



Codename One Academy

Codename One 6.0 “Chat”

Developer Guide

The official developer guide of the #1 Write Once Run Anywhere framework for Java and Kotlin developers

Codename One Developer Guide

Contents

Preface.....	1
1. Introduction.....	5
1.1. Build Cloud	5
1.1.1. Why Build Servers?	6
1.1.2. How does Codename One Work?	6
Lightweight Architecture	7
Why ParparVM	8
Windows Phone/UWP	8
JavaScript Port	9
Desktop and Android	9
1.1.3. Versions In Codename One	9
1.2. History	9
1.3. Core Concepts of Mobile Programming	10
1.3.1. Density.....	10
1.3.2. Touch Interface	12
1.3.3. Device Fragmentation	13
1.3.4. Performance	13
App Size.....	13
Power Drain	13
1.3.5. Sandbox and Permissions.....	14
1.4. Installing Codename One	14
1.5. IntelliJ/IDEA	14
1.6. NetBeans	15
1.7. Eclipse	16
1.8. Hello World Application	17
1.8.1. Running	19
1.8.2. The Source Code Of The Hello World App	20
1.8.3. Building and Deploying On Devices	24
Signing/Certificates	25
Build and Install	30
1.9. Kotlin	32
1.9.1. Hello Kotlin	32
2. Basics: Themes, Styles, Components and Layouts	37
2.1. Components	37
2.1.1. Layout Managers	38
Terse Syntax	43
Flow Layout	43
Box Layout	45

Border Layout	47
Grid Layout	49
2.1.2. Table Layout	50
The Full Potential	52
TextMode Layout.....	54
2.1.3. Layered Layout	55
Insets and Reference Components	56
auto Insets	57
% Insets.....	58
Insets, Margin, and Padding.....	58
Component References: Linking Components together	59
2.1.4. GridBag Layout	62
2.1.5. Group Layout.....	64
2.1.6. Mig Layout	66
2.2. Themes and Styles	66
2.2.1. Theme	68
Native Theme.....	72
2.3. GUI Builder.....	76
2.3.1. Hello World	78
NetBeans	78
IntelliJ/IDEA	79
Eclipse	80
Basic Usage.....	81
Events	84
Underlying XML	85
2.3.2. Auto-Layout Mode	87
The Inset Control.....	87
Auto Snap	88
Smart Insets	88
The Widget Control Pad.....	88
Keyboard Short-Cuts	89
Sub-Containers	89
The Canvas Resize Tool	90
3. Theme Basics	91
3.1. Understanding Codename One Themes	91
3.2. Customizing Your Theme	92
3.3. Customizing The Title	95
3.3.1. Background Priorities and Types	95
3.3.2. The Background Behavior and Image	96
3.3.3. The Color Settings.....	105
3.3.4. Alignment	106

3.3.5. Padding and Margin	106
3.3.6. Borders	107
3.3.7. 9-Piece Image Border	108
Customizing The 9-Piece Border	111
3.3.8. Horizontal/Vertical Image Border	113
3.3.9. Empty Border	114
3.3.10. Round Border	114
3.3.11. Rounded Rectangle Border	115
3.3.12. Bevel/Etched Borders	115
3.3.13. Derive	116
3.3.14. Fonts	117
Font Effects	121
4. Advanced Theming	123
4.1. Working With UIID's	123
4.2. Theme Layering	123
4.3. Override Resources In Platform	124
4.4. Theme Constants	125
4.5. Native Theming	136
4.6. Under the Hood of the Theme Engine	136
4.7. Understanding Images and Multi-Images	138
4.8. Use Millimeters for Padding/Margin and Font Sizes	140
4.8.1. Fractions of Millimeters	140
4.9. Creating a Great Looking Side Menu	141
4.10. Converting a PSD To A Theme	145
4.10.1. Breaking Down the PSD	147
Removing the Noise	149
The Camera Button	150
4.10.2. The Code	153
4.10.3. Styling The UI	156
Not Quite There Yet	158
5. CSS	159
5.1. Activating CSS	159
5.2. Supported CSS Selectors	160
5.2.1. Inheriting properties using <code>cn1-derive</code>	161
5.3. Special Selectors	161
5.3.1. <code>#Device</code>	161
5.3.2. <code>#Constants</code>	161
5.3.3. <code>Default</code>	163
5.4. Standard CSS Properties	163
5.5. Custom Properties	164
5.6. CSS Variables	164

5.7. CSS Properties	165
5.7.1. text-decoration	165
5.7.2. border	166
Round Borders	166
5.7.3. background	168
Background Images	168
Gradients	168
5.7.4. cn1-background-type	174
5.8. Images	175
5.8.1. Image DPI and Device Densities	175
5.8.2. Multi-Images vs Regular Images	176
5.8.3. Multi-Images as Inputs	176
5.8.4. Image Constants	177
5.9. Image Recipes	178
5.9.1. Import Multiple Images In Single Selector	178
5.9.2. Loading Images from URLs	179
5.9.3. Generating 9-Piece Image Borders	179
5.9.4. Image Backgrounds	180
Example Setting Background Image to Scale Fill	181
5.10. Image Compression	181
5.11. Fonts	181
5.11.1. <code>font-family</code>	182
Using TTF Fonts	182
5.11.2. <code>font-size</code>	183
5.11.3. <code>text-decoration</code>	183
5.11.4. Some Sample CSS Directives	184
5.12. Media Queries	188
5.12.1. Compound Media Queries	189
5.12.2. Order or Precedence	190
5.12.3. Font Scaling Constants	190
6. The Components of Codename One	193
6.1. Container	193
6.1.1. Composite Components	193
6.2. Form	194
6.3. Dialog	196
6.3.1. Styling Dialogs	197
6.3.2. Tint and Blurring	198
6.3.3. Popup Dialog	200
Styling The Arrow Of The Popup Dialog	200
6.4. InteractionDialog	201
6.5. Label	202

6.5.1. Label Gap	203
6.5.2. Autosizing Labels	203
6.6. TextField and TextArea	204
6.6.1. Masking	207
6.6.2. The Virtual Keyboard	207
Action Button Client Property	208
Next and Done on iOS	208
6.6.3. Clearable Text Field	209
6.7. TextComponent	209
6.7.1. Error Handling	210
6.7.2. InputComponent and PickerComponent	211
6.7.3. Underlying Theme Constants and UIID's	212
6.8. Button	213
6.8.1. Uppercase Buttons	214
6.8.2. Raised Button	214
6.8.3. Ripple Effect	215
6.9. CheckBox/RadioButton	216
6.9.1. Toggle Button	217
6.10. ComponentGroup	218
6.11. MultiButton	219
6.11.1. Styling The MultiButton	221
6.12. SpanButton	221
6.13. SpanLabel	221
6.14. OnOffSwitch	222
6.14.1. Validation	223
6.15. InfiniteProgress	224
6.16. InfiniteScrollAdapter and InfiniteContainer	225
6.16.1. The InfiniteContainer	227
6.17. List, MultiList, Renderers & Models	228
6.17.1. InfiniteContainer/InfiniteScrollAdapter vs. List/ContainerList	228
6.17.2. Why Isn't List Deprecated?	229
6.17.3. MVC In Lists	229
6.17.4. Understanding MVC	229
Why is this useful?	230
6.17.5. Important - Lists & Layout Managers	231
6.17.6. MultiList & DefaultListModel	231
Going Further With the ListModel	233
6.17.7. List Cell Renderer	235
6.17.8. Generic List Cell Renderer	237
Custom UIID Of Entry in GenenricListCellRenderer/MultiList	240
Rendering Prototype	240

6.17.9. ComboBox	241
6.18. Slider	242
6.19. Table	243
6.19.1. Sorting Tables	249
6.20. Tree	249
6.21. ShareButton	252
6.22. Tabs	254
6.23. MediaManager & MediaPlayer	256
6.24. ImageViewer	258
6.25. ScaleImageLabel & ScaleImageButton	261
6.26. Toolbar	262
6.26.1. Search Mode	265
6.26.2. South Component	267
6.26.3. Title Animations	267
6.27. BrowserComponent & WebBrowser	269
6.27.1. BrowserComponent Hierarchy	271
6.27.2. NavigationCallback	271
6.27.3. JavaScript	272
So what was wrong with the old API?	273
The New API	273
Synchronous Wrappers	274
Multi-use Callbacks	275
Passing Parameters to Javascript	275
Proxy Objects	276
Legacy JSObject Support	277
The JavaScript Bridge	278
6.27.4. Cordova/PhoneGap Integration	280
6.28. AutoCompleteTextField	281
6.28.1. Using Images In AutoCompleteTextField	283
6.29. Picker	285
6.30. SwipeableContainer	289
6.31. EmbeddedContainer	290
6.32. MapComponent	290
6.33. Chart Component	293
6.33.1. Device Support	293
6.33.2. Features	294
6.33.3. Chart Types	294
6.33.4. How to Create A Chart	296
6.34. Calendar	297
6.35. ToastBar	298
6.35.1. Actions In ToastBar	300

6.36. SignatureComponent	301
6.37. Accordion	301
6.38. Floating Hint	302
6.39. Floating Button	303
6.39.1. Using Floating Button as a Badge	304
6.40. SplitPane	304
7. Animations	307
7.1. Layout Reflow	307
7.2. Layout Animations	307
7.2.1. Unlayout Animations	311
7.2.2. Hiding & Visibility	312
7.2.3. Synchronicity In Animations	312
Animation Fade and Hierarchy	313
7.2.4. Sequencing Animations Via AnimationManager	314
Animation Manager to the Rescue	315
7.3. Low Level Animations	315
7.3.1. Why Not Just Write Code In Paint?	316
7.4. Transitions	316
7.4.1. Replace	317
7.4.2. Slide Transitions	318
7.4.3. Fade and Flip Transitions	320
7.4.4. Bubble Transition	321
7.4.5. Morph Transitions	323
7.4.6. SwipeBackSupport	323
8. The EDT - Event Dispatch Thread	325
8.1. What Is The EDT	325
8.2. Call Serially (And Wait)	326
8.2.1. callSerially On The EDT	327
8.3. Debugging EDT Violations	327
8.4. Invoke And Block	328
9. Monetization	333
9.1. Google Play Ads	333
9.2. In App Purchase	333
9.2.1. The SKU	333
9.2.2. Types of Products	334
9.2.3. The "Hello World" of In-App Purchase	334
9.2.4. Making it Consumable	338
9.2.5. Non-Renewable Subscriptions	339
9.2.6. The Server-Side	340
The Receipts API	340
9.2.7. The "Hello World" of Non-Renewable Subscriptions	340

Implementing the Receipt Store	341
Synchronizing Receipts	345
Expiry Dates and Subscription Status	346
Allowing the User to Purchase the Subscription.....	347
9.2.8. <code>subscribe()</code> vs <code>purchase()</code>	349
Handling Purchase Callbacks	349
9.2.9. Screenshots	349
9.2.10. Summary	351
9.2.11. Auto-Renewable Subscriptions	351
9.2.12. Auto-Renewable vs Non-Renewable. Best Choice?	352
9.2.13. Learning By Example	352
9.2.14. Building the IAP Demo Project	352
Setting up the Client Project.....	352
Setting up the Server Project	353
Setting up the Database.....	353
Testing the Project	354
9.2.15. Looking at the Source of the App	357
Client Side.....	357
Server-Side	359
9.2.16. The CN1-IAP-Validator Library	361
9.2.17. The <code>validateAndSaveReceipt()</code> Method	362
9.2.18. Google Play Setup	365
Creating the App in Google Play	365
Testing The App	367
Creating Google Play Receipt Validation Credentials	367
9.2.19. iTunes Connect Setup	371
Setting up In-App Products	371
Creating Test Accounts	371
Setting up Receipt Verification	372
10. Graphics, Drawing, Images & Fonts	373
10.1. Basics - Where & How Do I Draw Manually?.....	373
10.2. Glass Pane	375
10.3. Shapes & Transforms	376
10.4. Device Support	376
10.5. A 2D Drawing App	376
10.5.1. Implementing <code>addPoint()</code>	378
10.5.2. Using Bezier Curves	378
10.5.3. Detecting Platform Support	380
10.6. Transforms	380
10.6.1. Device Support	381
10.7. Example: Drawing an Analog Clock.....	381

10.7.1. The AnalogClock Component	382
10.7.2. Setting up the Parameters	382
10.7.3. Drawing the Tick Marks	383
10.7.4. Drawing the Numbers	385
10.7.5. Drawing the Hands	387
10.7.6. The Final Result	390
10.7.7. Animating the Clock	390
10.8. Starting and Stopping the Animation	391
10.9. Shape Clipping	391
10.10. The Coordinate System	393
10.10.1. Relative Coordinates	394
10.10.2. Transforms and Rotations	395
10.10.3. Event Coordinates	398
10.11. Images	398
10.11.1. Loaded Image	398
10.11.2. The RGB Image's	399
Internal	399
RGBImage class	399
10.11.3. EncodedImage	400
10.11.4. MultiImage	401
10.11.5. FontImage & Material Design Icons	401
Material Design Icons	402
10.11.6. Timeline	403
10.11.7. Image Masking	403
10.11.8. URLImage	404
Mask Adapter	406
URLImage In Lists	406
11. Events	409
11.1. High Level Events	409
11.1.1. Chain Of Events	409
11.1.2. Action Events	409
Types Of Action Events	410
Source Of Event	410
Event Consumption	410
NetworkEvent	411
11.1.3. DataChangeListener	412
11.1.4. FocusListener	412
11.1.5. ScrollListener	413
11.1.6. SelectionListener	413
11.1.7. StyleListener	414
11.1.8. Event Dispatcher	414

11.2. Low Level Events	414
11.2.1. Low Level Event Types	415
11.2.2. Drag Event Sanitation	416
11.3. BrowserNavigationCallback	416
12. File System, Storage, Network & Parsing	419
12.1. Jar Resources	419
12.2. Storage	419
12.2.1. The Preferences API	422
12.3. File System	423
12.3.1. File Paths & App Home	423
12.3.2. Storage vs. File System	425
12.4. SQL	426
12.5. Network Manager & Connection Request	428
12.5.1. Threading	430
12.5.2. Arguments, Headers & Methods	430
Arguments	430
Methods	431
Headers	431
Server Headers	432
Error Handling	432
Error Stream	433
12.5.3. GZIP	434
12.5.4. File Upload	434
12.5.5. Parsing	435
Parsing CSV	435
JSON	436
XML Parsing	439
XPath Processing	440
Properties Files	446
12.6. Debugging Network Connections	447
12.6.1. Simpler Downloads	447
Downloading Images	447
12.7. Rest API	449
12.7.1. Rest in Practice - Twilio	449
12.8. Webservice Wizard	451
12.9. Connection Request Caching	455
12.9.1. getCachedData()	456
12.9.2. cacheUnmodified()	456
12.9.3. purgeCache & purgeCacheDirectory	457
12.10. Cached Data Service	457
12.11. Externalizable Objects	457

12.12. UI Bindings & Utilities	462
12.13. Logging & Crash Protection	463
12.14. Sockets	463
12.15. Properties	465
12.15.1. Properties in Java	467
Encapsulation	467
Introspection & Observability	468
12.15.2. The Cool Stuff	468
Constructors	469
Seamless Serialization	470
Seamless SQL Storage	471
Preferences Binding	473
UI Binding	474
UI Generation	476
13. Push Notifications	479
13.1. Understanding Push Notifications	479
13.2. Implementing Push Support	479
13.3. The Push Lifecycle	480
13.3.1. Registration	480
13.3.2. Sending a Push Notification	481
13.3.3. Receiving a Push Notification	481
13.4. Testing Push Support	481
13.5. Push Types and Message Structure	484
13.5.1. Example Push Type 1	485
13.5.2. Example Push Type 2	485
13.5.3. Example Push Type 3	486
13.5.4. Example Push Type 4	486
13.5.5. Example Push Type 5	487
13.5.6. Example Push Type 100	487
13.5.7. Example Push Type 101	487
13.6. Rich Push Notifications	488
13.6.1. Image Attachment Support	488
13.6.2. Notification Actions	490
13.7. Deploying Push-Enabled Apps to Device	493
13.7.1. The Push Bureaucracy - Android	493
13.7.2. The Push Bureaucracy - iOS	496
13.7.3. The Push Bureaucracy - UWP (Windows 10)	498
13.7.4. The Push Bureaucracy - Javascript	499
13.8. Sending Push Messages	500
13.8.1. Sending a Push Message From Codename One	501
13.8.2. Sending Push Message From A Java or Generic Server	502

Server JSON Responses	503
14. Miscellaneous Features	505
14.1. Phone Functions	505
14.1.1. SMS	505
14.1.2. Dialing	506
14.1.3. E-Mail	506
14.2. Contacts API	507
14.3. Localization & Internationalization (L10N & I18N)	510
14.3.1. Localization Manager	511
14.3.2. RTL/Bidi	512
14.4. Location - GPS	513
14.4.1. Location In The Background - Geofencing	514
14.5. Background Music Playback	516
14.6. Capture - Photos, Video, Audio	516
14.6.1. Capture Asynchronous API	523
14.7. Gallery	523
14.8. Analytics Integration	524
14.8.1. Application Level Analytics	525
14.8.2. Overriding The Analytics Implementation	525
14.9. Native Facebook Support	525
14.9.1. Getting Started - Web Setup	525
14.9.2. IDE Setup	528
14.9.3. The Code	528
14.9.4. Facebook Publish Permissions	529
14.10. Google Sign-In	530
14.10.1. iOS Setup Instructions	530
14.10.2. Android Setup Instructions	533
14.10.3. OAuth Setup (Simulator and REST API Access)	536
Client ID, Client Secret and Redirect URL	536
14.10.4. Javascript Setup Instructions	537
14.10.5. The Code	537
14.11. Lead Component	538
14.11.1. Blocking Lead Behavior	539
14.12. Pull To Refresh	541
14.13. Running 3rd Party Apps Using Display's execute	541
14.14. Automatic Build Hint Configuration	542
14.15. Easy Thread	543
14.16. Mouse Cursor	544
14.17. Working With GIT	544
14.17.1. cn1lib's	544
14.17.2. Resource Files	545

14.17.3. Eclipse Version	545
14.17.4. IntelliJ/IDEA	546
15. Performance, Size & Debugging	549
15.1. Reducing Resource File Size	549
15.2. Improving Performance	550
15.3. Performance Monitor	551
15.4. Network Speed	552
15.5. Debugging Codename One Sources	553
15.6. Device Testing Framework/Unit Testing	553
15.7. EDT Error Handler and sendLog	554
15.8. Kitchen Sink Case Study	555
15.8.1. Scroll Performance - Threads aren't magic	556
16. Advanced Topics/Under the Hood	559
16.1. Sending Arguments To The Build Server	559
16.2. Offline Build	574
16.2.1. Prerequisites for iOS Builds	574
16.2.2. Prerequisites for Android Builds	575
16.2.3. Installation	575
16.2.4. Building	576
16.2.5. FAQ	577
Should I use the Offline Builder?	577
Can I Move/Backup my Builders?	577
Can I install the builders for all our developers?	577
What Happens if I Cancel?	578
When are Versions Released?	578
Are Version Numbers Sequential?	578
Why is this Feature Limited to Enterprise Subscribers?	578
How Different is the Code From Cloud Builds?	578
16.3. Android Permissions	578
16.3.1. Permissions Under Marshmallow (Android 6+)	579
Enabling Permissions	580
Permission Prompts	580
Code Changes	582
Simulating Prompts	583
AndroidNativeUtil's checkForPermission	583
16.4. On Device Debugging	583
16.4.1. Android Studio Debugging (Easy Way)	584
16.4.2. Android Studio Debugging the Hard Way	584
16.5. Native Interfaces	585
16.5.1. Introduction	585
Use the Android Main Thread (Native EDT)	588

Gradle Dependencies	589
16.5.2. Objective-C (iOS)	590
Using the iOS Main Thread (Native EDT)	591
Use Cocoapods For Dependencies	592
16.5.3. Javascript	592
JavaScript Examples	594
16.5.4. Native GUI Components	595
16.5.5. Type Mapping & Rules	596
16.5.6. Android Native Permissions	598
16.5.7. Native AndroidNativeUtil	599
16.5.8. Broadcast Receiver	599
Listening & Permissions	602
16.5.9. Native Code Callbacks	603
Accessing Callbacks from Objective-C	604
Accessing Callbacks from Javascript	605
Callbacks of the SMS Receiver	606
Asynchronous Callbacks & Threading	607
16.6. Libraries - cn1lib	608
16.6.1. Why Not Use JAR?	609
16.6.2. How To Use cn1libs?	609
16.6.3. Creating a Simple cn1lib	610
16.6.4. Build Hints in cn1libs	610
16.7. Integrating Android 3rd Party Libraries & JNI	614
16.8. Drag & Drop	615
16.9. Continuous Integration & Release Engineering	617
16.10. Android Lollipop ActionBar Customization	617
16.11. Intercepting URL's On iOS & Android	618
16.11.1. Passing Launch Arguments To The App	618
16.12. Native Peer Components	619
16.12.1. Why does Codename One Need Native Widgets at all?	619
16.12.2. So what's the problems with native widgets?	620
16.12.3. So how do we show dialogs on top of Peer Components?	620
16.12.4. Why can't we combine peer component scrolling and Codename One scrolling?	620
16.12.5. Native Components In The First Form	620
16.13. Integrating 3rd Party Native SDKs	620
16.13.1. Step 1 : Review the FreshDesk SDKs	621
16.13.2. Step 2: Designing the Codename One Public API	621
16.13.3. Step 3: The Architecture and Internal APIs	621
Things to Notice	623
16.13.4. Step 4: Implement the Public API and Native Interface	623
Adapting Method Signatures	623

Callbacks	624
Initialization	626
The Resulting Public API	627
The Native Interface	629
Connecting the Public API to the Native Interface	631
Implementing the Glue Between Public API and Native Interface	632
16.13.5. Step 5: Implementing the Native Interface in Android	633
16.13.6. Step 6: Bundling the Native SDKs	635
The FreshDesk SDK	635
Dependencies	635
16.13.7. Step 7 : Injecting Android Manifest and Proguard Config	637
Proguard Config	639
Troubleshooting Android Stuff	640
16.14. Part 2: Implementing the iOS Native Code	640
16.14.1. Using the MobiHelpNativeCallback	641
16.14.2. Bundling Native iOS SDK	641
16.14.3. Troubleshooting iOS	642
16.14.4. Adding Required Core Libraries and Frameworks	642
16.15. Part 3 : Packaging as a cn1lib	643
16.16. Building Your Own Layout Manager	643
16.16.1. Porting a Swing/AWT Layout Manager	645
16.17. Port a Language to Codename One	645
16.17.1. What is a JVM Language?	645
16.17.2. How Hard is it to Port a JVM Language to Codename One?	646
Step 1: Assess the Language	646
Step 2: Convert the Runtime Library into a CN1Lib	651
Step 3: Hello World	652
Step 4: A More Complex Hello World	653
Step 5: Automation and Integration	653
16.18. Update Framework	655
16.18.1. How does it Work?	655
16.18.2. What isn't Covered	656
17. Signing, Certificates & Provisioning	657
17.1. Common Terms In Signing & Distribution	657
17.1.1. What Is A Certificate?	657
17.1.2. What Is Provisioning?	657
17.1.3. What's a Signing Authority?	657
17.1.4. What is UDID?	657
17.1.5. Should I Reuse the Same Certificate for All Apps?	658
17.2. iOS Signing Wizard	658
17.2.1. Logging into the Wizard	658

17.2.2. Selecting Devices	659
17.2.3. Decisions & Edge Cases	660
17.2.4. App IDs and Provisioning Profiles	661
17.2.5. Installing Files Locally	662
17.2.6. Building Your App	663
17.3. Advanced iOS Signing	663
17.4. Provisioning Profile & Certificates Visual Guide	664
17.4.1. iOS Code Signing Failure Checklist	667
17.5. Android	669
17.5.1. Generating an Android Certificate Manually	669
17.6. RIM/BlackBerry	670
17.7. J2ME	670
18. Working with iOS	671
18.1. Troubleshooting iOS Debug Build installs	671
18.2. The iOS Screenshot/Splash Screen Process	672
18.2.1. Size	673
18.2.2. Mutable first screen	673
18.2.3. Unsupported component	674
18.3. Launch Screen Storyboards	674
18.3.1. Launch Storyboard vs Launch Images	674
18.4. Local Notifications on iOS and Android	674
18.4.1. Sending Notifications	675
Example Sending Notification	675
18.4.2. Receiving Notifications	675
Example Receiving Notification	676
18.4.3. Canceling Notifications	676
18.5. iOS Beta Testing (Testflight)	677
18.6. Accessing Insecure URL's	677
18.7. Using Cocoapods	677
18.7.1. Including Multiple Pods	678
18.7.2. Other Pod Related Build Hints	678
18.7.3. Converting PodFile To Build Hints	678
18.8. Including Dynamic Frameworks	679
19. Working with JavaScript	681
19.1. Limitations of the Javascript Port	681
19.1.1. No Multithreaded Code inside Static Initializers	681
19.2. Troubleshooting Build Errors	682
19.3. ZIP, WAR, or Preview. What's the difference?	683
19.4. Setting up a Proxy for Network Requests	685
19.4.1. Step 1: Setting up a Proxy	685
19.4.2. Step 2: Configuring your Application to use the Proxy	685

19.5. Using the CORS Proxy for Same Origin Requests	686
Using Apache as a Proxy	687
19.6. Customizing the Splash Screen	687
19.7. Debugging	687
19.8. Including Third-Party Javascript Libraries	688
19.8.1. Libraries vs Resources	688
19.8.2. The Javascript Manifest File	689
How to NOT generate the <script> tag	690
Library Directories	690
Including Remote Libraries	691
Including CSS Files	692
Embedding Variables in URLs	693
19.9. Browser Environment Variables	693
19.10. Changing the Native Theme	694
19.10.1. Example: Using Android Theme on Android	695
19.11. Disabling the 'OnBeforeUnload' Handler	695
19.11.1. Example: Toggling the BeforeUnload Prompt On/Off	695
19.12. Deploying as a Progressive Web App	696
19.12.1. Customizing the App Manifest File	697
19.12.2. Related Applications	697
19.12.3. Device/Browser Support for PWAs	698
19.13. Playing Media and Opening Links	699
20. Working with UWP	701
20.1. Deploying Outside of the Windows App Store (Sideloading)	701
20.1.1. Side-loading to Windows 10 Mobile Devices	701
Enabling Developer Mode on Device	701
Building App for UWP	703
Installing App On Device	703
20.1.2. Side-loading to Windows 10 Desktop Devices	704
Enabling Developer Mode on PC	704
Building the App	705
Installing the App	706
20.1.3. Building for the Windows Store	707
20.2. Debugging UWP Apps	710
20.2.1. No Line Numbers in Stack Traces	710
20.3. Customizing the Status Bar	712
20.4. Associating App with File Types	712
21. Working with Mac OS X	715
21.1. Mac OS Desktop Build Options	715
21.1.1. Bundle Types	715
21.1.2. Understanding Mac Certificates	715

21.1.3. Obtaining Certificates	716
21.1.4. Exporting Certificates as P12	717
21.1.5. Entitlements	718
22. Security	719
22.1. Constant Obfuscation	719
22.2. Storage Encryption	721
22.3. Disabling Screenshots	722
22.4. Blocking Copy & Paste	722
22.5. Blocking Jailbreak	722
22.6. Strong Android Certificates	723
22.6.1. The Bad News	723
22.6.2. The Good	724
22.7. Certificate Pinning	724
22.7.1. Certificate Pinning	724
23. Travis CI Integration	727
23.1. Quick Start	727
23.1.1. Enabling Travis	727
Activating/Deactivating Jobs	729
23.1.2. Pushing to GitHub	730
23.1.3. Activate Repository On Travis	731
23.1.4. Setting Environment Variables	731
23.1.5. Testing Travis Script	732
23.1.6. Writing Unit Tests	733
Disabling Travis	733
24. Working with Codename One Sources	735
24.1. Checking out the Sources	735
24.2. Building Sources	735
24.3. Running Unit Tests	735
24.4. Running iOS Unit Tests	735
24.4.1. Installing npm	736
24.4.2. Installing and Running Appium	736
24.4.3. Running the Unit tests	736
24.4.4. Overriding the Codename One Build Server Target	736

Preface

This developer guide is automatically generated from the wiki pages at <https://github.com/codenameone/CodenameOne/wiki> [<https://github.com/codenameone/CodenameOne/wiki>].

