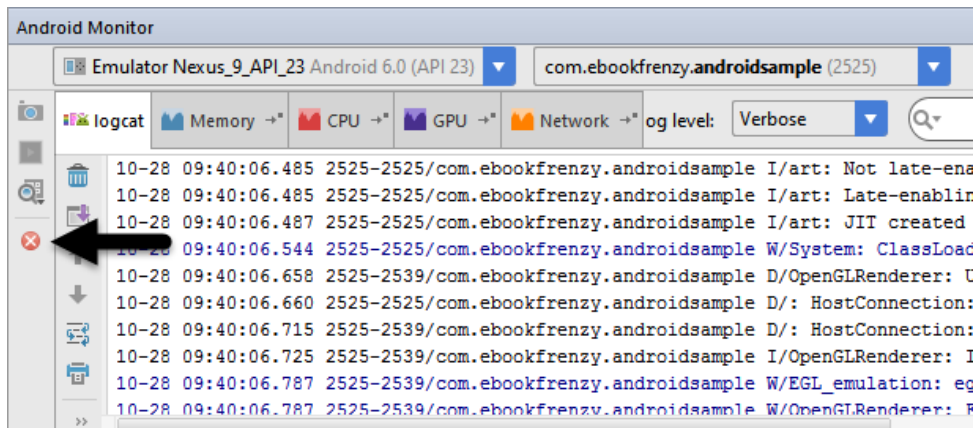


Figure 5-12

An alternative to using the Android tool window is to open the Android Device Monitor. This can be launched via the *Tools -> Android -> Android Device Monitor* menu option. Once launched, the process may be selected from the list (Figure 5-13) and terminated by clicking on the red *Stop* button located in the toolbar above the list.

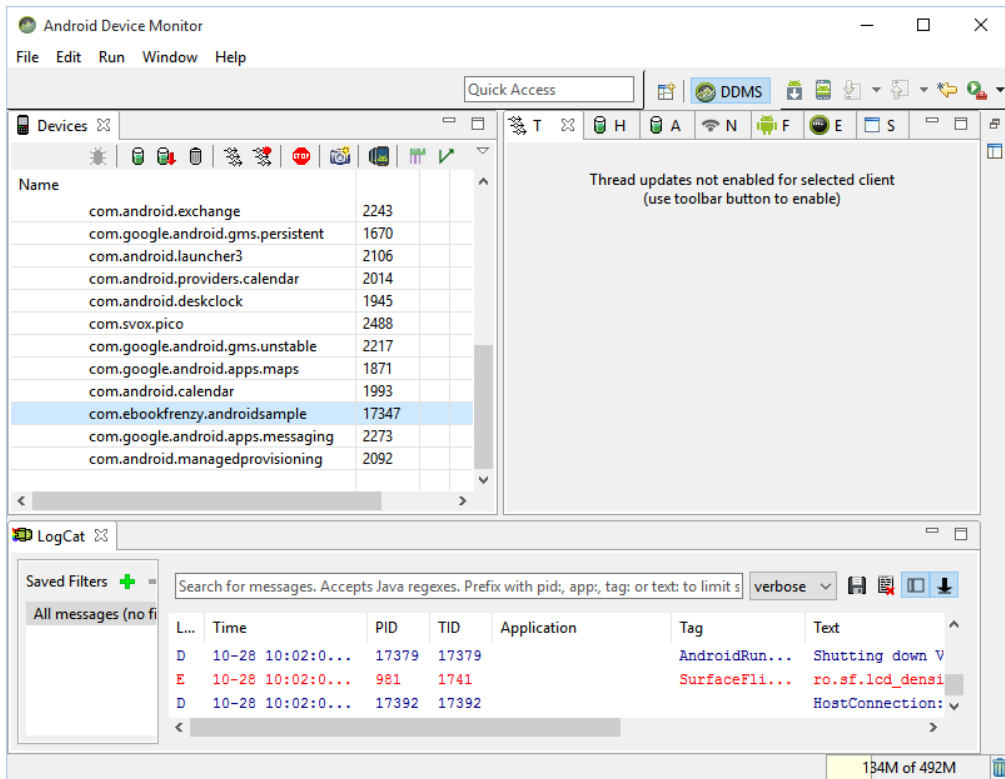


Figure 5-13

5.7 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 "Google Inc.:Google APIs:23"
```

Creating an Android Virtual Device (AVD) in Android Studio

```
Name: Google APIs
Type: Add-On
Vendor: Google Inc.
Revision: 1
Description: Android + Google APIs
Based on Android 6.0 (API level 23)
Libraries:
  * com.google.android.media.effects (effects.jar)
    Collection of video effects
  * com.android.future.usb.accessory (usb.jar)
    API for USB Accessories
  * com.google.android.maps (maps.jar)
    API for Google Maps
Skins: HVGA, QVGA, WQVGA400, WQVGA432, WSVGA, WVGA800 (default),
WVGA854, WXGA720, WXGA800, WXGA800-7in
Tag/ABIs : google_apis/x86
```

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 *Nexus9* using the target id for the Android 6.0 API level 23 device (in this case id 1) using the Google x86 ABI, the following command may be used:

```
android create avd -n Nexus9 -t 1 --abi "google_apis/x86"
```

The android tool will create the new AVD to the specifications required for a basic Android 6.0 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

Available Android Virtual Devices:
  Name: Nexus9
  Path: C:\Users\Neil\.android\avd\Nexus9.avd
  Target: Google APIs (Google Inc.)
    Based on Android 6.0 (API level 23)
  Tag/ABI: google_apis/x86
  Skin: WVGA800
```

```

-----
Name: Nexus_5_API_23_x86
Device: Nexus 5 (Google)
Path: C:\Users\Neil\.android\avd\Nexus_5_API_23_x86.avd
Target: Google APIs (Google Inc.)
       Based on Android 6.0 (API level 23)
Tag/ABI: google_apis/x86
Skin: nexus_5
Sdcard: 200M
Snapshot: no
-----

Name: Nexus_9_API_23
Device: Nexus 9 (Google)
Path: C:\Users\Neil\.android\avd\Nexus_9_API_23.avd
Target: Google APIs (Google Inc.)
       Based on Android 6.0 (API level 23)
Tag/ABI: google_apis/x86
Skin: nexus_9
Sdcard: C:\Users\Neil\.android\avd\Nexus_9_API_23.avd\sdcard.img
Snapshot: no

```

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

```
android delete avd -n <avd name>
```

5.8 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.

5.9 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 Nexus9 to Nexus9B, the following command may be executed:

```
android move avd -n Nexus9 -r Nexus9B
```

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/Nexus9Test:

```
android move avd -n Nexus9 -p /tmp/Nexus9Test
```

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

5.10 Summary

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

6. Testing Android Studio Apps on a Physical Android Device

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 Android features 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.

6.1 An Overview of the Android Debug Bridge (ADB)

The primary purpose of the ADB is to facilitate interaction between a development system, in this case Android Studio, 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 real 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, Android Studio 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
```