You can edit any page within the wiki pages using AsciiDoc. Your changes might make it (after review) into the official documentation available on the web here: <https://www.codenameone.com/manual/> and available as a PDF file here: <https://www.codenameone.com/files/developer-guide.pdf> [<https://www.codenameone.com/files/developer-guide.pdf>].

Occasionally this book is updated to the print edition available here: <https://www.amazon.com/dp/1549910035>

We also recommend that you check out the full [JavaDoc](https://www.codenameone.com/javadoc/) [<https://www.codenameone.com/javadoc/>] reference for Codename One which is very complete.

You can download the full Codename One source code from the [Codename One git repository](https://github.com/codenameone/CodenameOne/) [<https://github.com/codenameone/CodenameOne/>] where you can also edit the JavaDocs and submit pull requests to [include new features into Codename One](https://www.codenameone.com/blog/how-to-use-the-codename-one-sources.html) [<https://www.codenameone.com/blog/how-to-use-the-codename-one-sources.html>].

Authors

This document includes content from multiple authors and community wiki edits. If you edit pages within the guide feel free to add your name here alphabetized by surname:

- [Shai Almog](https://github.com/codenameone) [[https://github.com/codenameone/](https://github.com/codenameone)]
- [Ismael Baum](https://github.com/Isborg) [<https://github.com/Isborg>]
- [Eric Coolman](https://twitter.com/ericcoolmandev) [<https://twitter.com/ericcoolmandev>]
- [Chen Fishbein](http://github.com/chen-fishbein/) [<http://github.com/chen-fishbein/>]
- [Steve Hannah](http://github.com/shannah/) [<http://github.com/shannah/>]
- [Matt](https://github.com/kheops37) [<https://github.com/kheops37>]

Rights & Licensing

You may copy/redistribute/print this document without prior permission from Codename One. However, you may not charge for the document itself although charging for costs such as printing is permitted.

Notice that while you can print and reproduce sections arbitrarily such changes must explicitly and clearly state that this is a modified copy and link to the original source at <https://www.codenameone.com/manual/> in a clear way!

Conventions

This guide uses some notations to provide tips and further guidance.

In case of further information that breaks from the current tutorial flow we use a sidebar as such:

Sidebar

Dig deeper into some details that don't quite fit into the current flow. We use sidebars for things that are an important detour, you can skip them while reading but you might want to come back and read them later.

Here are common conventions for highlighting notes:



This is an important note



This is a helpful tip



This is a general informational note, something interesting but not crucial



This is a warning something we should pay attention to

This convention is used when we refer to a button or a widget to press in the UI. E.g. press the button labeled Press Here.

Quotes are presented as:

This is a quote

— By Author

Bold is used for light emphasis on a specific word or two within a sentence.

1. Introduction

Codename One is a Write Once Run Anywhere mobile development platform for Java/Kotlin developers. It integrates with IntelliJ/IDEA, Eclipse or NetBeans to provide seamless native mobile development.

Codename One's mission statement is:

Unify the complex and fragmented task of mobile device programming into a single set of tools, APIs and services. As a result create a more manageable approach to mobile application development without sacrificing the power/control given to developers.

This effectively means bringing that old "Write Once Run Anywhere" (WORA) Java mantra to mobile devices without "dumbing it down" to the lowest common denominator.

The things that make Codename One stand out from other tools in this field are:

- Write Once Run Anywhere support with no special hardware requirements and 100% code reuse
- Compiles Java/Kotlin into native code for iOS, UWP (Universal Windows Platform), Android and even JavaScript/PWA
- Open Source and Free with commercial backing/support
- Easy to use with 100% portable Drag and Drop GUI builder
- Full access to underlying native OS capabilities using the native OS programming language (e.g. Objective-C) without compromising portability
- Provides full control over every pixel on the screen
- Lets you use native widgets (views) and mix them with Codename One components within the same hierarchy (heavyweight/lightweight mixing)
- Supports seamless Continuous Integration out of the box

Codename One can trace its roots to the open source LWUIT project started at Sun Microsystem in 2007 by Chen Fishbein (co-founder of Codename One). It's a huge project that's been under constant development for over a decade!

1.1. Build Cloud

One of the things that make Codename One stand out is the build cloud approach to mobile development. iOS native development requires a Mac with xcode. Windows native development requires a Windows machine. To make matters worse, Apple, Google and Microsoft make changes to their tools on a regular basis...

This makes it hard to keep up.

When we develop an app in Codename One we use the builtin simulator when running and

debugging. When we want to build a native app we can use the build cloud where Macs create the native iOS apps and Windows machines create the native Windows apps. This works seamlessly and makes Codename One apps native as they are literally compiled by the native platform. E.g. for iOS builds the build cloud uses Macs running xcode (the native Apple tool) to build the app.



Codename One doesn't send source code to the build cloud, only compiled bytecode!

Notice that Codename One also provides an option to build offline which means corporations that have policies forbidding such cloud architectures can still use Codename One with some additional overhead/complexity of setting up the native build tools. Since Codename One is open source some developers use the source code to compile applications offline but that's outside the scope of this book.

1.1.1. Why Build Servers?

The build servers allow building native iOS Apps without a Mac and native Windows apps without a Windows machine. They remove the need to install/update complex toolchains and simplify the process of building a native app to a right click.

E.g.: Since building native iOS applications requires a Mac OS X machine with a recent version of xcode Codename One maintains such machines in the cloud. When developers send an iOS build such a Mac will be used to generate C source code using [ParparVM](https://github.com/codenameone/CodenameOne/tree/master/vm) [<https://github.com/codenameone/CodenameOne/tree/master/vm>] and it will then compile the C source code using xcode & sign the resulting binary using xcode. You can install the binary to your device or build a distribution binary for the appstore. Since C code is generated it also means that your app will be "future proof" in a case of changes from Apple. You can also inject Objective-C native code into the app while keeping it 100% portable thanks to the "native interfaces" capability of Codename One.

Subscribers can receive the C source code back using the include sources feature of Codename One and use those sources for benchmarking, debugging on devices etc.

The same is true for most other platforms. For the Android, J2ME & Blackberry the standard Java code is executed as is.

Java 8 syntax is supported thru [retrolambda](https://github.com/orfjackal/retrolambda) [<https://github.com/orfjackal/retrolambda>] installed on the Codename One servers. This is used to convert bytecode seamlessly down to Java 5 syntax levels. Java 5 syntax is translated to the JDK 1.3 cldc subset on J2ME/Blackberry to provide those language capabilities and API's across all devices. This is done using a server based bytecode processor based on retroweaver and a great deal of custom code. Notice that this architecture is transparent to developers as the build servers abstract most of the painful differences between devices.

1.1.2. How does Codename One Work?

Codename One uses a SaaS based approach so the information in this appendix might (and probably will) change in the future to accommodate improved architectures. I included this information for reference only, you don't need to understand this in order to follow the content of the book...

Since Android is already based on Java, Codename One is already native to Android and “just works” with the Android VM (ART/Dalvik).

On iOS, Codename One built and open sourced ParparVM, which is a very conservative VM. ParparVM features a concurrent (non-blocking) GC and it's written entirely in Java/C. ParparVM generates C source code matching the given Java bytecode. This effectively means that an xcode project is generated and compiled on the build servers. It's as if you handcoded a native app and is thus “future proof” for changes that Apple might introduce. E.g. Apple migrated to 64bit and later introduced bitcode support to iOS. ParparVM needed no modifications to comply with those changes.



Codename One translates the bytecode to C which is faster than Swift/Objective-C.
The port code that invokes iOS API's is hand coded in Objective-C

For Windows 10 desktop and Mobile support, Codename One uses iKVM to target UWP (Universal Windows Platform) and has open sourced the changes to the original iKVM code.

JavaScript build targets use TeaVM to do the translation statically. TeaVM provides support for threading using JavaScript by breaking the app down in a rather elaborate way. To support the complex UI Codename One uses the HTML5 Canvas API which allows absolute flexibility for building applications.

For desktop builds Codename One uses javapackager, since both Macs and Windows machines are available in the cloud the platform specific nature of javapackager is not a problem.

Lightweight Architecture

What makes Codename One stand out is the approach it takes to UI: “lightweight architecture”.

Lightweight architecture is the “not so secrete sauce” to Codename One’s portability. Essentially it means all the components/widgets in Codename One are written in Java. Thus their behavior is consistent across all platforms and they are fully customizable from the developer code as they don’t rely on OS internal semantics. This allows developers to preview the application accurately in the simulators and GUI builders.

One of the big accomplishments in Codename One is its unique ability to embed “heavyweight” widgets into place among the “lightweights”. This is crucial for apps such as Uber where the cars and widgets on top are implemented as Codename One components yet below them we have the native map component.

Codename One achieves fast performance by drawing using the native gaming API's of most platforms e.g. OpenGL ES on iOS. The core technologies behind Codename One are all open source including most of the stuff developed by Codename One itself, e.g. ParparVM but also the full library, platform ports, designer tool, device skins etc.

Lightweight Architecture Origin

Lightweight components date back to Smalltalk frameworks, this notion was popularized in the Java world by Swing. Swing was the main source of inspiration to Codename One's predecessor LWUIT. Many frameworks took this approach over the years including JavaFX and most recently Ionic in the JavaScript world.

Why ParparVM

On iOS, Codename One uses [ParparVM](https://github.com/codenameone/CodenameOne/tree/master/vm) [<https://github.com/codenameone/CodenameOne/tree/master/vm>] which translates Java bytecode to C code and boasts a non-blocking GC as well as 64 bit/bitcode support. This VM is fully open source in the [Codename One git repository](https://github.com/codenameone/CodenameOne/) [<https://github.com/codenameone/CodenameOne/>]. In the past Codename One used [XMLVM](http://www.xmlvm.org/) [<http://www.xmlvm.org/>] to generate native code in a very similar way but the XMLVM solution was too generic for the needs of Codename One. [ParparVM](https://github.com/codenameone/CodenameOne/tree/master/vm) [<https://github.com/codenameone/CodenameOne/tree/master/vm>] boasts a unique architecture of translating code to C (similarly to XMLVM), because of that Codename One is the only solution of its kind that can **guarantee** future iOS compatibility since the officially supported iOS toolchain is always used instead of undocumented behaviors.



XMLVM could guarantee that in theory but it is no longer maintained

The key advantages of ParparVM over other approaches are:

- **Truly Native**—since code is translated to C rather than directly to ARM or LLVM code the app is "more native". It uses the official tools and approaches from Apple and can benefit from their advancements e.g. latest bitcode changes or profiling capabilities.
- **Smaller Class Library**—ParparVM includes a very small segment of the full JavaAPI's resulting in final binaries that are smaller than the alternatives by orders of magnitude. This maps directly to performance and memory overhead.
- **Simple and Extensible**—to work with ParparVM you need a basic understanding of C. This is crucial for the fast moving world of mobile development, as Apple changes things left and right we need a more agile VM.

Windows Phone/UWP

In the past Codename One had 2 major Windows VM port rewrites and 3 or 4 rendering pipelines within those ports (depends on how you would define a "rewrite").



The old Windows Phone port was deprecated and is no longer supported, the UWP port is the only supported Windows mobile target

Codename One now targets UWP by leveraging a [modified version of iKVM](https://github.com/shannah/cn1-ikvm-uwp) [<https://github.com/shannah/cn1-ikvm-uwp>] to build native Windows Universal Applications.

iKVM uses a bytecode to CLR translation process that effectively converts Java bytecode directly to the .net equivalent. This is paired with a port of the Codename One API's that was built for the UWP environment. The UWP port generates native Windows 10 applications that can support ARM

Windows devices natively as well as desktops etc. These binaries can be uploaded directly to Microsofts online store without special processing.

JavaScript Port

The JavaScript port of Codename One is based on the amazing work of the [TeaVM project](http://teavm.org) [<http://teavm.org>]. The team behind TeaVM effectively built a JVM that translates Java bytecode into JavaScript source code while maintaining threading semantics using a very imaginative approach.

The JavaScript port allows unmodified Codename One applications to run within a desktop or mobile browser. The port itself is based on the HTML5 Canvas API, this provides a pixel perfect implementation of the Codename One API.



The JavaScript port is only available for Enterprise grade subscribers of Codename One

Desktop and Android

The other ports of Codename One use the VM's available on the host machines/environments to execute the runtime. [Retrolambda](https://github.com/orfjackal/retrolambda) [<https://github.com/orfjackal/retrolambda>] is used to provide Java 8 language features in a portable way.

The Android port uses the native Android tools including the gradle build environment in the latest versions.

The desktop port creates a standard JavaSE application which is packaged with the JRE and an installer.



The Desktop port is only available to pro grade subscribers of Codename One

1.1.3. Versions In Codename One

One of the confusing things about Codename One is the versions. Since Codename One is a SaaS product versioning isn't as simple as a 2.x or 3.x moniker. However, to conform to this convention Codename One does make versioned releases which contribute to the general confusion.

When a version of Codename One is released the version number refers to the libraries at the time of the release. These libraries are then frozen and are made available to developers who use the [Versioned Builds](https://www.codenameone.com/how-do-i---get-repeatable-builds-build-against-a-consistent-version-of-codename-one-use-the-versioning-feature.html) [<https://www.codenameone.com/how-do-i---get-repeatable-builds-build-against-a-consistent-version-of-codename-one-use-the-versioning-feature.html>] feature. The plugin, which includes the designer as well as all development that is unrelated to versioned builds continues with its regular updates immediately after release. The same is true for the build servers that move directly to their standard update cycle.

1.2. History



Figure 1. LWUIT App Screenshot circa 2007

Codename One was started by Chen Fishbein and Shai Almog who authored the Open Source LWUIT project at Sun Microsystems (circa 2007). The LWUIT project aimed to solve the fragmentation within J2ME/Blackberry devices by creating a higher standard of user interface than the common baseline at the time. LWUIT received critical acclaim and traction within multiple industries but was limited by the declining feature phone market. It was forked by several companies including Nokia. It was used as the base standard for DTV in Brazil. Another fork has brought a LWUIT into high end cars from Toyota and other companies. This fork later adapted Codename One as well.

In 2012 Shai and Chen formed Codename One as they left Oracle. The project has taken many of the basic concepts developed within the LWUIT project and adapted them to the smartphone world which is still experiencing similar issues to the device fragmentation of the old J2ME phones.

1.3. Core Concepts of Mobile Programming

Before we proceed I'd like to explain some universal core concepts of mobile programming that might not be intuitive. These are universal concepts that apply to mobile programming regardless of the tools you are using.

You can skip this section if you feel you are familiar enough with the core problems/issues in mobile app development.

1.3.1. Density

Density is also known as DPI (Dots Per Inch) or PPI (pixels or points per inch). Density is confusing, unintuitive and might collide with common sense. E.g. an iPhone 7 plus has a resolution of **1080x1920** pixels and a PPI of **401** for a 5 inch screen. On the other hand an iPad 4 has **1536x2048** pixels with a PPI of **264** on a **9.7** inch screen... Smaller devices can have higher resolutions!

As the following figure shows, if a Pixel 2 XL had pixels the size of an iPad it would have been twice the size of that iPad. While in reality it's nearly half the height of the iPad!

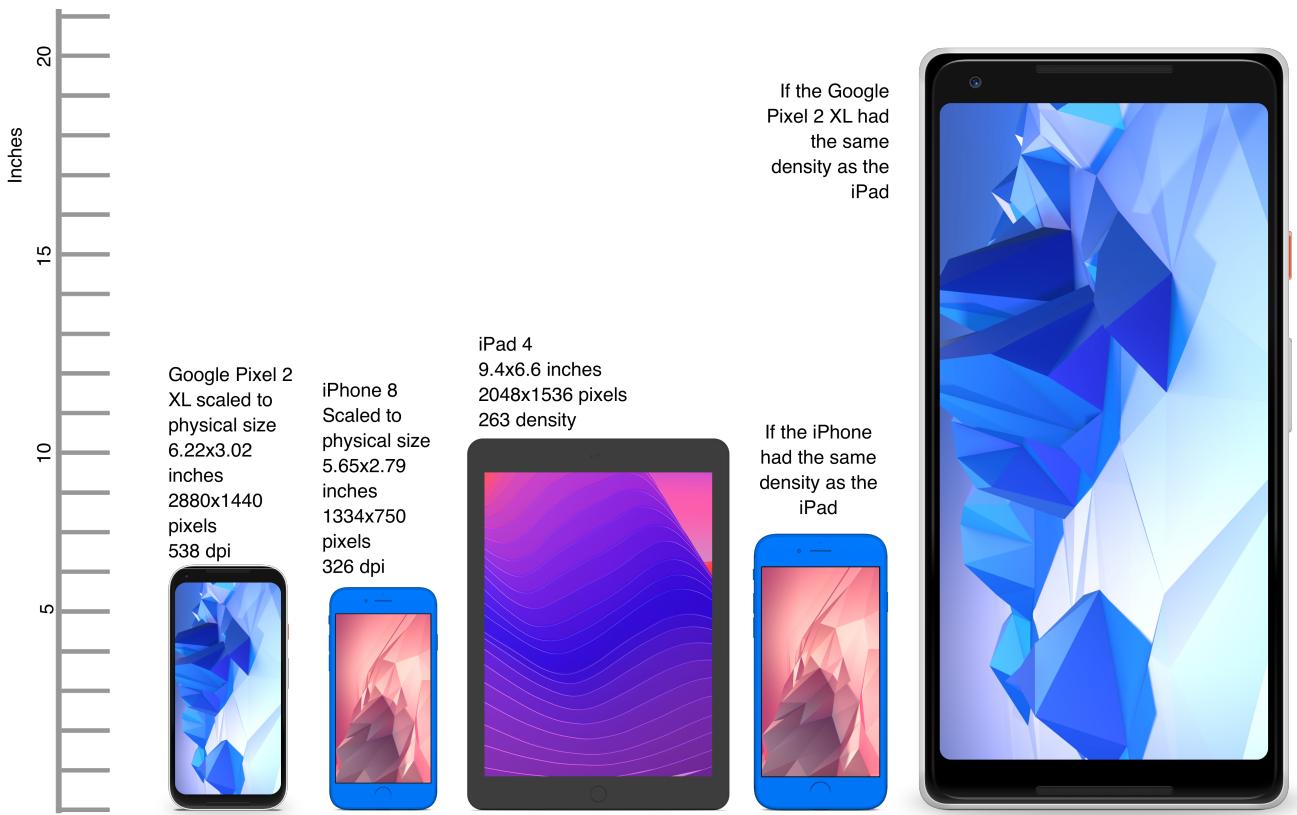
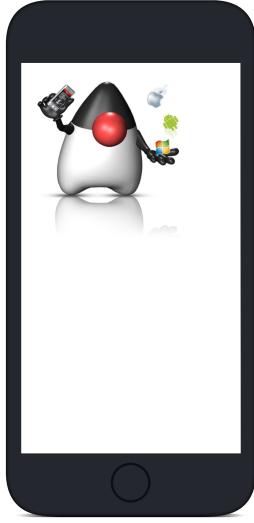


Figure 2. Device Density vs. Resolution

Differences in density can be extreme. A second generation iPad has 132 PPI, where modern phones have PPI that crosses the 600 mark. Low resolution images on high PPI devices will look either small or pixelated. High resolution images on low PPI devices will look huge, overscaled (artifacts) and will consume too much memory.

iPhone 8 Sized to Scale



iPad 4 Sized to Scale



Google Pixel 2 XL
Sized to Scale



Figure 3. How the Same Image Looks in Different Devices

The exact same image will look different on each device, sometimes to a comical effect. One of the solutions for this problem is multi-images. All OS's support the ability to define different images for various densities. I will discuss multi-images later in Chapter 2.

This also highlights the need for working with measurements other than pixels. Codename One supports millimeters (or dips) as a unit of measurement. This is highly convenient and is a better representation of size when dealing with mobile devices.

But there is a bigger conceptual issue involved. We need to build a UI that adapts to the wide differences in form factors. We might have fewer pixels on an iPad but because of its physical size we would expect the app to cram more information into that space so the app won't feel like a blown up phone application. There are multiple strategies to address that but one of the first steps is in the layout managers.

I'll discuss the layout managers in depth in Chapter 2 but the core concept is that they decide where a UI element is placed based on generic logic. That way the user interface can adapt automatically to the huge variance in display size and density.

1.3.2. Touch Interface

The fact that mobile devices use a touch interface today isn't news... But the implications of that aren't immediately obvious to some developers.

UI elements need to be finger sized and heavily spaced. Otherwise we risk the "fat finger" effect.

That means spacing should be in millimeters and not in pixels due to device density.

Scrolling poses another challenge in touch based interfaces. In desktop applications it's very common to nest scrollable items. However, in touch interfaces the scrolling gesture doesn't allow such nuance. Furthermore, scrolling on both the horizontal and vertical axis (side scrolling) can be very inconvenient in touch based interfaces.

1.3.3. Device Fragmentation

Some developers single out this wide range of resolutions and densities as “device fragmentation”. While it does contribute to development complexity for the most part it isn't a difficult problem to overcome.

Densities aren't the cause of device fragmentation. Device fragmentation is caused by multiple OS versions with different behaviors. This is very obvious on Android and for the most part relates to the slow rollout of Android vendor versions compared to Googles rollout. E.g. 7 months after the Android 8 (Oreo) release in 2018 it was still available on 1.1% of the devices. The damning statistic is that 12% of the devices in mid 2018 run Android 4.4 Kitkat released in 2013!

This makes QA difficult as the disparity between these versions is pretty big. These numbers will be out of date by the time you read this but the core problem remains. It's hard to get all device manufacturers on the same page so this problem will probably remain in the foreseeable future despite everything.

1.3.4. Performance

Besides the obvious need for performance and smooth animation within a mobile app there are a couple of performance related issues that might not be intuitive to new developers: size and power.

App Size

Apps are installed and managed via stores. This poses some restrictions about what an app can do. But it also creates a huge opportunity. Stores manage automatic update and to some degree the marketing/monetization of the app.

A good mobile app is updated once a month and sometimes even once a week. Since the app downloads automatically from the store this can be a huge benefit:

- Existing users are reminded of the app and get new features instantly
- New users notice the app featured on a “what's new” list

If an app is big it might not update over a cellular network connection. Google and Apple have restrictions on automatic updates over cellular networks to preserve battery life and data plans. A large app might negatively impact users perception of the app and trigger uninstalls e.g. when a phone is low on available space.

Power Drain

Desktop developers rarely think about power usage within their apps. In mobile development this is a crucial concept. Modern device OS's have tools that highlight misbehaving applications and this

can lead to bad reviews.

Code that loops forever while waiting for input will block the CPU from sleeping and slowly drain the battery.

Worse. Mobile OS's kill applications that drain the battery. If the app is draining the battery and is minimized (e.g. during an incoming call) the app could be killed. This will impact app performance and usability.

1.3.5. Sandbox and Permissions

Apps installed on the device are “sandboxed” to a specific area so they won't harm the device or its functionality. The filesystem of mobile applications is restricted so one application can't access the files of another application. Things that most developers take for granted on the desktop such as a “file picker” or accessing the image folder don't work on devices!

This means that when your application works on a file it belongs only to your application. In order to share the file with a different application you need to ask the operating system to do that for you.

Furthermore, some features require a “permission” prompt and in some cases require special flags in system files. Apps need to request permission to use sensitive capabilities e.g. Camera, Contacts etc.

Historically Android developers just declared required permissions for an app and the user was prompted with permissions during install. Android 6 adopted the approach used by iOS of prompting the user for permission when accessing a feature.

This means that in runtime a user might revoke a permission. A good example in the case of an Uber app is the location permission. If a user revokes that permission the app might lose its location.

1.4. Installing Codename One



The minimum JDK for Codename One is JDK 8



The following screenshots are from Mac OS but the process should work exactly the same on Windows and Linux

1.5. IntelliJ/IDEA

Codename One recommends IntelliJ/IDEA 2016 or newer.



Codename One **doesn't support** Android Studio! You can use IntelliJ/IDEA community edition instead

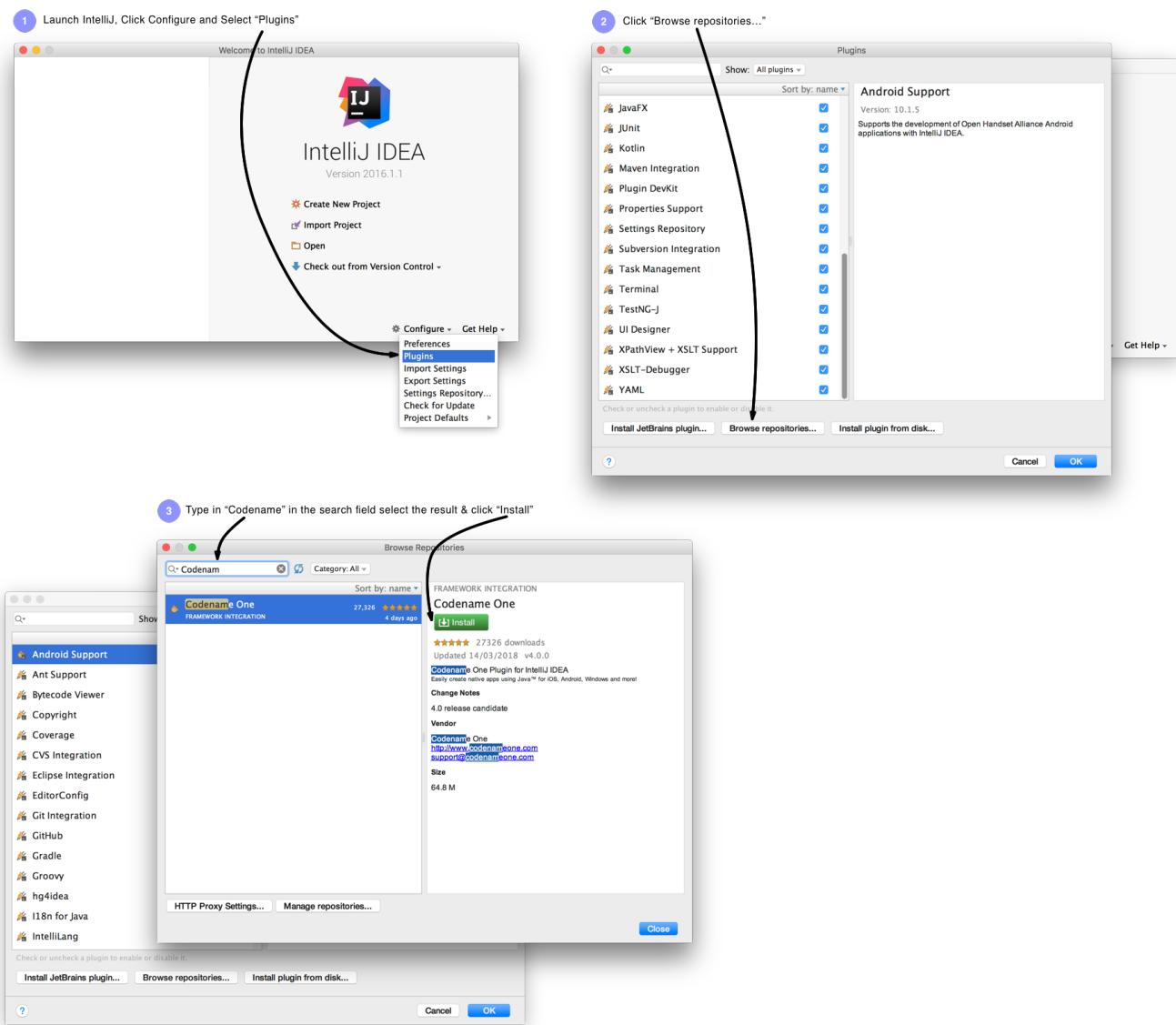


Figure 4. IntelliJ Installation Instructions

1.6. NetBeans

NetBeans install is pretty simple although the default “plugin center” for NetBeans is notoriously unreliable. That’s why we recommend using the Codename One plugin center: <https://www.codenameone.com/files/netbeans/updates.xml>

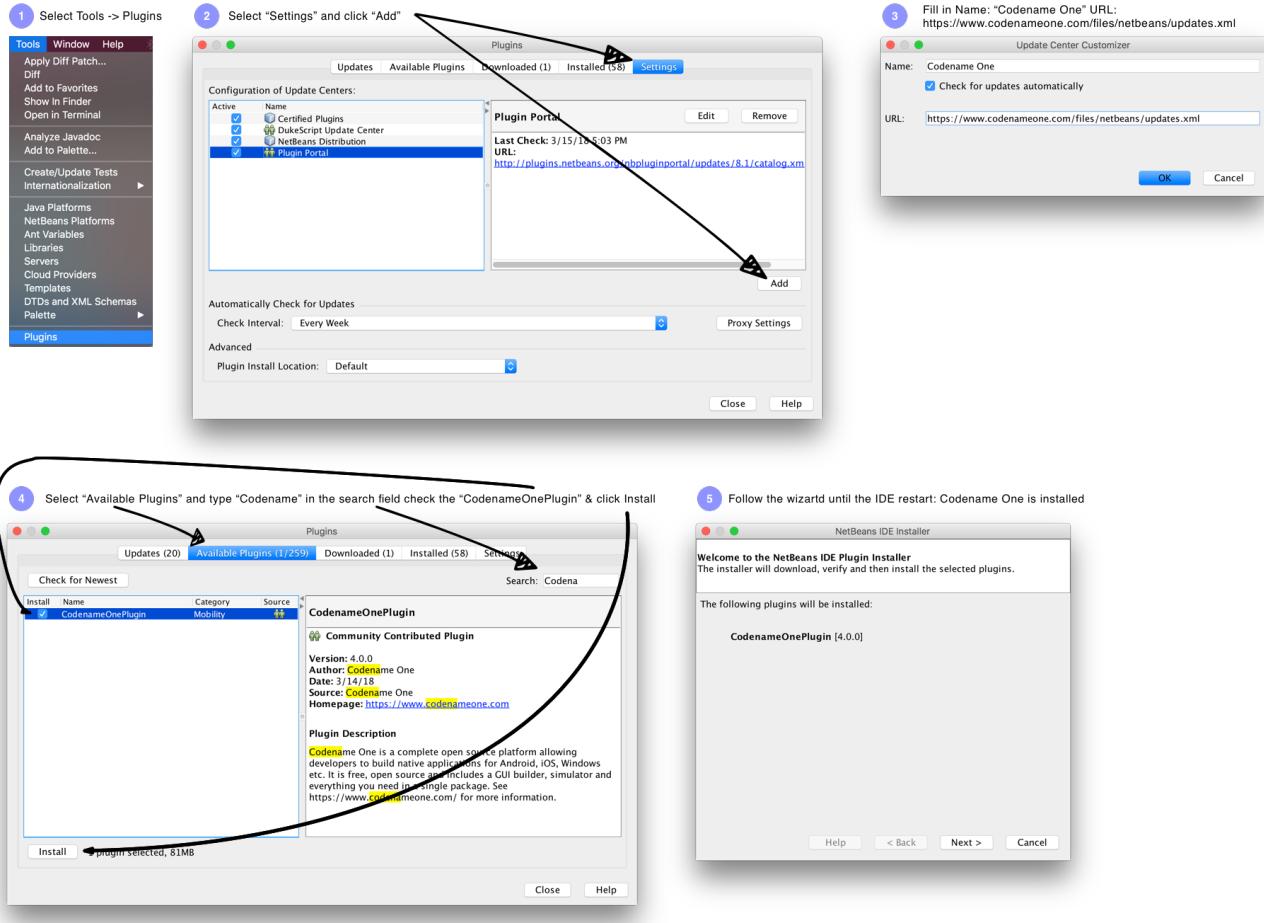


Figure 5. NetBeans Installation Instructions



Make sure you are using a NetBeans version that includes Java support, don't download a version for Ruby/PHP or J2ME and make sure the IDE runs on top of JDK 8

1.7. Eclipse

Codename One supports Eclipse Neon 2 or newer. There are a few pitfalls that can happen with an Eclipse install specifically when other JVM versions are installed on your machine.



If you are new to Java, Eclipse might be intimidating. It's a very powerful IDE but its configuration is rough

Make sure your `JAVA_HOME` environment variable points at JDK 8 and that the path to the JDK 8 `bin` directory is first in the PATH statement. If all else fails edit the `eclipse.ini` file to force Eclipse to use your JDK 8 install. See this site for help with editing the `eclipse.ini` file: <https://wiki.eclipse.org/Eclipse.ini>

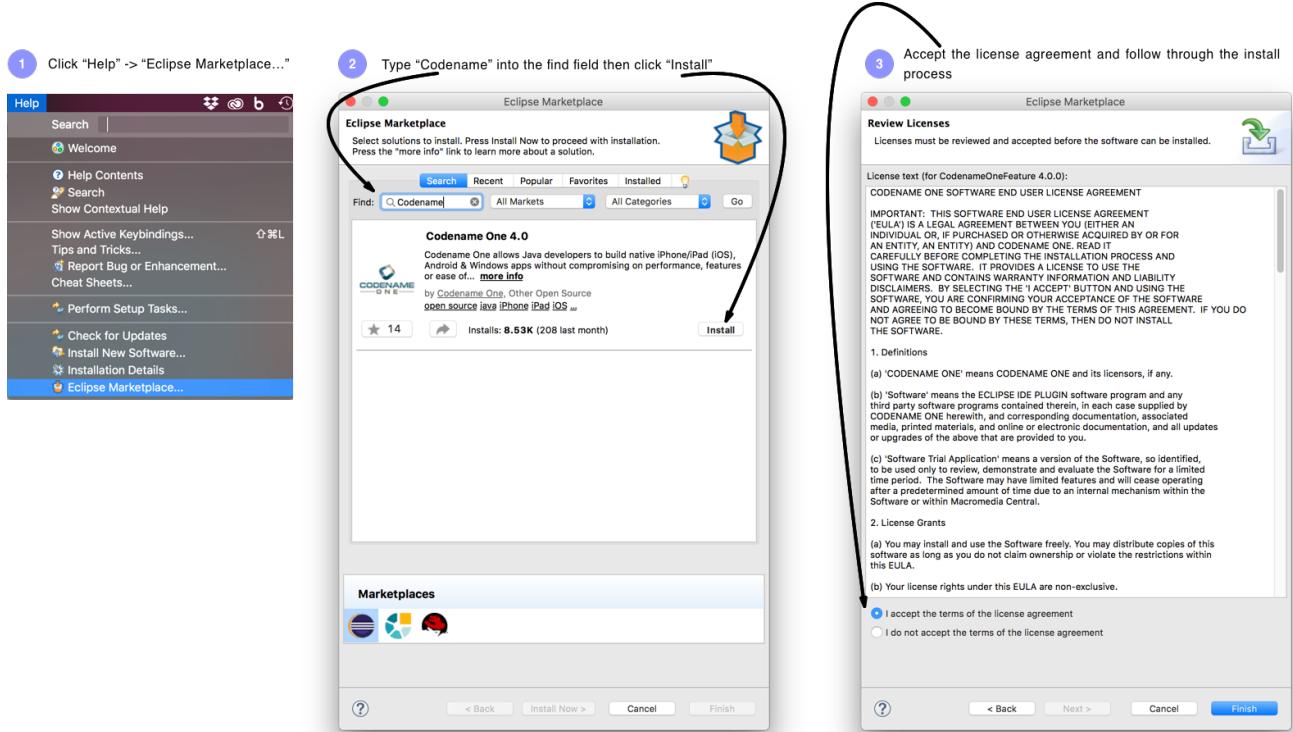


Figure 6. Eclipse Installation Instructions



In order to run the app in Eclipse make sure to select the `.launch` file in Eclipse

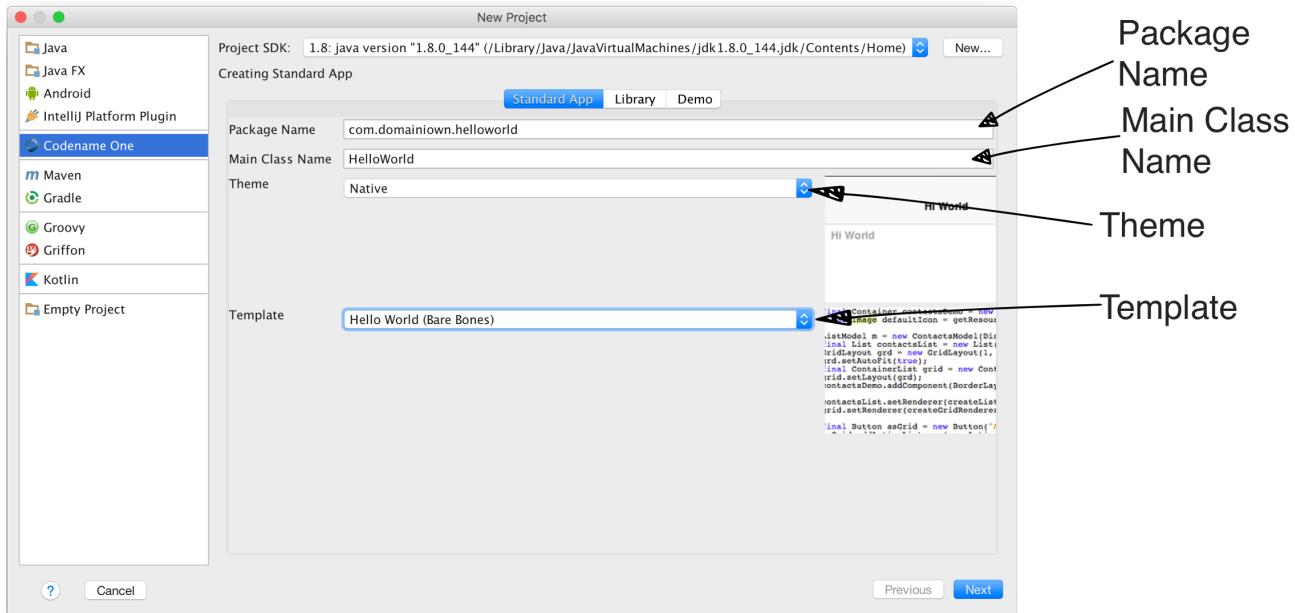
1.8. Hello World Application

Before we get to the code there are few important things we need to go over with the new project wizard.

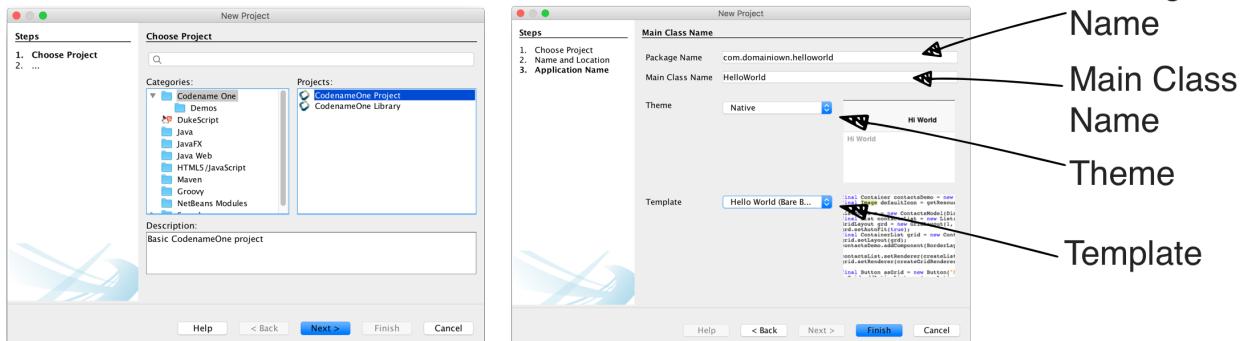
We need to create a new project. We need to pick a project name and I'll leave that up to you although it's hard to go wrong with `HelloWorld`. The following four values are important:

- **App Name** - This is the name of the app and the main class, it's important to get this right as it's hard to change this value later
- **Package Name** - It's **crucial** you get this value right. Besides the difficulty of changing this after the fact, once an app is submitted to iTunes/Google Play with a specific package name this can't be changed! See the sidebar "Picking a Package Name".
- **Theme** - There are various types of builtin themes in Codename One, for simplicity I pick `Native` as it's a clean slate starting point
- **Template** - There are several builtin app templates that demonstrate various features, for simplicity I always pick `Bare Bones` which includes the bare minimum

IntelliJ



NetBeans



Eclipse

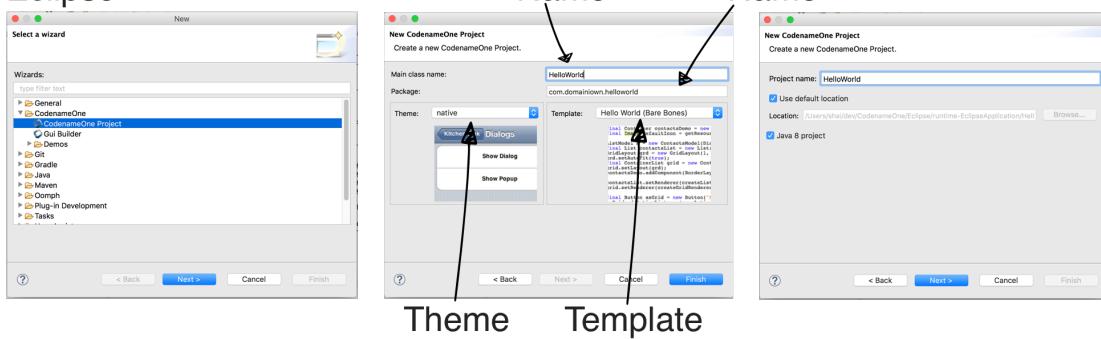


Figure 7. The New App Wizard

Picking a Package Name

Apple, Google and Microsoft identify applications based on their package names. If you use a domain that you don't own it's possible that someone else will use that domain and collide with you. In fact some developers left the default `com.mycompany` domain in place all the way into production in some cases.

This can cause difficulties when submitting to Apple, Google or Microsoft. Submitting to one of them is no guarantee of success when submitting to another.

To come up with the right package name use a reverse domain notation. So if my website is `goodstuff.co.uk` my package name should start with `uk.co.goodstuff`. I highly recommend the following guidelines for package names:

- **Lower Case** - some OS's are case sensitive and handling a mistake in case is painful. The Java convention is lower case and I would recommend sticking to that although it isn't a requirement
- **Avoid Dash and Underscore** - You can't use a dash character (-) for a package name in Java. Underscore (_) doesn't work for iOS. If you want more than one word just use a deeper package e.g.: `com.mydomain.deeper.meaningful.name`
- **Obey Java Rules** - A package name can't start with a number so you can't use `com.mydomain.1sler`. You should avoid using Java keywords like `this`, `if` etc.
- **Avoid Top Level** - instead of using `uk.co.goodstuff` use `uk.co.goodstuff.myapp`. That would allow you to have more than one app on a domain

1.8.1. Running

We can run the `HelloWorld` application by pressing the `Play` or `Run` button in the IDE for NetBeans or IntelliJ. In Eclipse we first need to select the simulator `.launch` file and then press run. When we do that the Codename One simulator launches. You can use the menu of the simulator to control and inspect details related to the device. You can rotate it, determine its location in the world, monitor networking calls etc.

With the `Skins` menu you can download device skins to see how your app will look on different devices.



Some skins are bigger than the screen size, uncheck the `Scrollable` flag in the `Simulator` menu to handle them more effectively

Debug works just like `Run` by pressing the IDE's debug button. It allows us to launch the simulator in debug mode where we can set breakpoints, inspect variables etc.

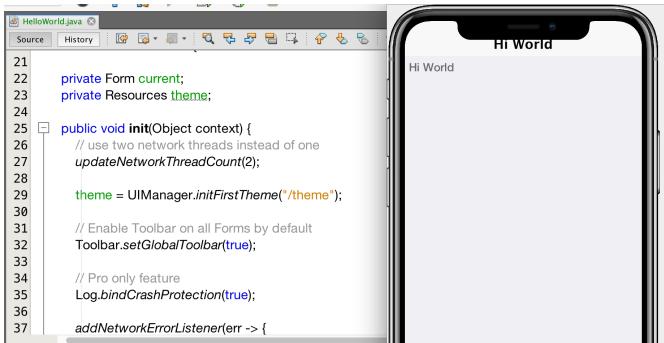


Figure 8. *HelloWorld* Running on the Simulator with an iPhone X Skin

Simulator vs. Emulator

Codename One ships with a simulator similarly to the iOS toolchain which also has a simulator. Android ships with an emulator. Emulators go the extra mile. They create a virtual machine that's compatible with the device CPU and then boot the full mobile OS within that environment. This provides an accurate runtime environment but is **painfully slow**.

Simulators rely on the fact that OS's are similar and so they leave the low level details in place and just map the API behavior. Since Codename One relies on Java it can start simulating on top of the virtual machine on the desktop. That provides several advantages including fast development cycles and full support for all the development tools/debuggers you can use on the desktop.

Emulators make sense for developers who want to build OS level services e.g. screensavers or low level services. Standard applications are better served by simulators.

1.8.2. The Source Code Of The Hello World App

After clicking finish in the new project wizard we have a **HelloWorld** project with a few default settings. I'll break the class down to small pieces and explain each piece starting with the enclosing class:

Listing 1. HelloWorld Class

```
public class HelloWorld { ①
    private Form current; ②
    private Resources theme; ③

    // ... class methods ...
}
```

- ① This is the main class, it's the entry point to the app, notice it doesn't have a **main** method but rather callback which we will discuss soon
- ② Forms are the “top level” UI element in Codename One. Only one **Form** is shown at a time and everything you see on the screen is a child of that **Form**
- ③ Every app has a theme, it determines how everything within the application looks e.g. colors, fonts etc.

Next let's discuss the first lifecycle method `init(Object)`. I discuss the lifecycle in depth in the [Application Lifecycle Sidebar](#).

Listing 2. HelloWorld init(Object)

```
public void init(Object context) { ①
    updateNetworkThreadCount(2); ②
    theme = UIManager.initFirstTheme("/theme"); ③
    Toolbar.setGlobalToolbar(true); ④
    Log.bindCrashProtection(true); ⑤
    addNetworkErrorListener(err -> { ⑥
        err.consume(); ⑦
        if(err.getError() != null) { ⑧
            Log.e(err.getError());
        }
        Log.sendLogAsync(); ⑨
        Dialog.show("Connection Error", ⑩
            "There was a networking error in the connection to " +
            err.getConnectionRequest().getUrl(), "OK", null);
    });
}
```

① `init` is the first of the four lifecycle methods. It's responsible for initialization of variables and values

② By default Codename One has one thread that performs all the networking, we set the default to two which gives better performance

③ The theme determines the appearance of the application. We'll discuss this in the next chapter

④ This enables the `Toolbar` API by default, it allows finer control over the title bar area

⑤ Crash protection automatically sends device crash logs through the cloud

⑥ In case of a network error the code in this block would run, you can customize it to handle networking errors effectively

⑦ `consume()` swallows the event so it doesn't trigger other alerts, it generally means "we got this"

⑧ Not all errors include an exception, if we have an exception we can log it with this code

⑨ This will email the log from the device to you if you have a pro subscription

⑩ This shows an error dialog to the user, in production you might want to remove that code

`init(Object)` works as a constructor to some degree. We recommend avoiding the constructor for the main class and placing logic in the `init` method instead. This isn't crucial but we recommend it since the constructor might happen too early in the application lifecycle.

In a cold start `init(Object)` is invoked followed by the `start()` method. However, `start()` can be invoked more than once if an app is minimized and restored, see the sidebar [Application Lifecycle](#):

Listing 3. HelloWorld start()

```
public void start() {  
    if(current != null){ ①  
        current.show(); ②  
        return;  
    }  
    Form hi = new Form("Hi World", BoxLayout.y()); ③  
    hi.add(new Label("Hi World")); ④  
    hi.show(); ⑤  
}
```

- ① If the app was minimized we usually don't want to do much, just show the last **Form** of the application
- ② **current** is a **Form** which is the top most visual element. We can only have one **Form** showing and we enforce that by using the **show()** method
- ③ We create a new simple **Form** instance. It has the title "Hello World" and arranges elements vertically (on the Y axis)
- ④ We add another **Label** below the title, see figure [\[TitleAndLabelImage\]](#). We will discuss component hierarchy later
- ⑤ The **show()** method places the **Form** on the screen. Only one **Form** can be shown at a time



Figure 9. Title and Label in the UI

There are some complex ideas within this short snippet which I'll address later in this chapter when talking about layout. The gist of it is that we create and show a **Form**. **Form** is the top level UI element, it takes over the whole screen. We can add UI elements to that **Form** object, in this case the **Label**. We use the **BoxLayout** to arrange the elements within the **Form** from top to the bottom vertically.

Application Lifecycle

A few years ago Romain Guy (a senior Google Android engineer) was on stage at the Google IO conference. He asked for a show of hands of people who understand the **Activity** lifecycle (**Activity** is similar to a Codename One main class). He then proceeded to jokingly call the audience members who lifted their hands “liars” claiming that after all his years in Google he still doesn’t understand it...

Lifecycle seems simple on the surface but hides a lot of nuance. Android’s lifecycle is ridiculously complex. Codename One tries to simplify this and also make it portable. Sometimes complexity leaks out and the nuances can be difficult to deal with.

Simply explained an application has three states:

- **Foreground** - it’s running and in the foreground which means the user can physically interact with the app
- **Suspended** - the app isn’t in the foreground, it’s either paused or has a background process running
- **Not Running** - the app was never launched, was killed or crashed

The lifecycle is the process of transitioning between these 3 states and the callbacks invoked when such a transition occurs. The first time we launch the app we start from a “Cold Start” (Not Running State) but on subsequent launches the app is usually started from the "Warm Start" (Suspended State).

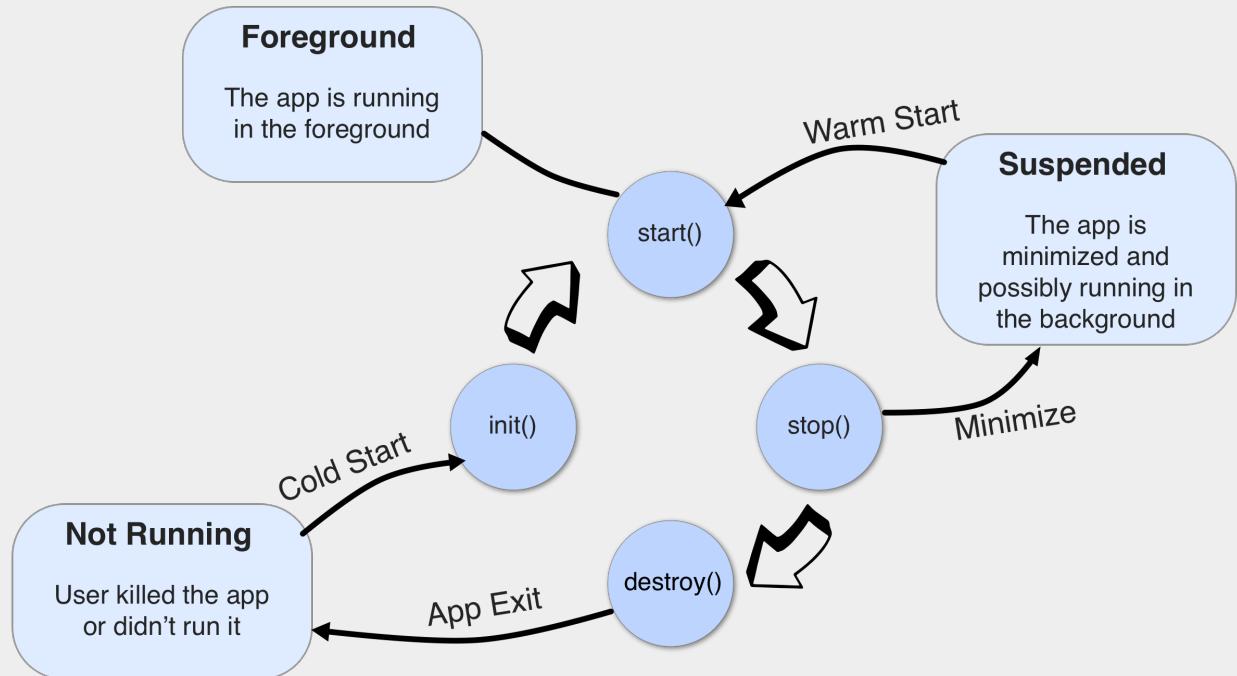


Figure 10. Codename One Application Lifecycle

Codename One has four standard callback methods in the lifecycle API:

- **init(Object)** - is invoked when the app is first launched from a *Not Running* state.

- `start()` - is invoked for two separate cases. After `start()` is finished the app transitions to the *Foreground* state.
 - Following `init(Object)` in case of a cold start. Cold start refers to starting the app from a *Not Running* state.
 - When the app is restored from *Suspended* state. In this case `init(Object)` isn't invoked
- `stop()` - is invoked when the app is minimized e.g. when switching to a different app. After `stop()` is finished the app transitions to the *Suspended* state.
- `destroy()` - is invoked when the app is destroyed e.g. killed by a user in the task manager. After `destroy()` is finished the app is no longer running hence it's in the *Not Running* state.



`destroy()` is optional there is no guarantee that it will be invoked. It should be used only as a last resort

Now that we have a general sense of the lifecycle lets look at the last two lifecycle methods:

Listing 4. HelloWorld stop() and destroy()

```
public void stop() { ①
    current = getCurrentForm(); ②
    if(current instanceof Dialog) { ③
        ((Dialog)current).dispose();
        current = getCurrentForm();
    }
}

public void destroy() { ④
}
```

① `stop()` is invoked when the app is minimized or a different app is opened

② As the app is stopped we save the current `Form` so we can restore it back in `start()` if the app is restored

③ `Dialog` is a bit of a special case restoring a `Dialog` might block the proper flow of application execution so we dispose them and then get the parent `Form`

④ `destroy()` is a very special case. Under normal circumstances you shouldn't write code in `destroy()`. `stop()` should work for most cases

That's it. Hopefully you have a general sense of the code. It's time to run on the device.

1.8.3. Building and Deploying On Devices

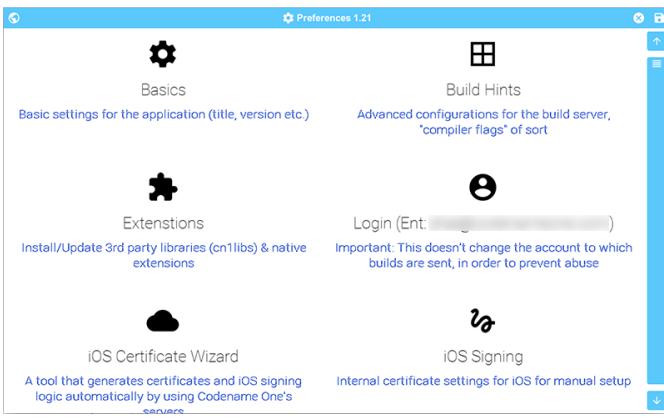


Figure 11. Codename One Settings

You can use the [project settings](#) to configure almost anything. Specifically, the application title, application version, application icon etc. are all found in the right click menu of your IDE.

Just right click your Codename One project icon in any IDE, select **Codename One → Codename One Settings**.

There are many options within this UI that control almost every aspect of the application from signing to basic settings.

Signing/Certificates

All of the modern mobile platforms require signed applications but they all take radically different approaches when implementing it.

Signing is a process that marks your final application for the device with a special value. This value (signature) is a value that only you can generate based on the content of the application and your certificate. Effectively it guarantees the app came from you. This blocks a 3rd party from signing their apps and posing as you to the appstore or to the user. It's a crucial security layer.

A certificate is the tool we use for signing. Think of it as a mathematical rubber stamp that generates a different value each time. Unlike a rubber stamp a signature can't be forged!

Signing on Android



Backup your Android certificate and save its password!

If you lose your Android certificate you will not be able to update your app

Android uses a self signed certificate approach. You can just generate a certificate by describing who you are and picking a password!

Anyone can do that. However, once a certificate is published it can't be replaced...

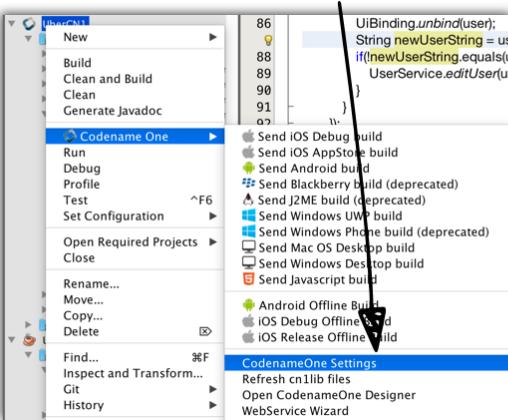
If this wasn't the case someone else could potentially push an "upgrade" to your app. Once an app is submitted with a certificate to Google Play this app can't be updated with any other certificate.

With that in mind generating an Android certificate is trivial.

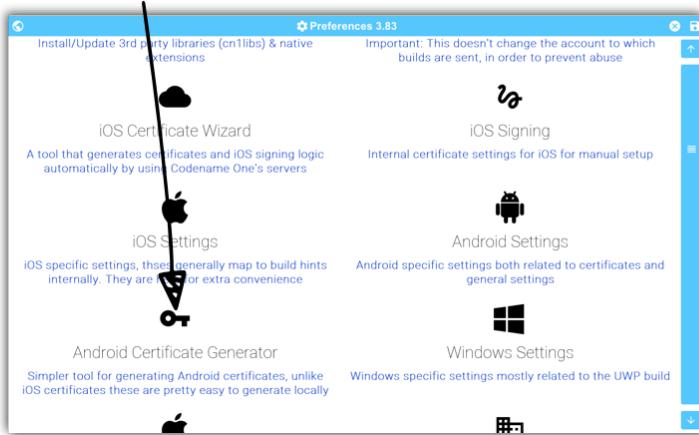


The following chart illustrates a process that's identical on all IDE's

Right click the project & select "Codename One Settings"



Click "Android Certificate Generator"



Don't forget the password

Press OK when you are done



Alias is a simple ID for the certificate e.g. codenameone

The other details will be visible to the users of the app when they inspect your apps signature

Checking this will decrease the likelihood of anyone forging this certificate in the foreseeable future. Notice that the current likelihood is **very** low. The new 512bit certificates only work on Android 4.1 or newer

Figure 12. Process of Certificate Generation for Android

Your certificate will generate into the file **Keychain.ks** in your home directory



Make sure to back that up and the password as losing these can have dire consequences

Should I Use a Different Certificate for Each App?

In theory yes. In practice it's a pain... Keeping multiple certificates and managing them is a pain so we often just use one.

The drawback of this approach occurs when you are building an app for someone else or want to sell the app. Giving away your certificate is akin to giving away your house keys. So it makes sense to have separate certificates for each app.

Signing and Provisioning iOS

Code signing for iOS relies on Apple as the certificate authority. This is something that doesn't exist on Android. iOS also requires provisioning as part of the certificate process and completely separates the process for development/release.

But first let's start with the good news:

- Losing an iOS certificate is no big deal - in fact we revoke them often with no impact on shipping apps

- Codename One has a wizard that hides most of the pain related to iOS signing

In iOS Apple issues the certificates for your applications. That way the certificate is trusted by Apple and is assigned to your Apple iOS developer account. There is one important caveat: You need an iOS Developer Account and Apple charges a 99USD Annual fee for that.



The 99USD price and requirement have been around since the introduction of the iOS developer program for roughly 10 years at the time of this writing. It might change at some point though

Apple also requires a “provisioning profile” which is a special file bound to your certificate and app. This file describes some details about the app to the iOS installation process. One of the details it includes during development is the list of permitted devices.

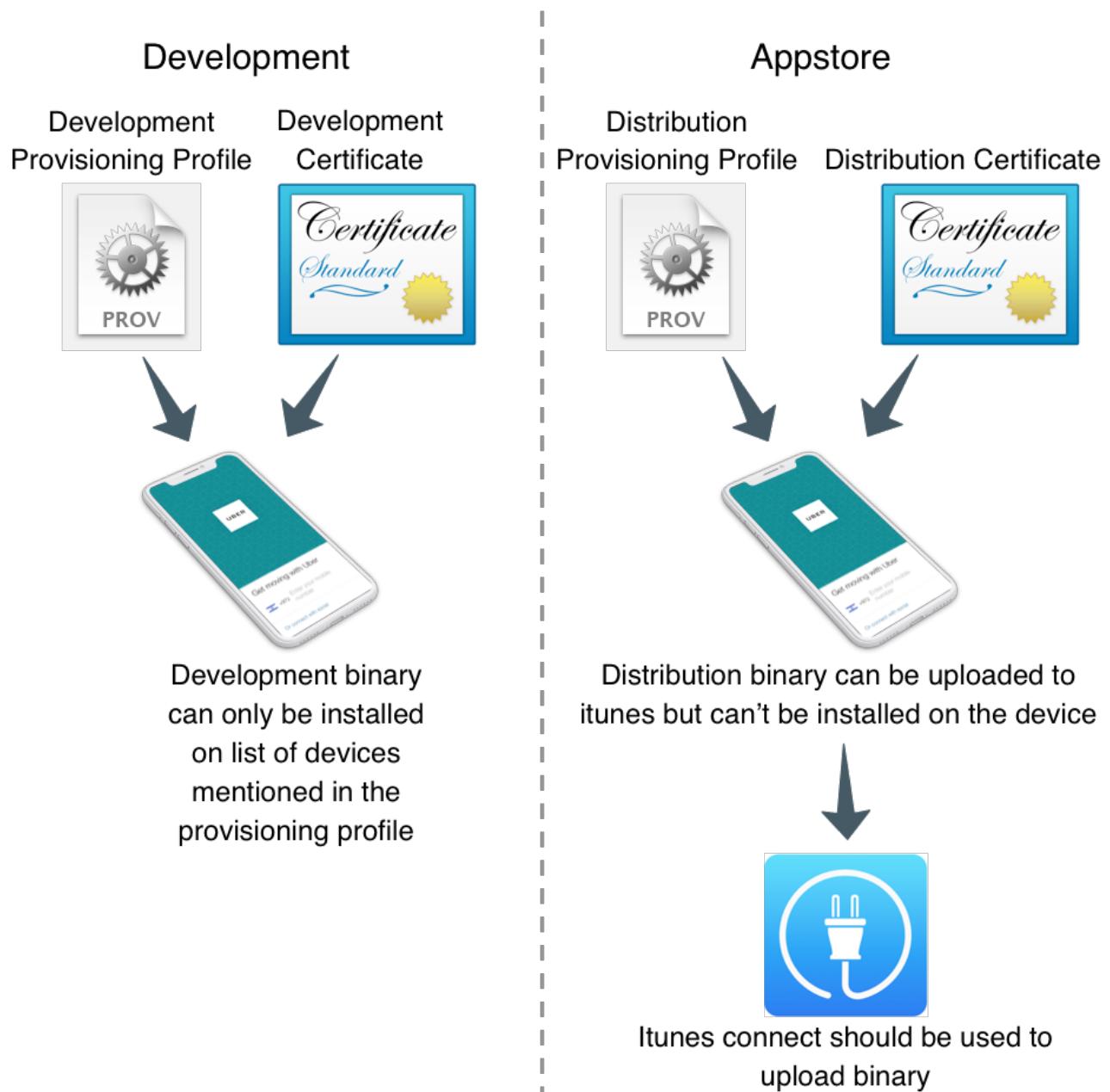


Figure 13. The Four Files Required for iOS Signing and Provisioning

We need 4 files for signing. Two certificates and two provisioning profiles:

- Production**—The production certificate/provisioning pair is used for builds that are uploaded to iTunes
- Development**—The development certificate/provisioning is used to install on your development devices

The certificate wizard automatically creates these 4 files and configures them for you.

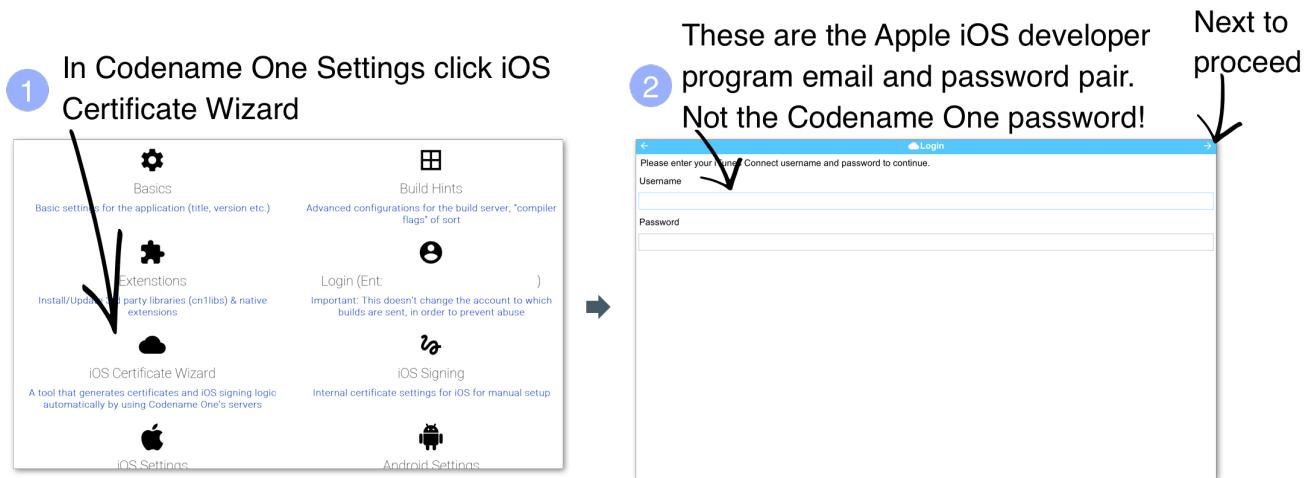


Figure 14. Using the iOS Certificate Wizard Steps 1 and 2

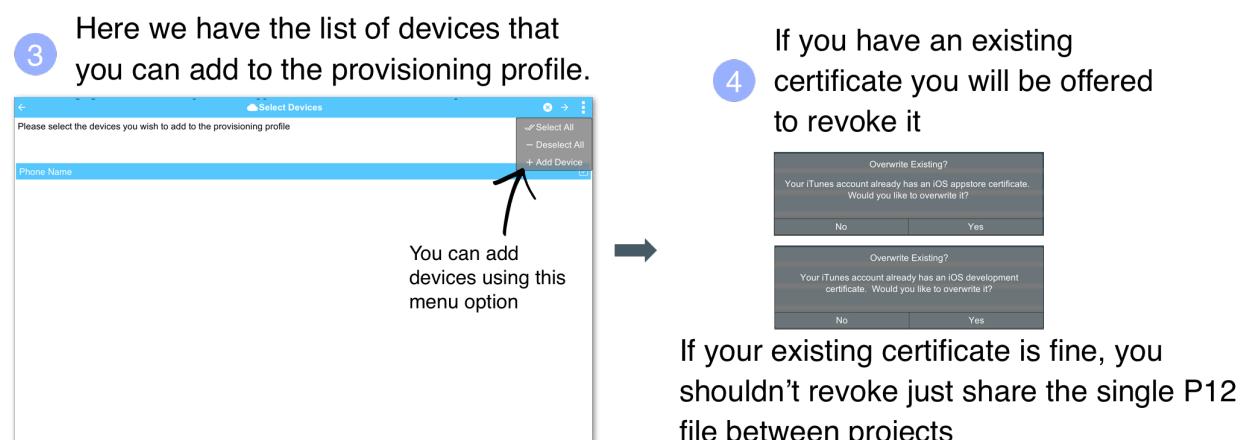
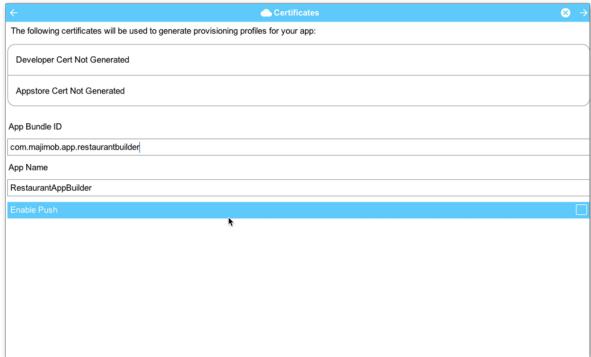


Figure 15. Using the iOS Certificate Wizard Steps 3 and 4

- 5 You are shown the details of the files that should be generated



- 6 The final form shows a summary of what was performed by the wizard

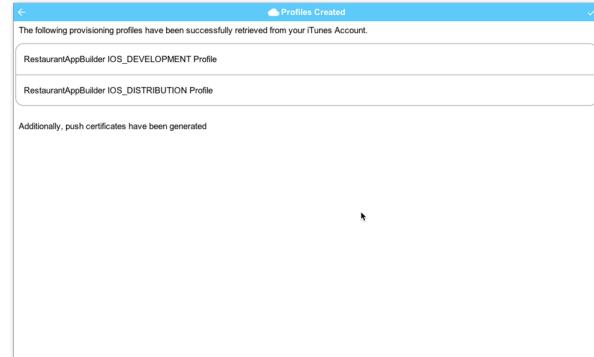


Figure 16. Using the iOS Certificate Wizard Steps 5 and 6



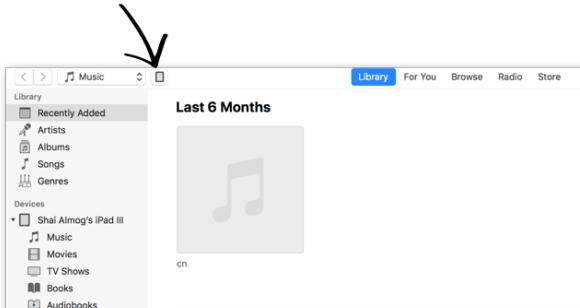
If you have more than one project you should use the same iOS P12 certificate files in all the projects and just regenerate the provisioning. In this situation the certificate wizard asks you if you want to revoke the existing certificate which you shouldn't revoke in such a case. You can update the provisioning profile in Apple's iOS developer website.

One important aspect of provisioning on iOS is the device list in the provisioning step. Apple only allows you to install the app on 100 devices during development. This blocks developers from skipping the appstore altogether. It's important you list the correct UDID for the device in the list otherwise install will fail.

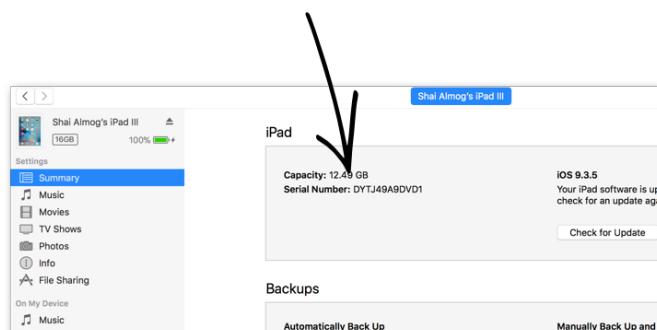


There are several apps and tools that offer the UDID of the device, they aren't necessarily reliable and might give a fake number!

- To get the UDID connect your iDevice to your computer and launch itunes. Then click on the device icon



- Click the serial number of the device



- The serial number turns to the UDID. Notice that this is in the same UI view, the serial number updates to the UDID when you click on it

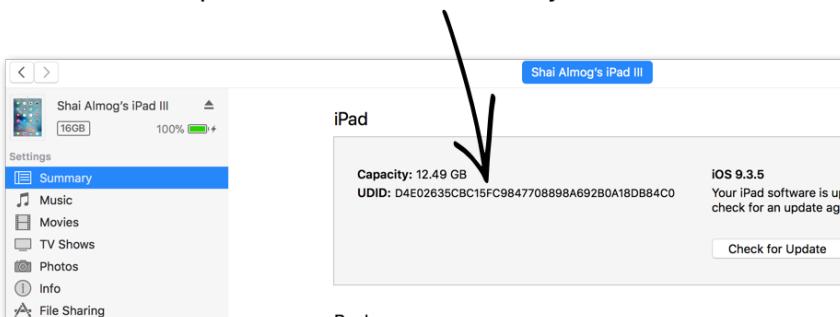


Figure 17. Get the UDID of a Device



You can right click the UDID and select **copy** to copy it

The simplest and most reliable process for getting a UDID is via iTunes. I've used other approaches in the past that worked but this approach is guaranteed.



Ad hoc provisioning allows 1000 beta testers for your application but it's a more complex process that we won't discuss here although it's supported by Codename One

Build and Install

Before we continue with the build we should sign up at <https://www.codenameone.com/build-server.html> where you can soon follow the progress of your builds. You need a Codename One account in order to build for the device.

Now that we have certificates the process of device builds is literally a right click away for both OS's. We can right click the project and select **Codename One** → **Send iOS Debug Build** or **Codename One** → **Send Android Build**.

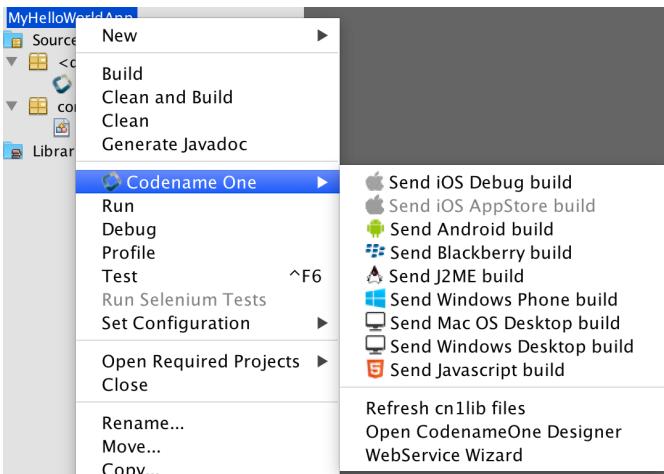


Figure 18. Right click menu options for sending device builds



The first time you send a build you will be prompted for the email and password you provided when signing up for Codename One

Once you send a build you should see the results in the build server page:

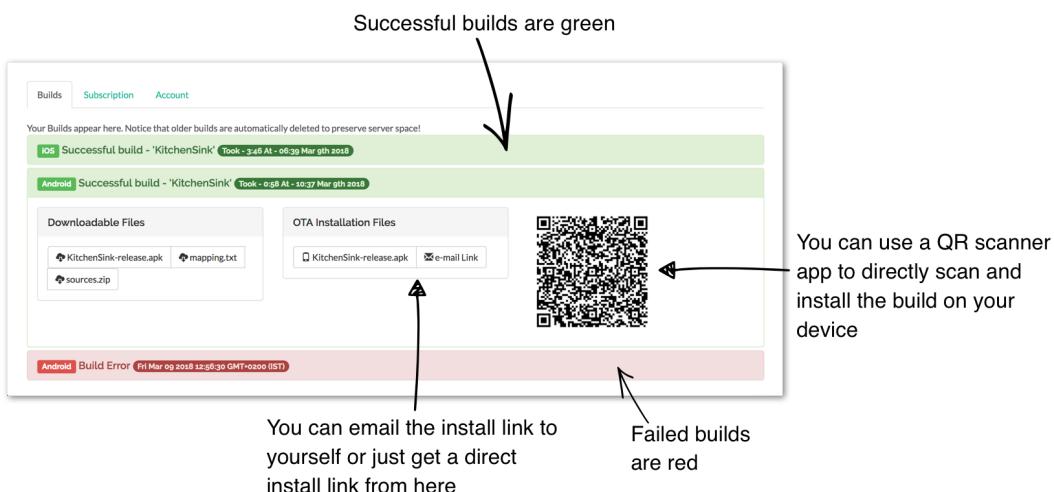


Figure 19. Build Results



On iOS make sure you use Safari when installing, as 3rd party browsers might have issues

Once you go through those steps you should have the **HelloWorld** app running on your device. This process is non-trivial when starting so if you run into difficulties don't despair and seek help at the discussion forum (<https://www.codenameone.com/discussion-forum.html>) or stack overflow (<https://stackoverflow/tags/codenameone/>). Once you go through signing and installation, it becomes easier.



You can also install the application either by emailing the install link to your account (using the **e-mail Link** button)

You can also download the binaries in order to upload them to the appstores.

1.9. Kotlin

Codename One started before Kotlin became public. Kotlin has since shown itself as an interesting option for developers especially within the Android community. With that in mind we decided to integrate support for Kotlin into Codename One.

To use Kotlin with Codename One you will need the following:

- You need to use IntelliJ/IDEA - Currently only the IDEA version of the Codename One plugin has Kotlin support
- You need to install the Kotlin support libraries from the extension manager tool in Codename One Preferences
- Don't use the project conversion tools or accept the warning that the project isn't a Kotlin project. We do our own build process
- Warnings and errors aren't listed correctly and a build that claimed to have 2 errors actually passed...
- This will increase your jar size by roughly 730kb which might make it harder for free tier users

You can install the Kotlin libraries which are required for compilation by using the extension manager.

In the right click menu select **Codename One** → **Codename One Preferences**.

Select **Extensions** type in **kotlin** and install. Then right click the project select **Codename One** → **Refresh Libs**.

1.9.1. Hello Kotlin

Due to the way Kotlin works you can just create a regular Java project and convert sources to Kotlin. You can mix Java and Kotlin code without a problem and Codename One would "just work".

The hello world Java source file looks like this (removed some comments and whitespace):

```

public class MyApplication {
    private Form current;
    private Resources theme;

    public void init(Object context) {
        theme = UIManager.initFirstTheme("/theme");
        Toolbar.setGlobalToolbar(true);
        Log.bindCrashProtection(true);
    }

    public void start() {
        if(current != null){
            current.show();
            return;
        }
        Form hi = new Form("Hi World", BoxLayout.y());
        hi.add(new Label("Hi World"));
        hi.show();
    }

    public void stop() {
        current = getCurrentForm();
        if(current instanceof Dialog) {
            ((Dialog)current).dispose();
            current = getCurrentForm();
        }
    }

    public void destroy() {
    }
}

```

When you select that file and select the menu option **Code** → **Convert Java file to Kotlin File** you should get this:

```

class MyApplication {
    private var current: Form? = null
    private var theme: Resources? = null

    fun init(context: Any) {
        theme = UIManager.initFirstTheme("/theme")
        Toolbar.setGlobalToolbar(true)
        Log.bindCrashProtection(true)
    }

    fun start() {
        if (current != null) {
            current!!.show()
            return
        }
        val hi = Form("Hi World", BoxLayout.y())
        hi.add(Label("Hi World"))
        hi.show()
    }

    fun stop() {
        current = getCurrentForm()
        if (current is Dialog) {
            (current as Dialog).dispose()
            current = getCurrentForm()
        }
    }

    fun destroy() {
    }
}

```

That's pretty familiar. The problem is that there are two bugs in the automatic conversion... That is the code for Kotlin behaves differently from standard Java.

The first problem is that Kotlin classes are final unless declared otherwise so we need to add the open keyword before the class declaration as such:

```
open class MyApplication
```

This is essential as the build server will fail with weird errors related to instanceof.



This only applies to the main class of the project, other classes in Codename One can remain **final**

The second problem is that arguments are non-null by default. The **init** method might have a null argument. So this fails with an exception. The solution is to add a question mark to the end of the call: **fun init(context: Any?)**.

So the full working sample is:

```
open class MyApplication {
    private var current: Form? = null
    private var theme: Resources? = null
    fun init(context: Any?) {
        theme = UIManager.initFirstTheme("/theme")
        Toolbar.setGlobalToolbar(true)
        Log.bindCrashProtection(true)
    }

    fun start() {
        if (current != null) {
            current!!.show()
            return
        }
        val hi = Form("Hi World", BoxLayout.y())
        hi.add(Label("Hi World"))
        hi.show()
    }

    fun stop() {
        current = getCurrentForm()
        if (current is Dialog) {
            (current as Dialog).dispose()
            current = getCurrentForm()
        }
    }

    fun destroy() {
    }
}
```

Once all of that is in place Kotlin should just work. This should be possible for additional JVM languages in the future.

2. Basics: Themes, Styles, Components and Layouts

Let's start with a brief overview of the ideas within Codename One. We'll dig deeper into these ideas as we move forward.

2.1. Components

Every button, label or element you see on the screen in a Codename One application is a **Component** [<https://www.codenameone.com/javadoc/com/codenname1/ui/Component.html>]. This is a highly simplified version of this class hierarchy:

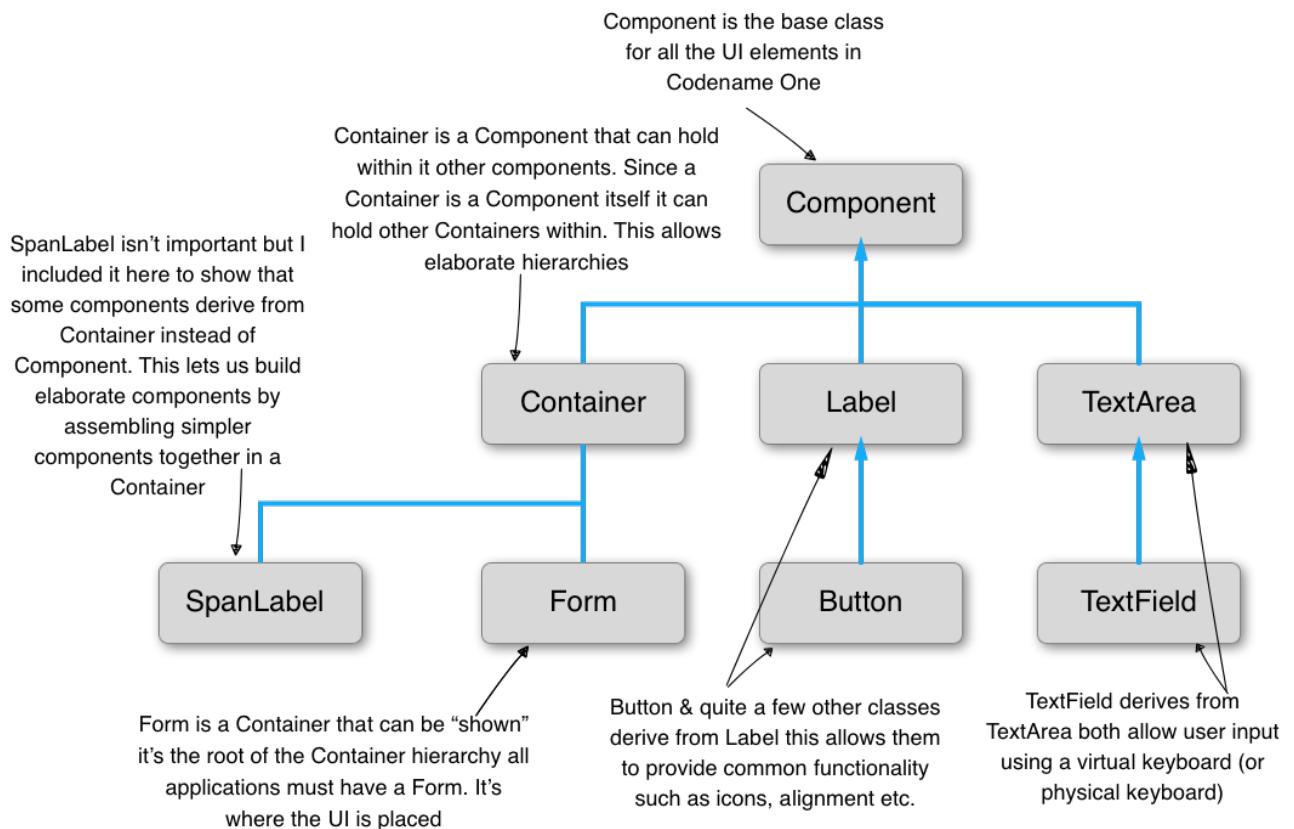


Figure 20. The Core Component Class Hierarchy

The **Form** [<https://www.codenameone.com/javadoc/com/codenname1/ui/Form.html>] is a special case **Component**. It's the root component that you can show to the user. **Container** is a **Component** type that can hold other components within it. This lets us create elaborate hierarchies by nesting **Container** instances.

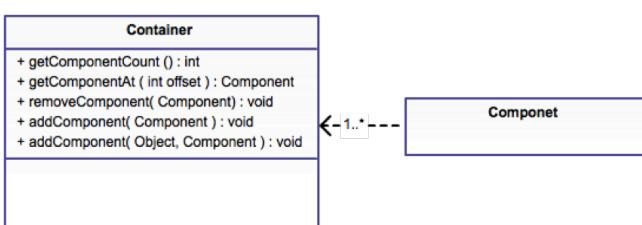


Figure 21. Component UML

A Codename One application is effectively a series of forms, only one Form can be shown at a time. The Form includes everything we see on the screen. Under the hood the **Form** is comprised of a few separate pieces:

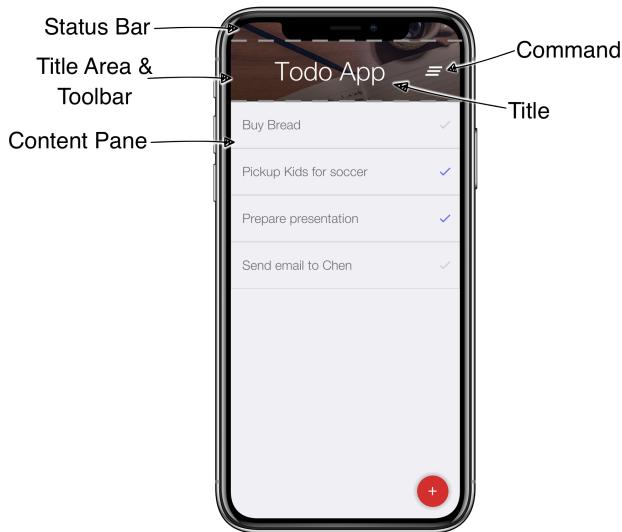


Figure 22. Structure of a Form

- Content Pane - this is literally the body of the **Form**. When we **add** a **Component** into the **Form** it goes into the content pane. Notice that Content Pane is scrollable by default on the Y axis!
- Title Area - we can't add directly into this area. The title area is managed by the **Toolbar** class. **Toolbar** is a special component that resides in the top portion of the form and abstracts the title design. The title area is broken down into two parts:
 - Title of the **Form** and its commands (the buttons on the right/left of the title)
 - Status Bar - on iOS the area on the top includes a special space so the notch, battery, clock etc. can fit. Without this the battery indicator/clock or notch would be on top of the title

Now that we understand this let's look at the new project we created and open the Java file `TodoApp.java`. In it we should see the lines that setup the UI in the `start()` method:

2.1.1. Layout Managers

A layout manager is an algorithm that decides the size and location of the components within a **Container**. Every **Container** has a layout manager associated with it. The default layout manager is **FlowLayout**.

To understand layouts we need to understand a basic concept about **Component**. Each component has a “preferred size”. This is the size in which a component “wants” to appear. E.g. for a **Label** the preferred size will be the exact size that fits the label text, icon and padding of the component.

Understanding Preferred Size

The [Component](https://www.codenameone.com/javadoc/com/codename1/ui/Component.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Component.html] class contains many useful methods. One of the most important ones is `calcPreferredSize()` which is invoked to recalculate the size a component “wants” when something changes



By default Codename One invokes the `getPreferredSize()` method and not `calcPreferredSize()` directly.
`getPreferredSize()` invokes `calcPreferredSize()` and caches the value

The preferred size is decided by the component based on internal constraints such as the font size, border sizes, padding etc.

When a layout manager positions and sizes the component, it **MIGHT** take the preferred size into account. Notice that it **MIGHT** ignore it entirely!

E.g. [FlowLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html) [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html] always gives components their exact preferred size, yet [BorderLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html) [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html] resizes the center component by default (and the other components are resized on one of their axis).

You can define a group of components to have the same preferred width or height by using the `setSameWidth` and `setSameHeight` methods e.g.:

Listing 5. setSameWidth/Height

```
Component.setSameWidth(cmp1, cmp2, cmp3, cmp4);  
Component.setSameHeight(cmp5, cmp6, cmp7);
```

Codename One has a `setPreferredSize` method that allows developers to explicitly request the size of the component. However, this caused quite a lot of problems. E.g. the preferred size should change with device orientation or similar operations. The API also triggered frequent inadvertent hardcoding of UI values such as forcing pixel sizes for components. As a result the method was deprecated.

We recommend developers use `setSameWidth`, `setSameHeight` when sizing alignment is needed. `setHidden` when hiding is needed. As a last resort we recommend overriding `calcPreferredSize`.

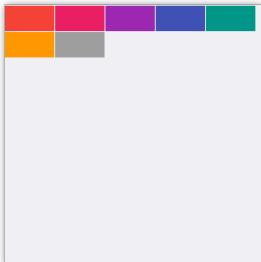
A layout manager places a component based on its own logic and the preferred size (sometimes referred to as “natural size”). A [FlowLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html) will just traverse the components based on the order they were added and size/place them one after the other. When it reaches the end of the row it will go to the new row.

Use FlowLayout Only for Simple Things

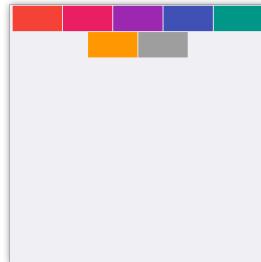


FlowLayout is great for simple things but has issues when components change their sizes dynamically (like a text field). In those cases it can make bad decisions about line breaks and take up too much space

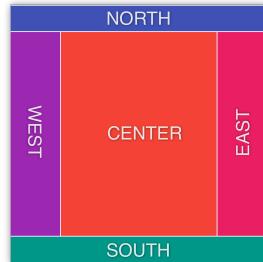
Flow Layout sizes components based on their preferred size and arranges them from left to right. When we reach the end of the line it breaks a line



Flow layout has several modes including a center mode that center aligns elements. It can align elements to the right and align vertically as well



Border Layout can position elements in the NORTH, SOUTH, EAST, WEST and CENTER. Components take their preferred size on the opposing axis. Center takes up the available space by default



Border Layout has several modes including absolute center mode where the center component takes up only its preferred size

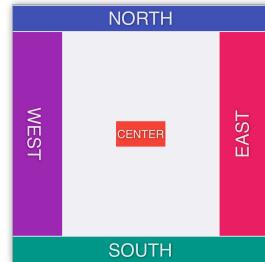
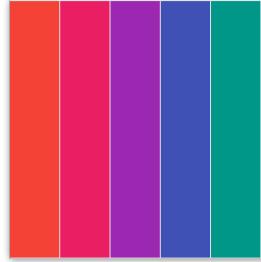


Figure 23. Layout Manager Primer Part I

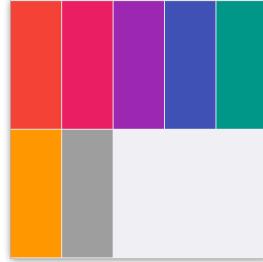
BoxLayout Y arranges components vertically giving them the available width & their preferred height



BoxLayout X arranges components horizontally giving them the available height & their preferred width



GridLayout arranges components in a grid and gives every element the same size to match the preferred size of the largest elements



LayeredLayout places components one on top of the other. They have some spacing here so you can see the layers below

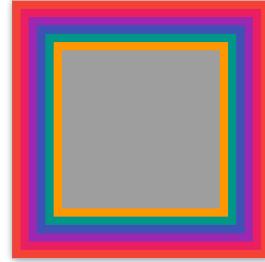


Figure 24. Layout Manager Primer Part II

Scrolling doesn't work well for all types of layouts as the positioning algorithm within the layout might break. Scrolling on the Y axis works great for **BoxLayout Y** which is why I picked it for the **TodoForm**:

Table 1. Scrolling in Layout Managers

Layout	Scrollable
FlowLayout	Possible on Y axis only
BorderLayout	Scrolling is blocked
BoxLayout Y	Scalable only on the Y axis
BoxLayout X	Scalable only on the X axis
GridLayout	Scalable
LayeredLayout	Not scrollable (usually)

Nesting Scrollable Containers

Only one element can be scrollable within the hierarchy, otherwise if you drag your finger over the `Form` Codename One won't know which element you are trying to scroll. By default form's content pane is scrollable on the Y axis unless you explicitly disable it (setting the layout to `BorderLayout` implicitly disables scrolling).

It's important to notice that it's OK to have non-scrollable layouts, e.g. `BorderLayout`, as items within a scrollable container type. E.g. in the TodoApp we added `TodoItem` which uses `BorderLayout` into a scrollable `BoxLayout Form`.

Layouts can be divided into two distinct groups:

- Constraint Based - `BorderLayout` (and a few others such as `GridBagLayout`, `MigLayout` and `TableLayout`)
- Regular - All of the other layout managers

When we add a `Component` to a `Container` with a regular layout we do so with a simple add method:

Listing 6. Adding to a Regular Container

```
Container cnt = new Container(BoxLayout.y());
cnt.add(new Label("Just Added"));
```

This works great for regular layouts but might not for constraint based layouts. A constraint based layout accepts another argument. E.g. `BorderLayout` needs a location for the `Component`:

Listing 7. Adding to a Regular Container

```
cnt.add(NORTH, new Label("Just Added"));
```

This line assumes you have an `import static com.codename1.ui.CN.*;` in the top of the file. In `BorderLayout` (which is a constraint based layout) placing an item in the `NORTH` places it in the top of the `Container`.



The `CN` class is a class that contains multiple static helper methods and functions. It's specifically designed for static import in this way to help keep our code terse

Static Global Context

The `CN` class is a thin wrapper around features in `Display`, `NetworkManager`, `Storage`, `FileSystemStorage` etc. It also adds common methods and constants from several other classes so Codename One code feels more terse e.g. we can do:

```
import static com.codename1.ui.CN.*;
```



That's optional, if you don't like static imports you can just write `CN.` for every element

From that point on you can write code that looks like this:

```
callSerially(() -> runThisOnTheEDT());
```

Instead of:

```
Display.getInstance().callSerially(() -> runThisOnTheEDT());
```

The same applies for most network manager calls e.g.:

```
addToQueue(myConnectionRequest);
```

Instead of:

```
NetworkManager.getInstance().addToQueue(myConnectionRequest);
```

Some things were changed so we won't have too many conflicts e.g. `Log.p` or `Log.e` would have been problematic so we now have:

```
log("my log message");
log(myException);
```

Instead of `Display.getInstance().getCurrent()` we now have `getCurrentForm()` since `getCurrent()` is too generic. For most methods you should just be able to remove the `NetworkManager` or `Display` access and it should "just work".

The motivation for this is three fold:

- Terse code
- Small performance gain

- Cleaner API without some of the baggage in `Display` or `NetworkManager`

Some of our samples in this guide might rely on that static import being in place. This helps us keep the code terse and readable in the code listings.

Terse Syntax

Almost every layout allows us to `add` a component using several variants of the `add` method:

Listing 8. Versions of add

```
Container cnt = new Container(BoxLayout.y());
cnt.add(new Label("Just Added")); ①
cnt.addAll(new Label("Adding Multiple"),
           new Label("Second One"));

cnt.add(new Label("Chaining")). ③
    add(new Label("Value"));
```

① Regular add

② `addAll` accepts several components and adds them in a batch

③ `add` returns the parent `Container` instance so we can chain calls like that

In the race to make code “tighter” we can make this even shorter. Almost all layout managers have their own custom terse syntax style e.g.:

Listing 9. Terse Syntax

```
Container boxY = BoxLayout.encloseY(cmp1, cmp2); ①
Container boxX = BoxLayout.encloseX(cmp3, cmp4);
Container flowCenter = FlowLayout. ②
    encloseCenter(cmp5, cmp6);
```

① Most layouts have a version of `enclose` to encapsulate components within

② `FlowLayout` has variants that support aligning the components on various axis

To sum this up, we can use layout managers and nesting to create elaborate UI’s that implicitly adapt to different screen sizes and device orientation.

Flow Layout

Flow Layout

First	Second	Third
Fourth	Fifth	

Figure 25. Flow Layout

Flow layout [https://www.codenameone.com/javadoc/com/codenname1/ui/layouts/Layout.html] lets the components “flow” horizontally and break a line when reaching the edge of the container. It’s the default layout manager for containers. Because it’s so flexible it’s also problematic as it can result in incorrect preferred size values for the parent Container [https://www.codenameone.com/javadoc/com/codename1/ui/Container.html]. This can create a reflow issue, as a result we recommend using flow layout only for trivial cases. Avoid it for things such as text input etc. As the size of the text input can vary in runtime.

```
Form hi = new Form("Flow Layout", new FlowLayout());
hi.add(new Label("First"));
    add(new Label("Second"));
    add(new Label("Third"));
    add(new Label("Fourth"));
    add(new Label("Fifth"));
hi.show();
```

Flow layout also supports terse syntax shorthand such as:

```
Container flowLayout = FlowLayout.encloseIn(
    new Label("First"),
    new Label("Second"),
    new Label("Third"),
    new Label("Fourth"),
    new Label("Fifth"));
```

Flow layout can be aligned to the left (the default), to the center, or to the right. It can also be vertically aligned to the top (the default), middle (center), or bottom.

Flow Layout

First	Second	Third
Fourth	Fifth	

Figure 26. Flow layout aligned to the center

Flow Layout

First	Second	Third
Fourth	Fifth	

Figure 27. Flow layout aligned to the right

Flow Layout

First	Second	Third
Fourth	Fifth	

Figure 28. Flow layout aligned to the center horizontally & the middle vertically

Components within the flow layout get their natural preferred size by default and are not stretched in any axis.



The natural sizing behavior is often used to prevent other layout managers from stretching components. E.g. if we have a border layout element in the south and we want it to keep its natural size instead of adding the element to the south directly we can wrap it using `parent.add(BorderLayout.SOUTH, FlowLayout.encloseCenter(dontGrowThisComponent))`.

Box Layout

BoxLayout [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BoxLayout.html] places elements in a row (`X_AXIS`) or column (`Y_AXIS`) according to box orientation. Box is a very simple and

predictable layout that serves as the "workhorse" of component lists in Codename One.

You can create a box layout Y using something like this:

```
Form hi = new Form("Box Y Layout", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(new Label("First"));
add(new Label("Second"));
add(new Label("Third"));
add(new Label("Fourth"));
add(new Label("Fifth"));
```

Which results in [this](#)

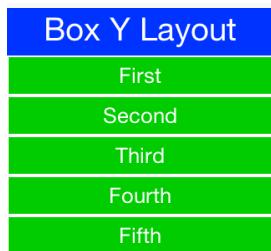


Figure 29. BoxLayout Y

Box layout also supports a shorter terse notation which we use here to [demonstrate the X axis box](#).

```
Container box = BoxLayout.encloseX(new Label("First"),
    new Label("Second"),
    new Label("Third"),
    new Label("Fourth"),
    new Label("Fifth"));
```

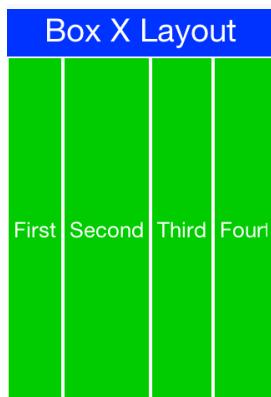


Figure 30. BoxLayout X

The box layout keeps the preferred size of its destination orientation and scales elements on the other axis. Specifically **X_AXIS** will keep the preferred width of the component while growing all the components vertically to match in size. Its **Y_AXIS** counterpart keeps the preferred height while

growing the components horizontally.

This behavior is very useful since it allows elements to align as they would all have the same size.

In some cases the growing behavior in the X axis is undesired, for these cases we can use the `X_AXIS_NO_GROW` variant.



Figure 31. `BoxLayout X_AXIS_NO_GROW`

FlowLayout vs. `BoxLayout.X_AXIS`



When applicable we recommend `BoxLayout` over `FlowLayout` as it acts more consistently in all situations. Another advantage of `BoxLayout` is the fact that it grows and thus aligns nicely

Border Layout



Figure 32. `Border Layout`

`Border layout` [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html>] is quite unique. `BorderLayout` is a constraint-based layout that can place up to five components in one of the five positions: `NORTH`, `SOUTH`, `EAST`, `WEST` or `CENTER`.

```
Form hi = new Form("Border Layout", new BorderLayout());
hi.add(BorderLayout.CENTER, new Label("Center"));
add(BorderLayout.SOUTH, new Label("South"));
add(BorderLayout.NORTH, new Label("North"));
add(BorderLayout.EAST, new Label("East"));
add(BorderLayout.WEST, new Label("West"));
hi.show();
```



The Constraints are Included in the `CN` class

You can use the static import of the `CN` class and then the syntax can be `add(SOUTH, new Label("South"))`

The layout always stretches the **NORTH/SOUTH** components on the X-axis to completely fill the container and the **EAST/WEST** components on the Y-axis. The center component is stretched to fill the remaining area by default. However, the **setCenterBehavior** allows us to manipulate the behavior of the center component so it is placed in the center without stretching.

E.g.:

```
Form hi = new Form("Border Layout", new BorderLayout());
((BorderLayout)hi.getLayout()).setCenterBehavior(BorderLayout.CENTER_BEHAVIOR_CENTER);
hi.add(BorderLayout.CENTER, new Label("Center"));
add(BorderLayout.SOUTH, new Label("South"));
add(BorderLayout.NORTH, new Label("North"));
add(BorderLayout.EAST, new Label("East"));
add(BorderLayout.WEST, new Label("West"));
hi.show();
```

Results in:



Figure 33. Border Layout with *CENTER_BEHAVIOR_CENTER*

Scrolling is Disabled in Border Layout



Because of its scaling behavior scrolling a border layout makes no sense. **Container** implicitly blocks scrolling on a border layout, but it can scroll its parents/children

In the case of RTL the EAST and WEST values are implicitly reversed as shown in this image:



Figure 34. Border Layout in RTL mode

RTL and Bidi

RTL (Right To Left) or Bidi (bi-directional) are common terms used for languages such as Hebrew, Arabic etc. These languages are written from the right to left direction hence all the UI needs to be “reversed”. Bidi denotes the fact that while the language is written from right to left, the numbers are still written in the other direction hence two directions...

Preferred Size Still Matters



The preferred size of the center component doesn't matter in border layout but the preferred size of the sides is. E.g. If you place a very large component in the **SOUTH** it will take up the entire screen and won't leave room for anything

Grid Layout

GridLayout [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridLayout.html>] accepts a predefined grid (rows/columns) and grants all components within it equal size based on the dimensions of the largest components.



The main use case for this layout is a grid of icons e.g. like one would see in the iPhone home screen

If the number of **rows * columns** is smaller than the number of components added a new row is implicitly added to the grid. However, if the number of components is smaller than available cells (won't fill the last row) blank spaces will be left in place.

In this example we can see that a 2x2 grid is used to add 5 elements, this results in an additional row that's implicitly added turning the grid to a 3x2 grid implicitly and leaving one blank cell.

```
Form hi = new Form("Grid Layout 2x2", new GridLayout(2, 2));
hi.add(new Label("First"));
    add(new Label("Second"));
    add(new Label("Third"));
    add(new Label("Fourth"));
    add(new Label("Fifth"));
```

Grid 2x2 Layout	
First	Second
Third	Fourth
Fifth	

Figure 35. Grid Layout 2x2

When we use a 2x4 size ratio we would see elements getting cropped as we do here. The grid layout uses the grid size first and doesn't pay too much attention to the preferred size of the components it holds.

Grid 2x4 Layout			
First	Second	Third	Fourth
Fifth			

Figure 36. Grid Layout 2x4

Grid also has an `autoFit` attribute that can be used to automatically calculate the column count based on available space and preferred width. This is really useful for working with UI's where the device orientation might change.

There is also a terse syntax for working with a grid that has two versions, one that uses the "auto fit" option and another that accepts the number of columns. Here's a sample of the terse syntax coupled with auto fit followed by screenshots of the same code in two orientations:

```
GridLayout.encloseIn(new Label("First"),
    new Label("Second"),
    new Label("Third"),
    new Label("Fourth"),
    new Label("Fifth"));
```

Grid Layout AutoFit			
First	Second	Third	
Fourth	Fifth		

Figure 37. Grid Layout autofit portrait

Grid Layout AutoFit			
First	Second	Third	Fourth
Fifth			

Figure 38. Grid Layout autofit landscape

2.1.2. Table Layout

The [TableLayout](https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.html) [https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.html] is a very elaborate **constraint based** layout manager that can arrange elements in rows/columns while

defining constraints to control complex behavior such as spanning, alignment/weight etc.

Note the Different Package for `TableLayout`



The `TableLayout` is in the `com.codename1.ui.table` package and not in the layouts package.

This is due to the fact that `TableLayout` was originally designed for the `Table` [<https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html>] class.

Despite being constraint based the `TableLayout` isn't strict about constraints and will implicitly add a constraint when one is missing. This is unlike the `BorderLayout` which will throw an exception in this case.



Unlike `GridLayout` `TableLayout` won't implicitly add a row if the row/column count is incorrect

```
Form hi = new Form("Table Layout 2x2", new TableLayout(2, 2));
hi.add(new Label("First"));
    add(new Label("Second"));
    add(new Label("Third"));
    add(new Label("Fourth"));
    add(new Label("Fifth"));
hi.show();
```

Table Layout 2x2

First	Second
Third	Fourth

Figure 39. 2x2 `TableLayout` with 5 elements, notice that the last element is missing

`TableLayout` supports the ability to grow the last column which can be enabled using the `setGrowHorizontally` method. You can also use a shortened terse syntax to construct a `TableLayout` however since the `TableLayout` is a constraint based layout you won't be able to utilize its full power with this syntax.

The default usage of the `encloseIn` method below uses the `setGrowHorizontally` flag.

```
Container tl = TableLayout.encloseIn(2, new Label("First"),
    new Label("Second"),
    new Label("Third"),
    new Label("Fourth"),
    new Label("Fifth"));
```

TableLayout Enclose 2	
First	Second
Third	Fourth
Fifth	

Figure 40. `TableLayout.encloseIn()` with default behavior of growing the last column

The Full Potential

`TableLayout` is a beast, to truly appreciate it we need to use the constraint syntax which allows us to span, align and set width/height for the rows and columns.

`TableLayout` works with a `Constraint` [<https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.Constraint.html>] instance that can communicate our intentions into the layout manager. Such constraints can include more than one attribute e.g. span and height.



`TableLayout` constraints can't be reused for more than one component

The constraint class supports the following attributes

Table 2. Constraint Properties

<code>column</code>	The column for the table cell. This defaults to -1 which will just place the component in the next available cell
<code>row</code>	Similar to column, defaults to -1 as well
<code>width</code>	The column width in percentages, -1 will use the preferred size. -2 for width will take up the rest of the available space
<code>height</code>	Similar to width but doesn't support the -2 value
<code>spanHorizontal</code>	The cells that should be occupied horizontally defaults to 1 and can't exceed the column count - current offset.
<code>spanVertical</code>	Similar to spanHorizontal with the same limitations
<code>horizontalAlign</code>	The horizontal alignment of the content within the cell, defaults to the special case -1 value to take up all the cell space can be either <code>-1</code> , <code>Component.LEFT</code> , <code>Component.RIGHT</code> or <code>Component.CENTER</code>
<code>verticalAlign</code>	Similar to horizontalAlign can be one of <code>-1</code> , <code>Component.TOP</code> , <code>Component.BOTTOM</code> or <code>Component.CENTER</code>



You only need to set `width/height` to one cell in a column/row

The [table layout constraint sample](#) tries to demonstrate some of the unique things you can do with

constraints.

```
TableLayout tl = new TableLayout(2, 3); ①
Form hi = new Form("Table Layout Cons", tl);
hi.setScrollable(false); ②
hi.add(tl.createConstraint(). ③
        widthPercentage(20),
        new Label("AAA")).  
  
add(tl.createConstraint(). ④
        horizontalSpan(2).
        heightPercentage(80).
        verticalAlign(Component.CENTER).
        horizontalAlign(Component.CENTER),
        new Label("Span H")).  
  
add(new Label("BBB")).  
  
add(tl.createConstraint().
        widthPercentage(60).
        heightPercentage(20),
        new Label("CCC")).  
  
add(tl.createConstraint().
        widthPercentage(20),
        new Label("DDD"));
```

- ① We need the `TableLayout` instance to create constraints. A constraint must be created for every component and must be used with the same layout as the parent container
- ② To get the look in the [screenshot](#) we need to turn scrolling off so the height constraint doesn't take up available height. Otherwise it will miscalculate available height due to scrolling. You can scroll a `TableLayout` but sizing will be different
- ③ We create the constraint and instantly apply width to it. This is a shorthand syntax for the [code block below](#)
- ④ We can chain constraint creation using a call like this so multiple constraints apply to a single cell. Notice that we don't span and set width on the same axis (horizontal span + width), doing something like that would create confusing behavior

Here is the full code mentioned in item 3:

```
TableLayout.Constraint cn = tl.createConstraint();
cn.setWidthPercentage(20);
hi.add(cn, new Label("AAA")).
```

Table Layout Cons		
AAA	Span H	
BBB	CCC	DDD

Figure 41. TableLayout constraints can be used to create very elaborate UI's

TextMode Layout

`TextModeLayout` is a unique layout manager. It acts like `TableLayout` on Android and like `BoxLayout.Y_AXIS` in other platforms. Internally it delegates to one of these two layout managers so in a sense it doesn't have as much functionality of its own.

E.g. this is a sample usage of `TextModeLayout`:

```
TextModeLayout tl = new TextModeLayout(3, 2);
Form f = new Form("Pixel Perfect", tl);
TextComponent title = new TextComponent().label("Title");
TextComponent price = new TextComponent().label("Price");
TextComponent location = new TextComponent().label("Location");
TextComponent description = new TextComponent().label("Description").multiline(true);

f.add(tl.createConstraint().horizontalSpan(2), title);
f.add(tl.createConstraint().widthPercentage(30), price);
f.add(tl.createConstraint().widthPercentage(70), location);
f.add(tl.createConstraint().horizontalSpan(2), description);
f.setEditableOnShow(title.getField());
f.show();
```

Title	Vintage Dress
Price	5
Location	Somewhere
Description	Long text that can span multiple lines

Figure 42. TextModeLayout on iOS

Title	Vintage dress
Price	Location
5	Somewhere
Description	
Long text	

Figure 43. TextModeLayout on Android with the same code

As you can see from the code and samples above there is a lot going on under the hood. On Android we want a layout that's similar to `TableLayout` so we can "pack" the entries. On iOS we want a box

layout Y type of layout but we also want the labels/text to align properly...

The `TextModeLayout` isn't really a layout as much as it is a delegate. When running in the Android mode (which we refer to as the “on top” mode) the layout is almost an exact synonym of `TableLayout` and in fact delegates to an underlying `TableLayout`. In fact there is a `public final` table instance within the layout that you can refer to directly...

There is one small difference between the `TextModeLayout` and the underlying `TableLayout` and that's our choice to default to align entries to `TOP` with this mode.



Aligning to `TOP` is important for error handling for `TextComponent` in Android otherwise the entries “jump”

When working in the non-android environment we use a `BoxLayout` on the Y axis as the delegate. There's one thing we do here that's different from a default box layout: grouping. Grouping allows the labels to align by setting them to the same width, internally it invokes `Component.setSameWidth()`. Since text components hide the labels there is a special `group` method there that can be used. However, this is implicit with the `TextModeLayout` which is pretty cool.

`TextModeLayout` was created specifically for the `TextComponent` and `InputComponent` so check out the section about them in the components chapter.

2.1.3. Layered Layout

When used without constraints, the `LayeredLayout` [<https://www.codenameone.com/javadoc/com/codenname1/ui/layouts/LayeredLayout.html>] places the components in order one on top of the other and sizes them all to the size of the largest component. This is useful when trying to create an overlay on top of an existing component. E.g. an “x” button to allow removing the component.



Figure 44. The X on this button was placed there using the layered layout code below

The code to generate this UI is slightly complex and contains very few relevant pieces. The only truly relevant piece is this block:

```
hi.add(LayeredLayout.encloseIn(settingsLabel,  
    FlowLayout.encloseRight(close)));
```

We are doing three distinct things here:

1. We are adding a layered layout to the form
2. We are creating a layered layout and placing two components within. This would be the equivalent of just creating a [LayeredLayout Container](#) and invoking `add` twice
3. We use [FlowLayout](#) to position the X close button in the right position



When used without constraints, the layered layout sizes all components to the exact same size one on top of the other. It usually requires that we use another container within; in order to position the components correctly

This is the full source of the example for completeness:

```
Form hi = new Form("Layered Layout");
int w = Math.min(Display.getInstance().getDisplayWidth(), Display.getInstance().getDisplayHeight());
Button settingsLabel = new Button("");
Style settingsStyle = settingsLabel.getAllStyles();
settingsStyle.setFgColor(0xff);
settingsStyle.setBorder(null);
settingsStyle.setBgColor(0xff00);
settingsStyle.setBgTransparency(255);
settingsStyle.setFont(settingsLabel.getUnselectedStyle().getFont().derive(w / 3, Font.STYLE_PLAIN));
FontImage.setMaterialIcon(settingsLabel, FontImage.MATERIAL_SETTINGS);
Button close = new Button("");
close.setUUID("Container");
close.getAllStyles().setFgColor(0xff0000);
FontImage.setMaterialIcon(close, FontImage.MATERIAL_CLOSE);
hi.add(LayeredLayout.encloseIn(settingsLabel,
    FlowLayout.encloseRight(close)));
```

Forms have a built in layered layout that you can access via [getLayeredPane\(\)](#), this allows you to overlay elements on top of the content pane.

The layered pane is used internally by components such as [InteractionDialog](#) [<https://www.codenameone.com/javadoc/com/codename1/components/InteractionDialog.html>], [AutoComplete](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html>] etc.



Codename One also includes a GlassPane that resides on top of the layered pane. Its useful if you just want to "draw" on top of elements but is harder to use than layered pane

Insets and Reference Components

As of Codename One 3.7, [LayeredLayout](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.html>] supports [insets](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.LayeredLayoutConstraint.Inset.html>] for its children. This effectively allows you to position child components precisely where you want them, relative to their container or siblings. This functionality forms the under-pinnings of the GUI Builder's [Auto-Layout mode](#).

As an example, suppose you wanted to position a button in the lower right corner of its container. This can be achieved with [LayeredLayout](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/>]

layouts/LayeredLayout.html] as follows:

```
Container cnt = new Container(new LayeredLayout());
Button btn = new Button("Submit");
LayeredLayout ll = (LayeredLayout)cnt.getLayout();
cnt.add(btn);
ll.setInsets(btn, "auto 0 0 auto");
```

The result is:



The only thing new here is this line:

```
ll.setInsets(btn, "auto 0 0 auto");
```

This is called after `btn` has already been added to the container. It says that we want its insets to be "auto" on the top and left, and 0 on the right and bottom. This `insets` string follows the CSS notation of `top right bottom left` (i.e. start on top and go clockwise), and the values of each inset may be provided in pixels (px), millimetres (mm), percent (%), or the special "auto" value. Like CSS, you can also specify the insets using a 1, 2, or 3 values. E.g.

1. "`1mm`" - Sets 1mm insets on all sides.
2. "`1mm 2mm`" - Sets 1mm insets on top and bottom; 2mm on left and right.
3. "`1mm 10% 2mm`" - Sets 1mm on top, 10% on left and right, and 2mm on bottom.
4. "`1mm 2mm 1px 50%`" - Sets 1mm on top, 2mm on right, 1px on bottom, and 50% on left.

auto Insets

The special "auto" inset indicates that it is a flexible inset. If all insets are set to "auto", then the component will be centered both horizontally and vertically inside its "bounding box".



The "inset bounding box" is the containing box from which a component's insets are measured. If the component's insets are not linked to any other components, then its inset bounding box will be the inner bounds (i.e. taking padding into account) of the component's parent container.

If one inset is fixed (i.e. defined in px, mm, or %), and the opposite inset is "auto", then the "auto"

inset will simply allow the component to be its preferred size. So if you want to position a component to be centered vertically, and 5mm from the left edge, you could do:

```
ll.setInsets(btn, "auto auto auto 5mm");
```

Resulting in:



Figure 45. Button vertically centered 5mm from left edge

Move it to the right edge with:

```
ll.setInsets(btn, "auto 5mm auto auto");
```

% Insets

Percent (%) insets are calculated with respect to the inset bounding box. A 50% inset is measured as 50% of the length of the bounding box on the inset's axis. E.g. A 50% inset on top would be 50% of the height of the inset bounding box. A 50% inset on the right would be 50% of the width of the inset bounding box.

Insets, Margin, and Padding

A component's position in a layered layout is determined as follows: (Assume that `cmp` is the component that we are positioning, and `cnt` is the container (In pseudo-code):

```
x = cnt.paddingLeft + cmp.calculatedInsetLeft + cmp.marginLeft  
y = cnt.paddingTop + cmp.calculatedInsetTop + cmp.marginTop  
w = cnt.width - cnt.verticalScroll.width - cnt.paddingRight - cmp.calculatedInsetRight - cmp.marginRight - x  
h = cnt.height - cnt.horizontalScroll.height - cnt.paddingBottom - cmp.calculatedInsetBottom - cmp.marginBottom - y
```

The `calculatedInsetXXX` values here will be the same as the corresponding provided inset if the inset has no reference component. If it does have a reference component, then the calculated inset will depend on the position of the reference component.



If no inset is specified, then it is assumed to be 0. This ensures compatibility with designs that were created before layered layout supported insets.

Component References: Linking Components together

If all you need to do is position a component relative to its parent container's bounds, then mere insets provide you with sufficient vocabulary to achieve this. But most UIs are more complex than this and require another concept: reference components. In many cases you will want to position a component relative to another child of the same container. This is also supported.

For example, suppose I want to place a text field in the center of the form (both horizontally and vertically), and have a button placed beside it to the right. Positioning the text field is trivial (`setInset(textField, "auto")`), but there is no inset that we can provide that would position the button to the right of the text field. To accomplish our goal, we need to set the text field as a reference component of the button's left inset - so that the button's left inset is "linked" to the text field. Here is the syntax:

```
Container cnt = new Container(new LayeredLayout());
LayeredLayout ll = (LayeredLayout)cnt.getLayout();
Button btn = new Button("Submit");
TextField tf = new TextField();
cnt.add(tf).add(btn);
ll.setInsets(tf, "auto");
    .setInsets(btn, "auto auto auto 0")
    .setReferenceComponentLeft(btn, tf, 1f);
```

This would result in:

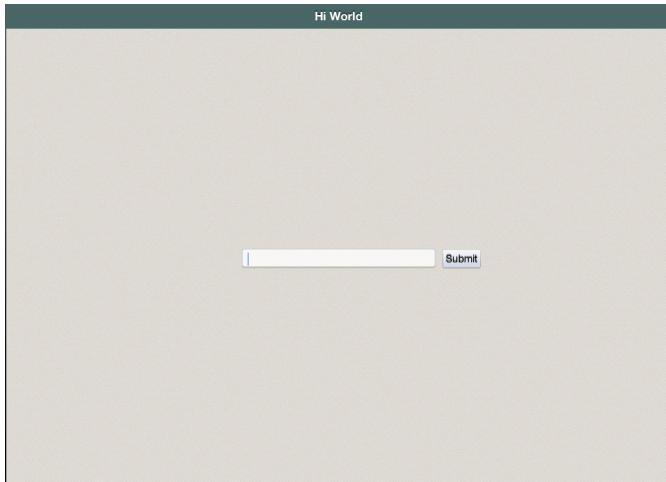


Figure 46. Button's left inset linked to text field

The two active lines here are the last two:

```
.setInsets(btn, "auto auto auto 0") ①
.setReferenceComponentLeft(btn, tf, 1f); ②
```

① Sets the left inset on `btn` to 0.

② Links `btn's left inset to 'tf` so that it is measured from the text field. The third parameter (1.0) is the reference position. This will generally either be 0 (meaning the reference point is the left edge of the text field), or 1 (meaning the reference point is the right edge of the text field). In

this case we set a reference position of **1.0** because we want the button to be aligned to the text field's right edge.

The reference position is defined as the distance, expressed as a fraction of the reference component's length on the inset's axis, between the reference component's leading (outer) edge and the point from which the inset is measured. A reference position of 0 means that the inset is measured from the leading edge of the reference component. A value of 1.0 means that the inset is measured from the trailing edge of the reference component. A value of 0.5 means that the inset is measured from the center of the reference component. Etc... Any floating point value can be used. The designer currently only makes use of 0 and 1.



The definition above may make reference components and reference position seem more complex than it is. Some examples:

1. For a top inset:

- a. `referencePosition == 0` ⇒ the inset is measured from the **top** edge of the reference component.
- b. `referencePosition == 1` ⇒ the inset is measured from the **bottom** edge of the reference component.

2. For a bottom inset:

- a. `referencePosition == 0` ⇒ the inset is measured from the **bottom** edge of the reference component.
- b. `referencePosition == 1` ⇒ the inset is measured from the **top** edge of the reference component.

3. For a left inset:

- a. `referencePosition == 0` ⇒ the inset is measured from the **left** edge of the reference component.
- b. `referencePosition == 1` ⇒ the inset is measured from the **right** edge of the reference component.

4. For a right inset:

- a. `referencePosition == 0` ⇒ the inset is measured from the **right** edge of the reference component.
- b. `referencePosition == 1` ⇒ the inset is measured from the **left** edge of the reference component.

Layers In Codename One

Codename One allows placing components one on top of the other and we commonly use layered layout to do that. The form class has a builtin **Container** that resides in a layer on top of the content pane of the form.

When you add an element to a form it implicitly goes into the content pane. However, you can use `getLayeredPane()` and add any **Component** there. Such a **Component** will appear above the content pane. Notice that this layer resides below the title area (on the Y axis) and won't draw on top of that.

When Codename One introduced the layered pane it was instantly useful. However, its popularity caused conflicts. Two separate pieces of code using the layered pane could easily collide with one another. Codename One solved it with `getLayeredPane(Class c, boolean top)`. This method allocates a layer for a specific class within the layered pane. This way if two different classes use this method instead of the `getLayeredPane()` method they won't collide. Each will get its own container in a layered layout within the layered pane seamlessly. The **top** flag indicates whether we want the layer to be the top most or bottom most layer within the layered pane (assuming it wasn't created already). This allows you to place a layer that can appear above or below the already installed layers.

We only make use of the layered pane in this book but there are two additional layers on top of it. The form layered pane is identical to the layered pane but spans the entire height of the **Form** (including the title area). As a result the form layered pane is slower as it needs to handle some special cases to support this functionality.

The glass pane is the top most layer, unlike the layered pane it's purely a graphical layer. You can only draw on the glass pane with a **Painter** instance and a **Graphics** object. You can't add components into that layer.

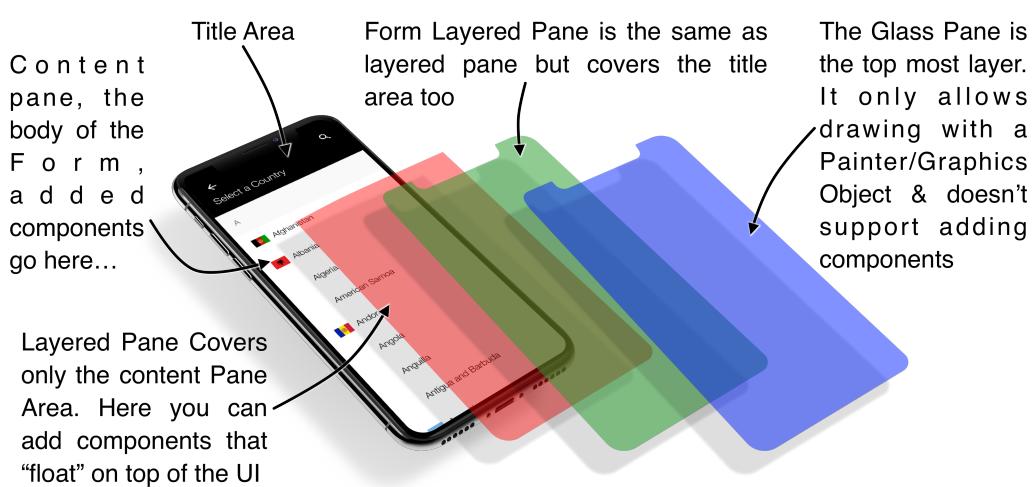


Figure 47. The Layered Pane

2.1.4. GridBag Layout

[GridBagLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html>] was introduced to simplify the process of porting existing Swing/AWT code with a more familiar API. The API for this layout is problematic as it was designed for AWT/Swing where styles were unavailable. As a result it has its own insets API instead of using elements such as padding/margin.

Our recommendation is to [use Table](#) which is just as powerful but has better Codename One integration.

To demonstrate [GridBagLayout](#) we ported [the sample from the Java tutorial](#) [<http://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html>] to Codename One.

```

Button button;
hi.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
//natural height, maximum width
c.fill = GridBagConstraints.HORIZONTAL;
button = new Button("Button 1");
c.weightx = 0.5;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
hi.addComponent(c, button);

button = new Button("Button 2");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 1;
c.gridy = 0;
hi.addComponent(c, button);

button = new Button("Button 3");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 2;
c.gridy = 0;
hi.addComponent(c, button);

button = new Button("Long-Named Button 4");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 40;      //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
hi.addComponent(c, button);

button = new Button("5");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 0;       //reset to default
c.weighty = 1.0;   //request any extra vertical space
c.anchor = GridBagConstraints.PAGE_END; //bottom of space
c.insets = new Insets(10,0,0,0); //top padding
c.gridx = 1;        //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = 2;        //third row
hi.addComponent(c, button);

```

Notice that because of the way gridbag works we didn't provide any terse syntax API for it although it should be possible.



Figure 48. GridbagLayout sample from the Java tutorial running on Codename One

2.1.5. Group Layout

[GroupLayout](https://www.codenameone.com/javadoc/com/codenname1/ui/layouts/Layout.html) [https://www.codenameone.com/javadoc/com/codenname1/ui/layouts/Layout.html] is a layout that would be familiar to the users of [the NetBeans GUI builder \(Matisse\)](https://netbeans.org/features/java/swing.html) [https://netbeans.org/features/java/swing.html]. Its a layout manager that's really hard to use for manual coding but is powerful for some elaborate use cases. Although [MiGLayout](#) and [LayeredLayout](#) might be superior options.

It was originally added during the LWUIT days as part of an internal attempt to port Matisse to LWUIT. It's still useful to this day as developers copy and paste Matisse code into Codename One and produce very elaborate layouts with drag and drop.

Since the layout is based on an older version of [GroupLayout](#) some things need to be adapted in the code or you should use the special "compatibility" library for Matisse to get better interaction. We also recommend tweaking Matisse to use import statements instead of full package names, that way if you use [Label](#) just changing the awt import to a Codename One import will make it work for Codenmae One's [Label](#).

Unlike any other layout manager [GroupLayout](#) adds the components into the container instead of the standard API. This works nicely for GUI builder code but as you can see from this sample it doesn't make the code very readable:

```
Form hi = new Form("GroupLayout");

Label label1 = new Label();
Label label2 = new Label();
Label label3 = new Label();
Label label4 = new Label();
Label label5 = new Label();
Label label6 = new Label();
Label label7 = new Label();

label1.setText("label1");
label2.setText("label2");
label3.setText("label3");
label4.setText("label4");
label5.setText("label5");
label6.setText("label6");
label7.setText("label7");

GroupLayout layout = new GroupLayout(hi.getContentPane());
hi.setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(GroupLayout.LEADING)
```

```

.layout.createSequentialGroup()
    .addContainerGap()
    .add(layout.createParallelGroup(GridLayout.LEADING)
        .add(layout.createSequentialGroup()
            .add(label1, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
            .addPreferredGap(LayoutConstraint.RELATED)
            .add(layout.createParallelGroup(GridLayout.LEADING)
                .add(label4, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
                .add(label3, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
                .add(label2, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)))
            .add(label5, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
        .add(layout.createSequentialGroup()
            .add(label6, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
            .addPreferredGap(LayoutConstraint.RELATED)
            .add(label7, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)))
        .addContainerGap(296, Short.MAX_VALUE))
    );
layout.setVerticalGroup(
    layout.createParallelGroup(GridLayout.LEADING)
    .add(layout.createSequentialGroup()
        .addContainerGap()
        .add(layout.createParallelGroup(GridLayout.TRAILING)
            .add(label2, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
            .add(label1, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE))
        .addPreferredGap(LayoutConstraint.RELATED)
        .add(label3, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
        .addPreferredGap(LayoutConstraint.RELATED)
        .add(label4, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
        .addPreferredGap(LayoutConstraint.RELATED)
        .add(label5, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
        .addPreferredGap(LayoutConstraint.RELATED)
        .add(layout.createParallelGroup(GridLayout.LEADING)
            .add(label6, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)
            .add(label7, GridLayout.PREFERRED_SIZE, GridLayout.DEFAULT_SIZE, GridLayout.PREFERRED_SIZE)))
        .addContainerGap(150, Short.MAX_VALUE))
    );

```

GroupLayout



Figure 49. GroupLayout Matisse generated UI running in Codename One

If you are porting newer Matisse code there are simple changes you can do:

- Change `addComponent` to `add`
- Change `addGroup` to `add`
- Remove references to `ComponentPlacement` and reference `LayoutConstraint` directly

2.1.6. Mig Layout

[MigLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/mig/MigLayout.html) [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/mig/MigLayout.html] is a popular cross platform layout manager that was ported to Codename One from Swing.



MiG is still considered experimental so proceed with caution!

The API was deprecated to serve as a warning of its experimental status.

The best reference for MiG would probably be its [quick start guide \(PDF link\)](#) [http://www.miglayout.com/QuickStart.pdf]. As a reference we ported one of the samples from that PDF to Codename One:

```
Form hi = new Form("MigLayout", new MigLayout("fillx,insets 0"));

hi.add(new Label("First"));
    add("span 2 2", new Label("Second")); // The component will span 2x2 cells.
    add("wrap", new Label("Third")); // Wrap to next row
    add(new Label("Forth"));
    add("wrap", new Label("Fifth")); // Note that it "jumps over" the occupied cells.
    add(new Label("Sixth"));
    add(new Label("Seventh"));
hi.show();
```



Figure 50. MiG layout sample ported to Codename One

It should be reasonably easy to port MiG code but you should notice the following:

- MiG handles a lot of the spacing/padding/margin issues that are missing in Swing/AWT. With Codename One styles we have the padding and margin which are probably a better way to do a lot of the things that MiG does
- The `add` method in Codename One can be changed as shown in the sample above.
- The constraint argument for Codename One `add` calls appears before the `Component` instance.

2.2. Themes and Styles

Next we need to introduce you to 3 important terms in Codename One: Theme, Style and UIID.

Themes are very similar conceptually to CSS, in fact they can be created with CSS syntax as we'll

discuss soon. The various Codename One ports ship with a native theme representing the appearance of the native OS UI elements. Every Codename One application has its own theme that derives the native theme and overrides behavior within it.

If the native theme has a button defined, we can override properties of that button in our theme. This allows us to customize the look while retaining some native appearances. This works by merging the themes to one big theme where our application theme overrides the definitions of the native theme. This is pretty similar to the cascading aspect of CSS if you are familiar with that.

Themes consist of a set of UIID definitions. Every component in Codename One has a UIID associated with it. UIID stands for User Interface Identifier. This UIID connects the theme to a specific component. A UIID maps to CSS classes if you are familiar with that concept. However, Codename One doesn't support the complex CSS selector syntax options as those can impact runtime performance.

E.g. see this code where:

Listing 10. setUIID on TextField

```
nameText.setUIID("Label");
```

This is a text field component (user input field) but it will look like a **Label**.

Effectively we told the text field that it should use the UIID of **Label** when it's drawing itself. It's very common to do tricks like that in Codename One. E.g. `button.setUIID("Label")` would make a button appear like a label and allow us to track clicks on a "Label".

The UIID's translate the theme elements into a set of **Style** objects. These **Style** objects get their initial values from the theme but can be further manipulated after the fact. So if I want to make the text field's foreground color red I could use this code:

Listing 11. setUIID on TextField

```
nameText.getAllStyles().setFgColor(0xff0000);
```

The color is in hexadecimal **RRGGBB** format so **0xff00** would be green and **0xff0000** would be red.

`getAllStyles()` returns a **Style** object but why do we need "all" styles?

Each component can have one of 4 states and each state has a **Style** object. This means we can have 4 style objects per Component:

- **Unselected**—used when a component isn't touched and doesn't have focus. You can get that object with `getUnselectedStyle()`.
- **Selected**—used when a component is touched or if focus is drawn for non-touch devices. You can get that object with `getSelectedStyle()`.
- **Pressed**—used when a component is pressed. Notice it's only applicable to buttons and button subclasses usually. You can get that object with `getPressedStyle()`.

- **Disabled**—used when a component is disabled. You can get that object with `getDisabledStyle()`.

The `getAllStyles()` method returns a special case `Style` object that lets you set the values of all 4 styles from one class so the code before would be equivalent to invoking all 4 `setFgColor` methods. However, `getAllStyles()` only works for setting properties not for getting them!

Don't use `getStyle()` for manipulation



`getStyle()` returns the current `Style` object which means it will behave inconsistently. The paint method uses `getStyle()` as it draws the current state of the `Component` but other code should avoid that method. Use the specific methods instead: `getUnselectedStyle()`, `getSelectedStyle()`, `getPressedStyle()`, `getDisabledStyle()` and `getAllStyles()`

As you can see, it's a bit of a hassle to change styles from code which is why the theme is so appealing.

2.2.1. Theme

A theme allows the designer to define the styles externally via a set of UIID's (User Interface ID's), the themes are created via the Codename One Designer tool and allow developers to separate the look of the component from the application logic.



You can customize the theme using CSS which we'll discuss a bit later

The theme is stored in the `theme.res` file in the `src` root of the project. We load the theme file using this line of code in the `init(Object)` method in the main class of the application:

Listing 12. Theme Loading Code

```
theme = UIManager.initFirstTheme("/theme");
```

This code is shorthand for resource file loading and for the installation of theme. You could technically have more than one theme in a resource file at which point you could use `initNamedTheme()` instead. The resource file is a special file format that includes inside it several features:

- Themes
- Images
- Localization Bundles
- Data files

It also includes some legacy features such as the old GUI builder.



We're mentioning the legacy GUI builder for reference only we won't discuss the old GUI builders in this guide

We can open the designer tool by double clicking the res file. The UI can be a bit overwhelming at first so I'll try to walk slowly through the steps.

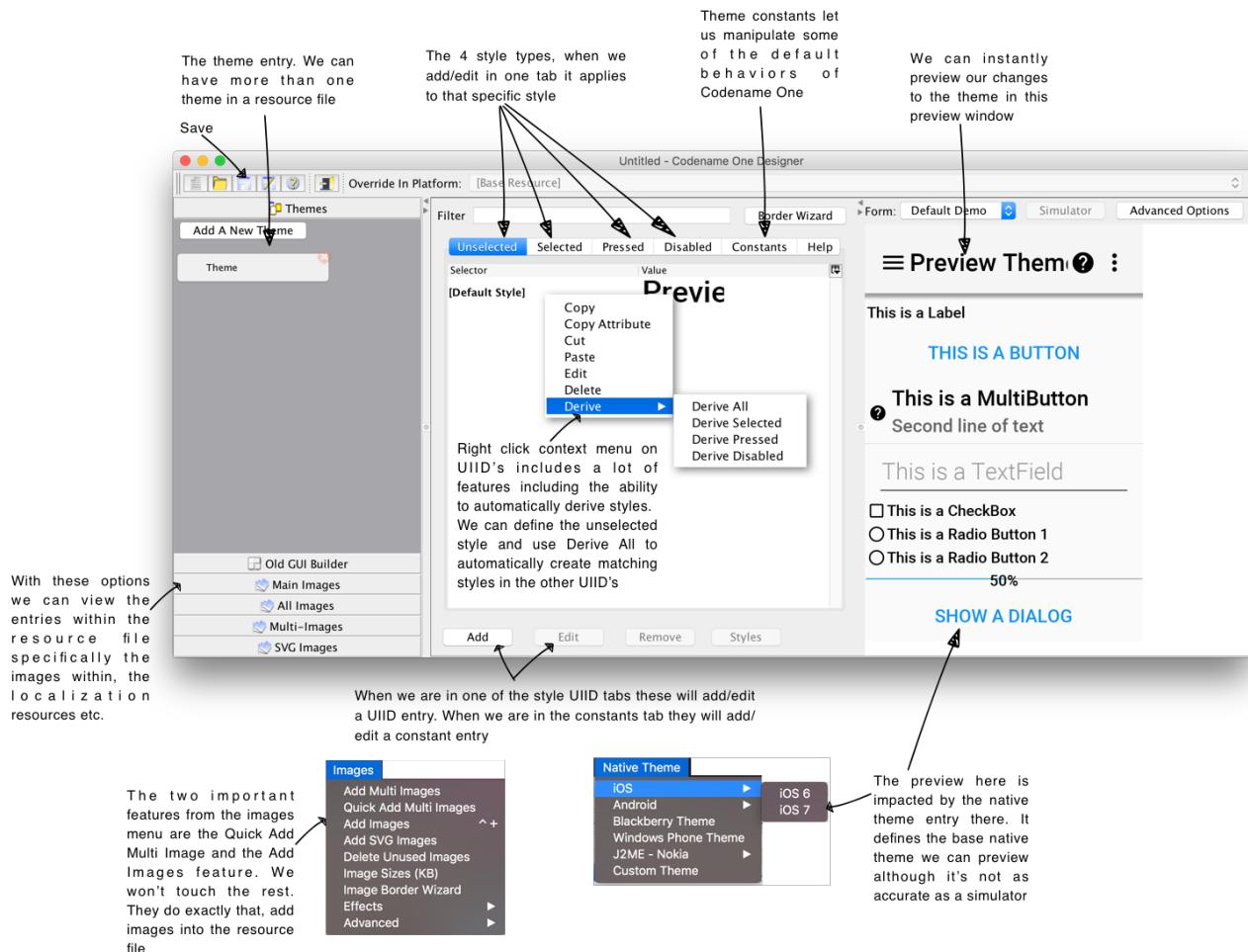


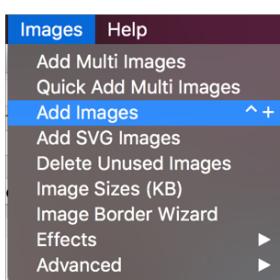
Figure 51. Codename One Designer Feature Map



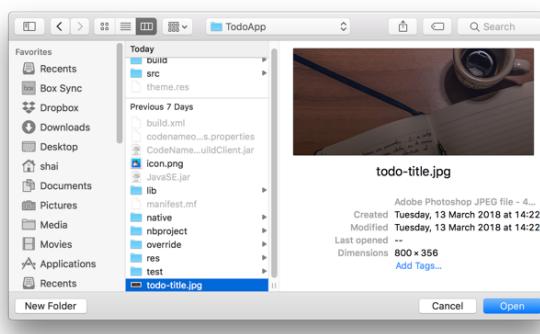
If you end up using CSS this section isn't crucial but it's worth browsing through as using the designer tool helps in tracking CSS issues

There are two crucial features in this tool: theming and images.

- 1 Select the Images menu and click Add Images



- 2 Pick the image in the file picker dialog box



- 3 You should be able to see the image in the Main Images section in the designer tool

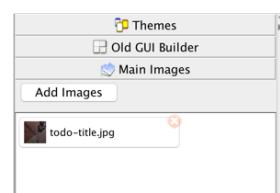


Figure 52. Add a New Image

You will notice there is more than one type of image. We'll discuss multi-images later.

Now we can go back to the theme view in the designer tool and press the Add button in the **Unselected** tab. Notice we have tabs for every state of **Style** as well as a special tab for theme constants that we will discuss later.

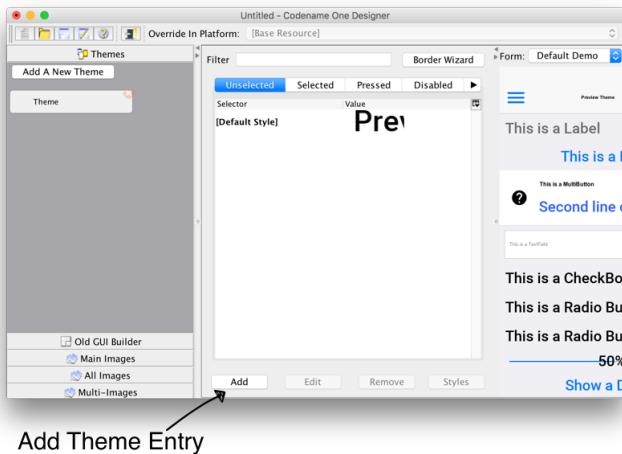


Figure 53. The Add Button

After pressing that button we should see something that looks like this:

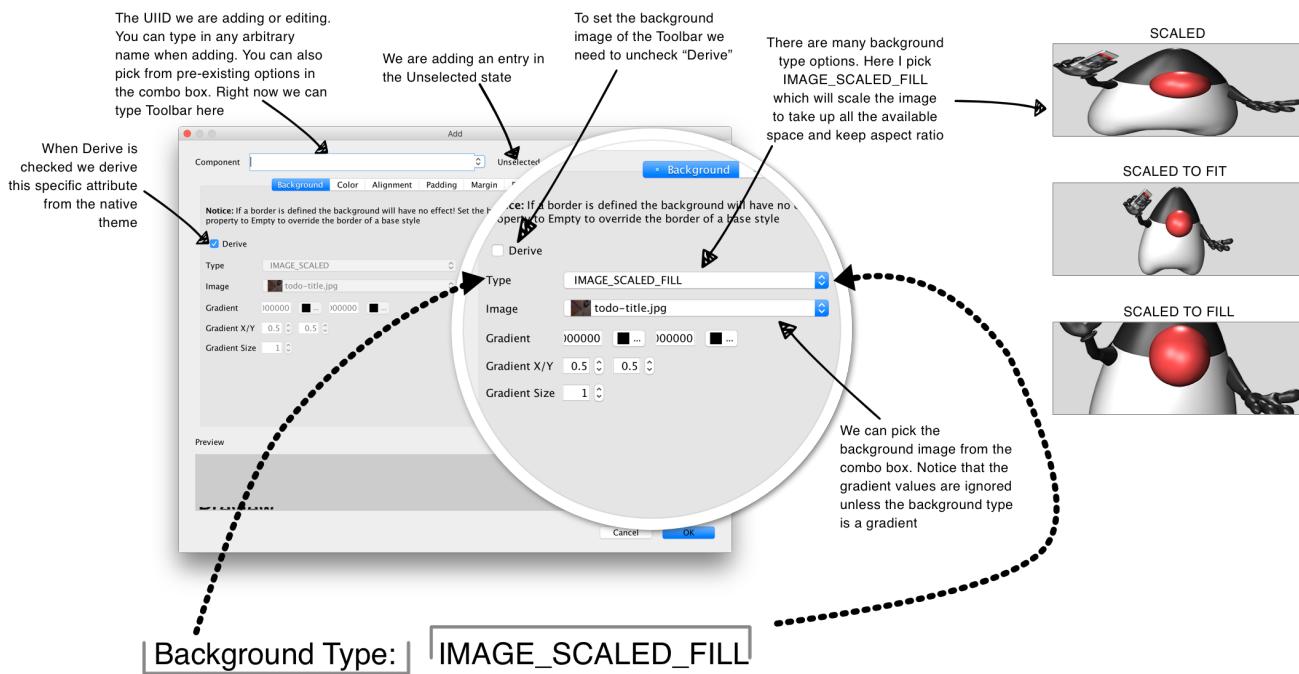


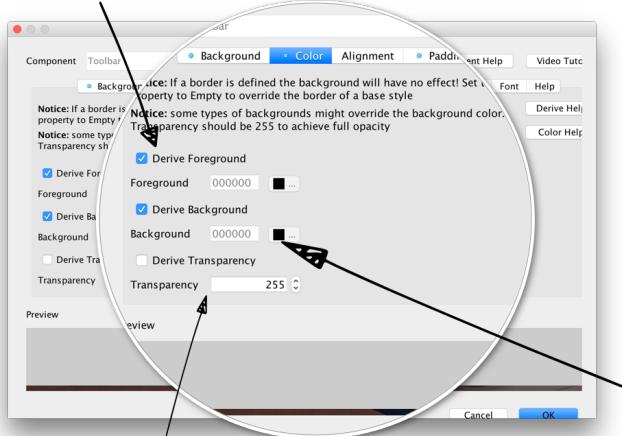
Figure 54. Add the Theme Entry for the Toolbar



Don't forget to press the save button in the designer after making changes

There are several other options in the add theme entry dialog. Lets go over them and review what we should do for each tab in this UI:

The foreground color of the component e.g. the text color of a label in this case we don't need the foreground as we'll style it in the "Title" UIID



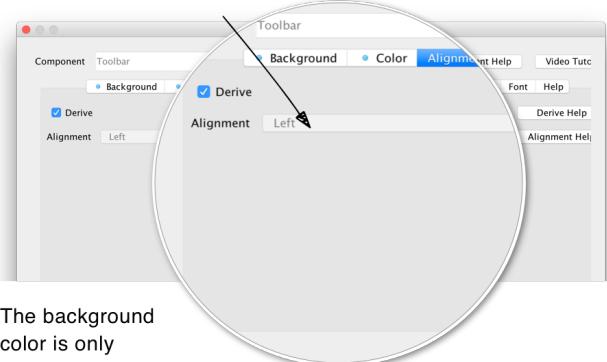
Transparency of 255 indicates completely opaque & 0 indicates complete transparency. It's best to define it to 255 as the image we show is opaque

Padding is the extra space the component takes beyond its "natural size". It can be expressed in millimeters, pixels or percentage of the screen size.

We almost always use millimeters for padding. Notice that in the screenshot below I ignore the right margin as the title on Android is aligned to the left



Alignment can be left/right or center. This isn't applicable to all components and will only work for components deriving from Label or TextArea.



The background color is only applicable if the component doesn't have a border & doesn't have a background type

Margin is the space between this component & the other components next to it. We often set it to 0 when we want to take up available space

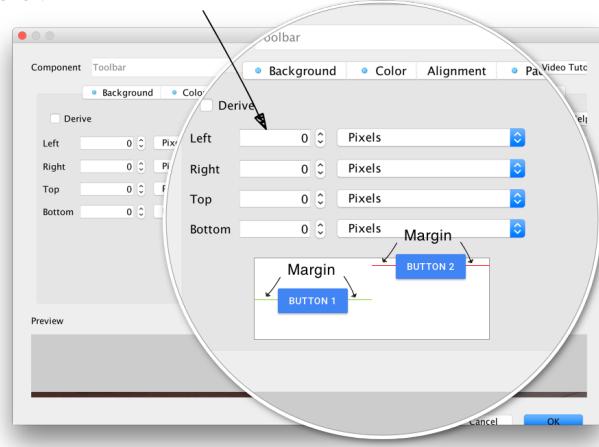
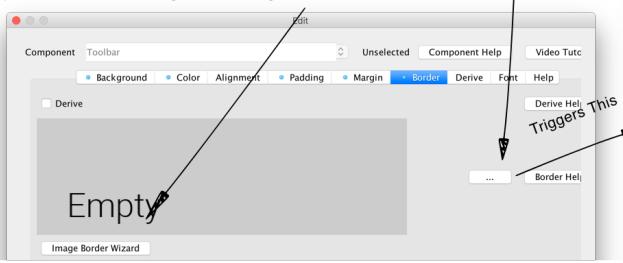
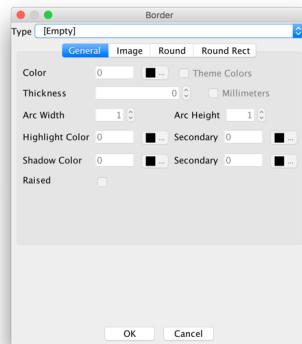


Figure 55. The Rest of the Add Theme Entry Dialog - Part I

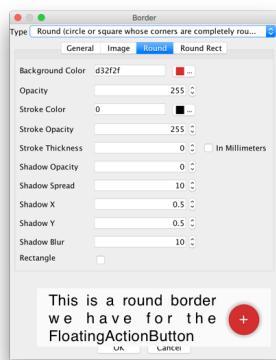
When a border is defined it overrides the background & color. That's why it's important to define it as empty sometimes. The Toolbar has a shadow border defined on Android & we want to disable that so our background image will show. We can edit the border with the "... button & we can create a 9-piece border using the "Image Border Wizard"



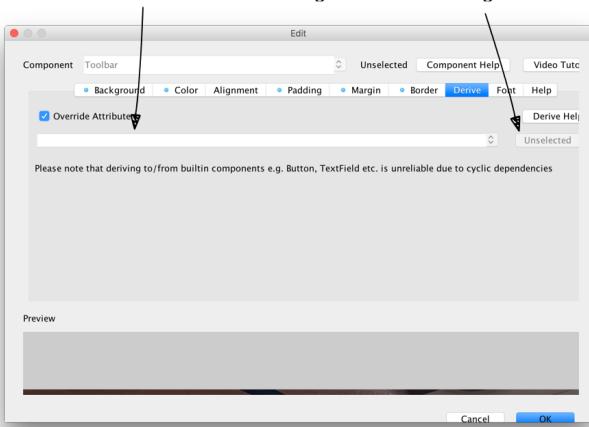
There are multiple simple border types in the border editing dialog but the ones we use most of all are the Round, RoundRect, Line,



When we select the Round Border we can create either a circle or a pill shape (by activating the rectangle)



Derive inherits the styling of the given UIID you can type the UIID on the left and select the right state on the right



The Font UI is somewhat confusing due to a heavy dose of legacy features. For modern applications it makes sense to pick a native true type font from the combo box here and size it in millimeters

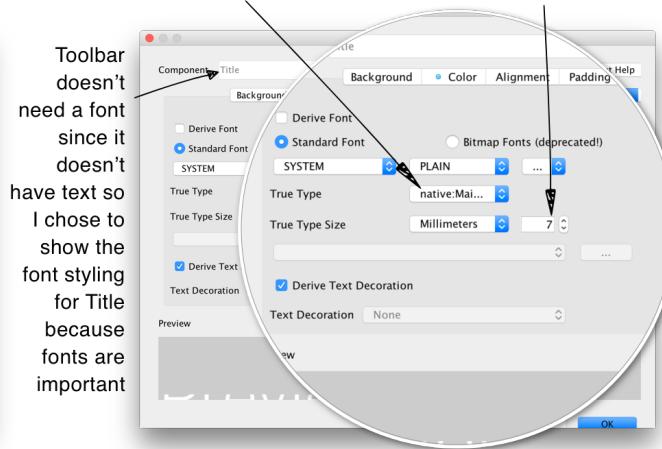


Figure 56. The Rest of the Add Theme Entry Dialog - Part II

We'll cover these options in-depth in the theming chapters.

Native Theme

By default Codename One applications are created with a theme that derives from the builtin OS native theme. You can add additional themes into the resource file by pressing the **Add A New Theme** button. You can also create multiple themes and ship them with your app or download them dynamically.

You can create a theme from scratch or customize one of the Codename one themes to any look you desire.



To preview the look of your theme in the various platforms use the **Native Theme menu option** in the designer

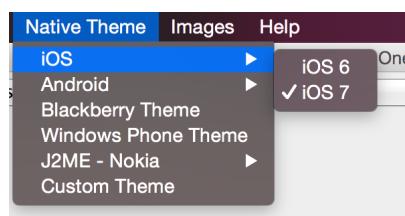


Figure 57. The native theme menu option

You can easily create deep customizations that span across all themes by adding a UIID or changing an existing UIID. E.g. looking at the getting started application that ships with the plugin you will notice a green button. This button is defined using the "GetStarted" UIID which is defined within the designer as:

- A green background
- White foreground
- A thin medium sized font
- Center aligned text
- A small amount of spacing between the text and the edges

To achieve the colors we [define them in the color tab](#).



We define the transparency to 255 which means the background will be a solid green color. This is important since the native OS's might vary with the default value for background transparency so this should be defined explicitly

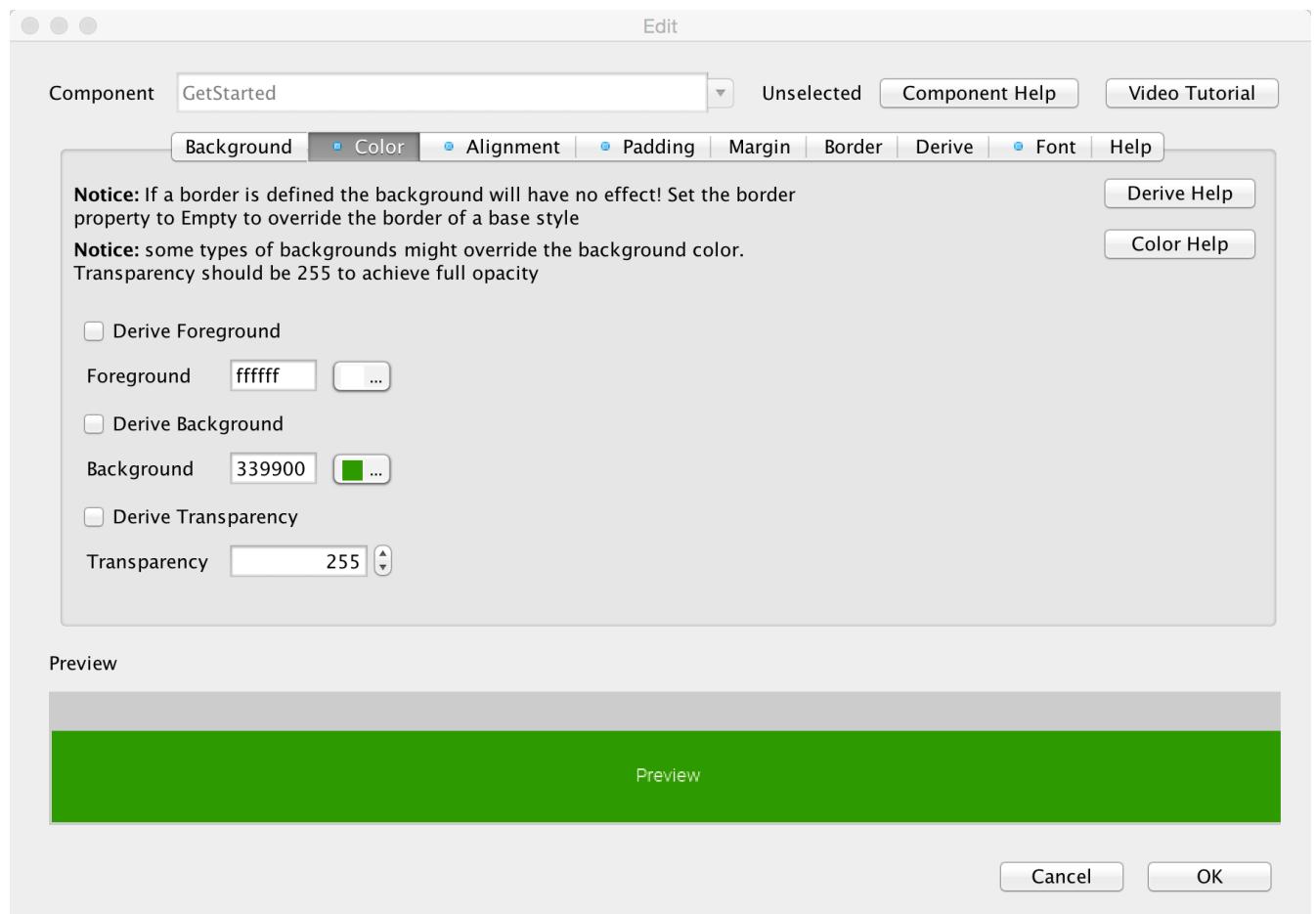


Figure 58. The Color tab for the get started button theme settings

The alignment of the text is pretty simple, notice that the alignment style attribute applies to text and doesn't apply to other elements. To align other elements we use layout manager logic.

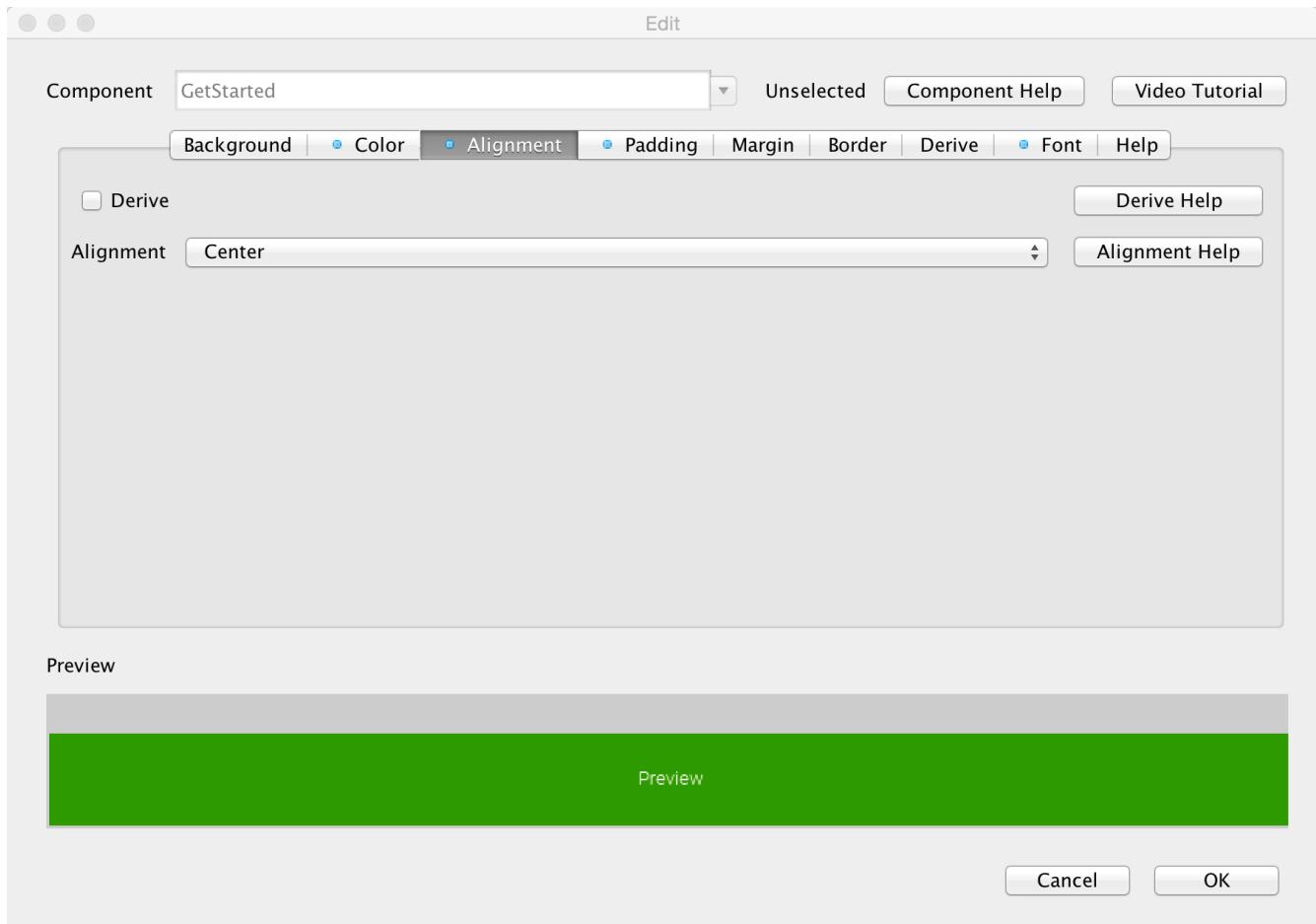


Figure 59. The alignment tab for the get started button theme settings

Padding can be expressed in pixels, millimeters (approximate) or percentages of the screen size.



We recommend using millimeters for all measurements to keep the code portable for various device DPI's.

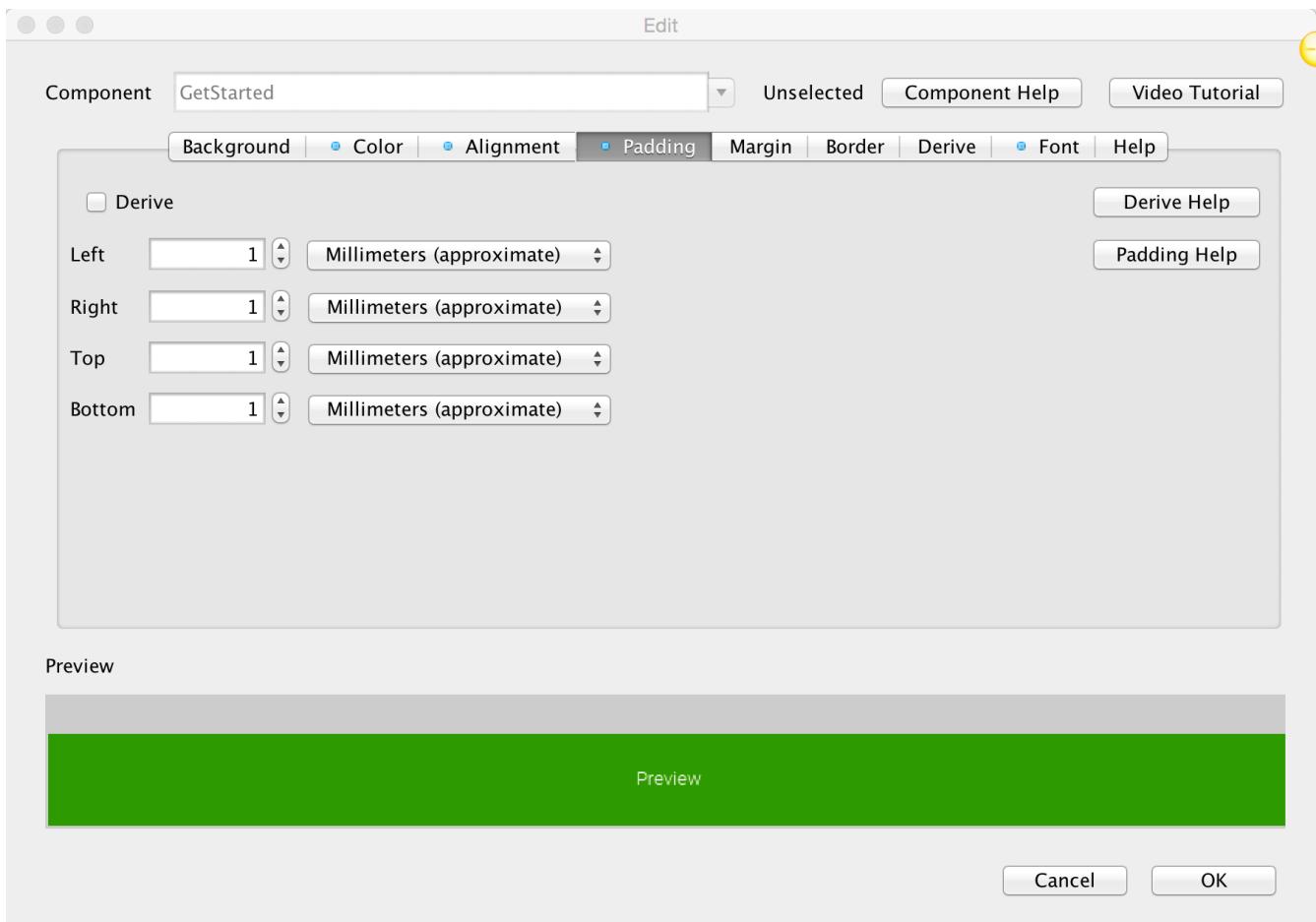


Figure 60. The padding tab for the get started button theme settings

The font uses native OS light font but has a fallback for older OS's that don't support truetype fonts. The "True Type" font will be applicable for most modern OS's. In the case of the "native:" fonts Android devices will use Roboto whereas iOS devices will use Helvetica Neue. You can supply your own TTF and work with that.



Since Codename One cannot legally ship Helvetica Neue fonts the simulator will fallback to Roboto on PC's.



At the time of this writing the desktop/simulator version of the Roboto font doesn't support many common character sets (languages). This will have no effect on an Android device where the native font works properly.

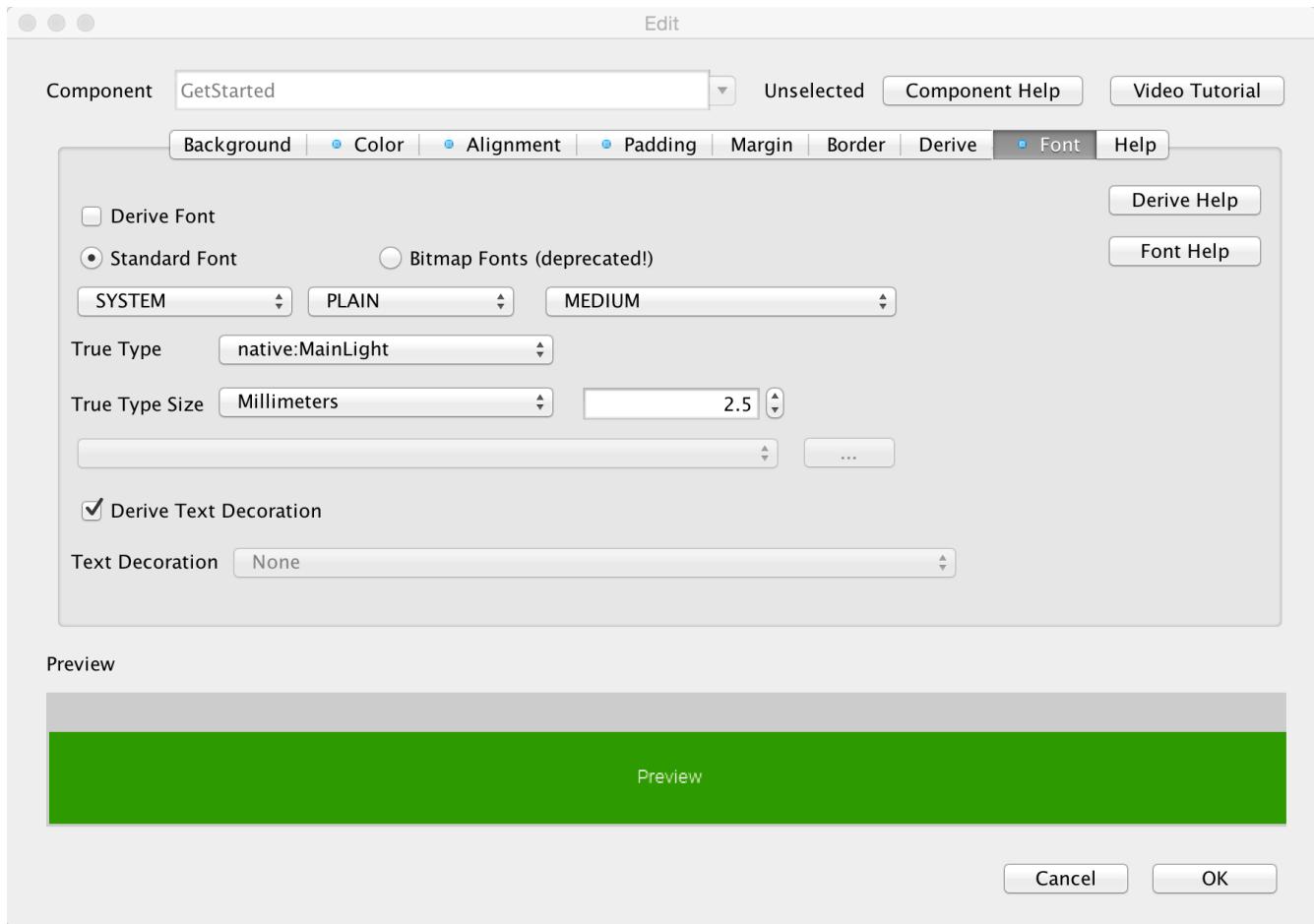


Figure 61. The font tab for the get started button theme settings

2.3. GUI Builder

The GUI builder allows us to arrange components visually within a UI using drag and drop, property sheets etc. With the GUI builder we can create elaborate, rich UI's without writing the layout code.

Why two GUI Builders?

The original old GUI builder has its roots in our work at Sun Microsystems, it was developed directly into the designer tool and stores its data as part of the resource file. When creating an application for the old GUI builder you must define it as a "visual application" which will make it use the old GUI builder.

The roots of this GUI builder are pretty old. When we initially built it we still had to support feature phones with 2mb of RAM and the iPad wasn't announced yet. Due to that we picked an architecture that made sense for those phones with a greater focus on navigation and resource usage. Newer mobile applications are rivaling desktop applications in complexity and in those situations the old GUI builder doesn't make as much sense

The old GUI builder is in the designer tool, it's a Swing application that includes the theme design etc.

It generates a **Stateemachine** class that contains all the main user GUI interaction code.

The new GUI builder is a standalone application that you launch from the right click menu by selecting a form as explained below. Here are screenshots of both to help you differentiate:

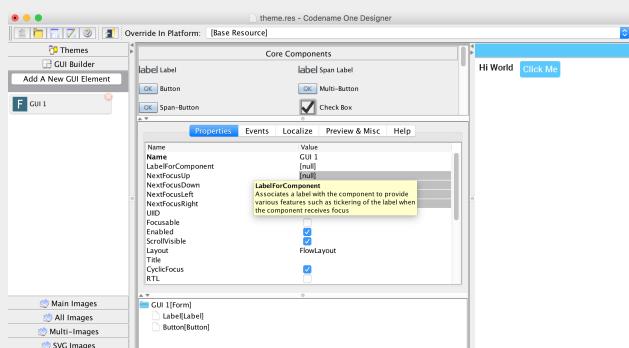


Figure 62. The old GUI builder

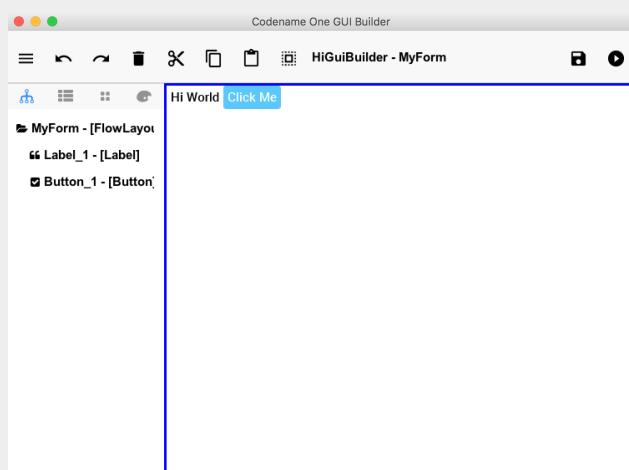


Figure 63. The same UI in the new GUI builder

As of version 3.7, the new GUI Builder also supports an auto layout mode which allows you to freely position and resize components on a canvas. This mode is now the default for new GUI forms, and it always uses LayeredLayout as the root layout manager.

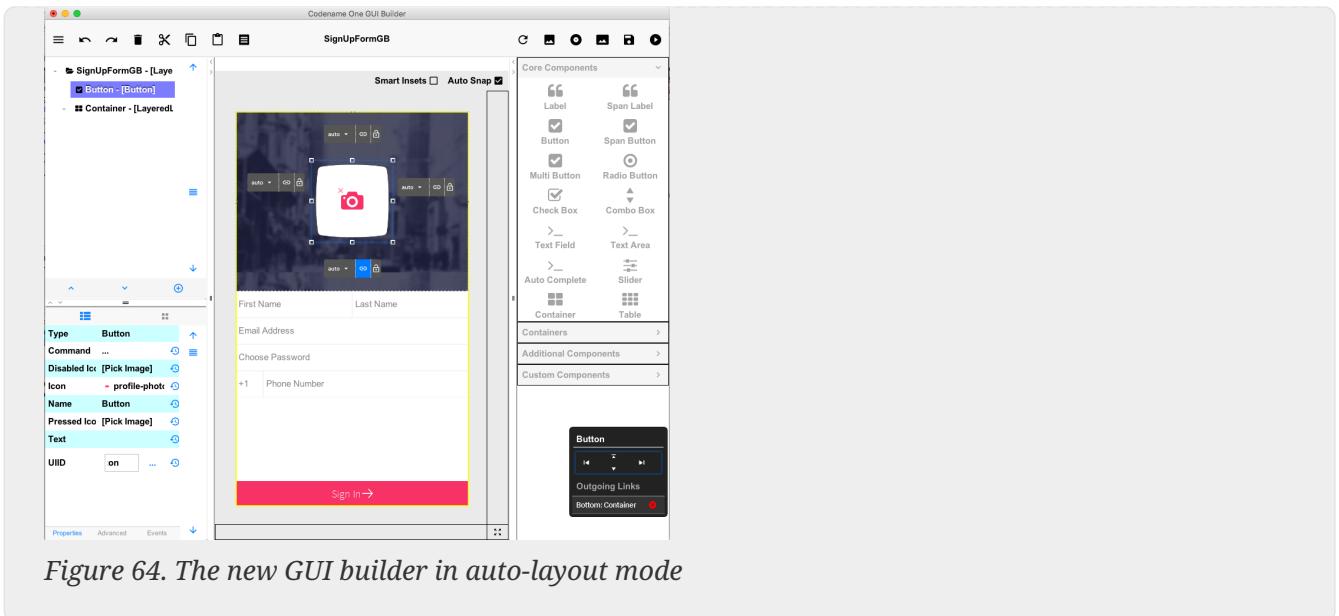


Figure 64. The new GUI builder in auto-layout mode

2.3.1. Hello World

Creating a hello world app in the new GUI builder is actually pretty trivial, you need to start with a regular handcoded application. Not a GUI builder application as it refers to the old GUI builder!

This is the exact same hello world application we created before...

Following are the instructions for creating a form and launching the GUI builder. While they are similar there are minor IDE differences. Usage of the GUI builder is identical in all IDE's as the GUI builder is a separate application.

NetBeans

In NetBeans you need to follow these 4 steps:

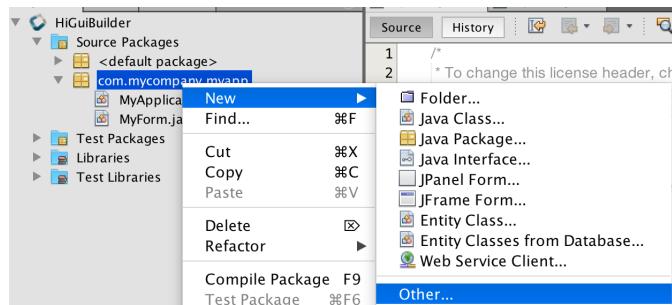


Figure 65. Right click the package select **New** → **Other**

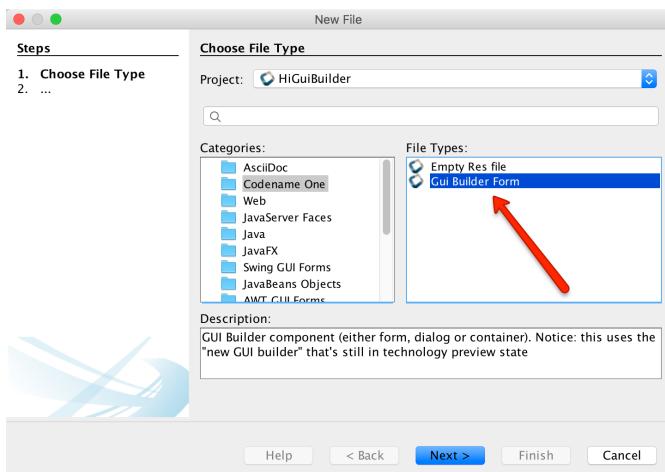


Figure 66. In the Codename One section select the GUI builder form

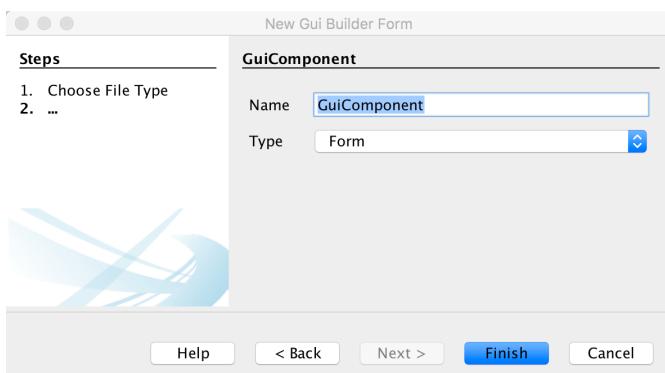


Figure 67. Type in the name of the form and click finish, you can change the type to be a Container or Dialog

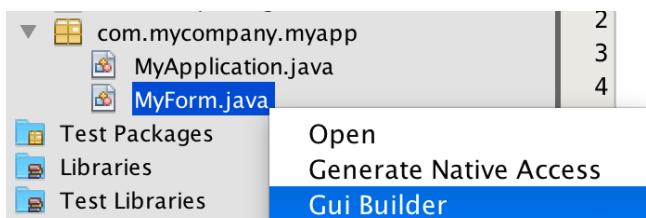


Figure 68. Launch the GUI builder thru the right click menu on the newly created file

IntelliJ/IDEA

In IntelliJ you need to follow these 3 steps:

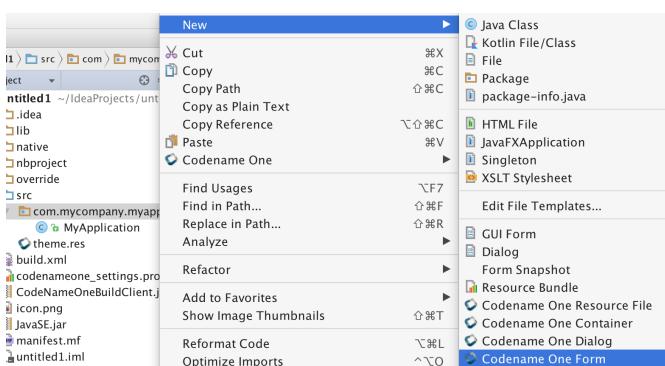


Figure 69. Right click the package select **New** → **Codename One AutoLayout Form** (or Dialog/Container)



Figure 70. Type in a name for the new form

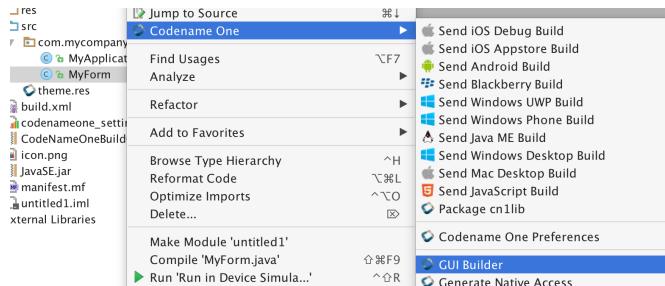


Figure 71. Launch the GUI builder thru the right click menu on the newly created file

Eclipse

In Eclipse you need to follow these 4 steps:

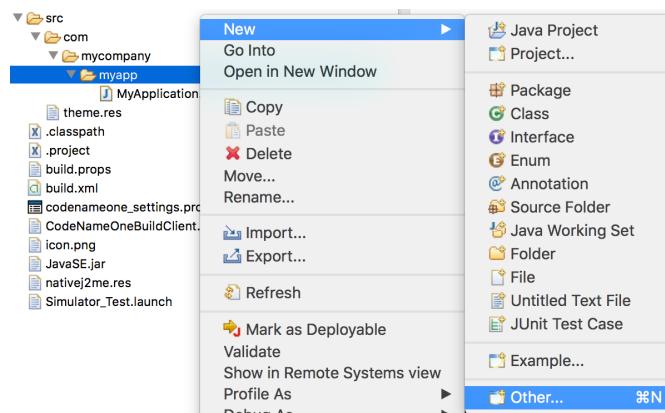


Figure 72. Right click the package select **New** → **Other**

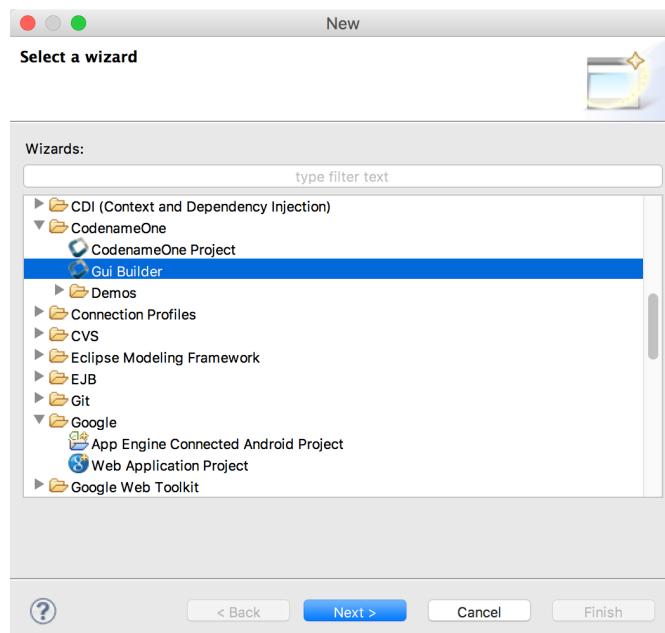


Figure 73. In the Codename One section select the GUI builder option

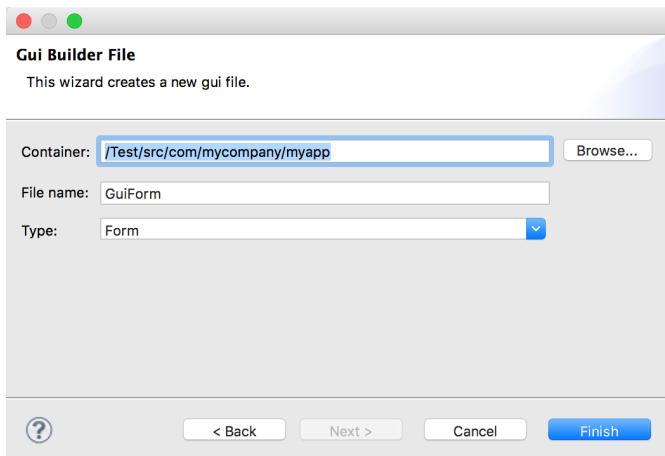


Figure 74. Type in the name of the form and click finish, you can change the type to be a Container or Dialog

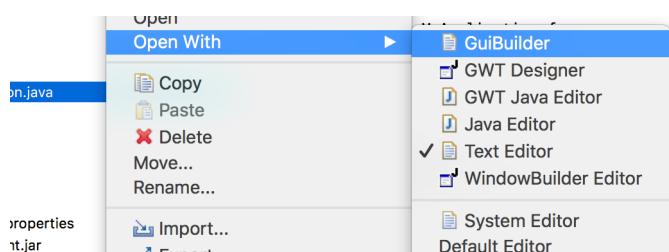


Figure 75. Launch the GUI builder thru the right click menu on the newly created file

Basic Usage

Notice that the UI of the new GUIBuilder might change in various ways but the basic concepts should remain the same.

The GUI builder is controlled via it's main toolbar, notice that your changes will only be applied when you click the **Save** button on the right:

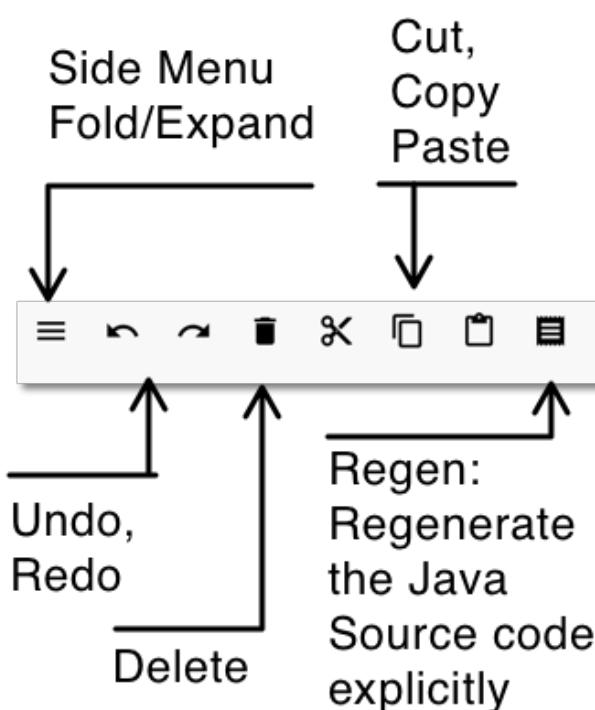


Figure 76. The features of the left toolbar

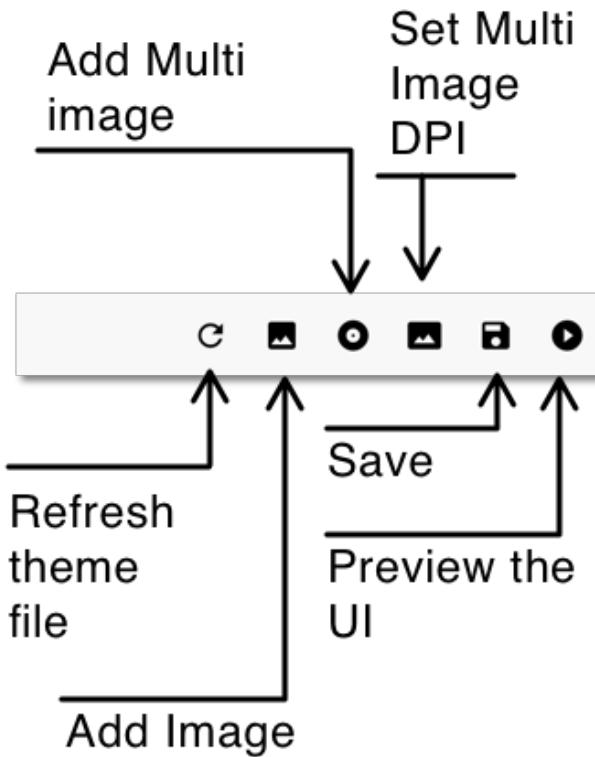


Figure 77. The features of the right toolbar

The main UI includes four important parts:

- **Main Form** — This is where we place the components of the UI we are building
- **Component Tree** — This is a logical representation of the component hierarchy within the **Main Form**. It's often easier to pick a component from the tree rather than the form itself
- **Property Inspector** — When we select an element in the tree or form we can see its details here. We can then edit the various details of the component in this area
- **Palette** — Components can be dragged from the palette to the **Main Form** and placed in the UI

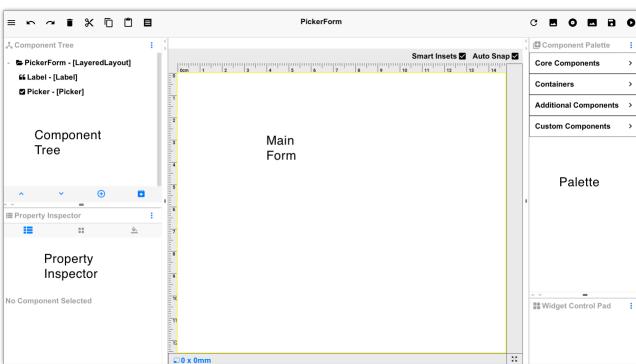
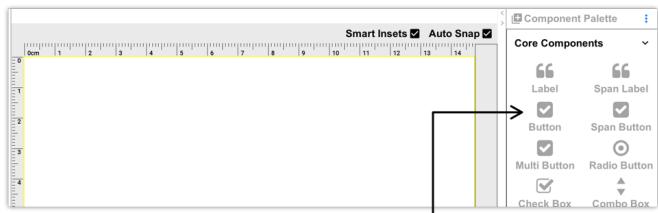


Figure 78. The four parts of the GUI builder

We'll start by selecting the **Component Palette** and dragging a button into the UI:



We Drag the Button from the Palette to the Main Form

This is what we end up with, we can still drag and reposition the button

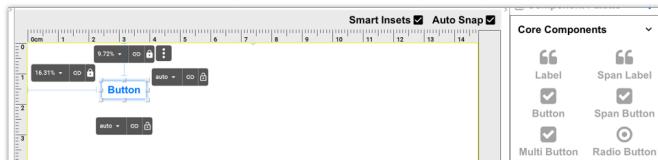


Figure 79. You can drag any component you want from the palette to the main UI

By default the auto-layout mode of the GUI builder uses layered layout to position components. Sides can be bound to a component or to the **Form**. We then use distance units to determine the binding behavior. The GUI builder tries to be "smart" and guesses your intention as you drag the components along.

When you select the component you placed you can edit the properties of that component:

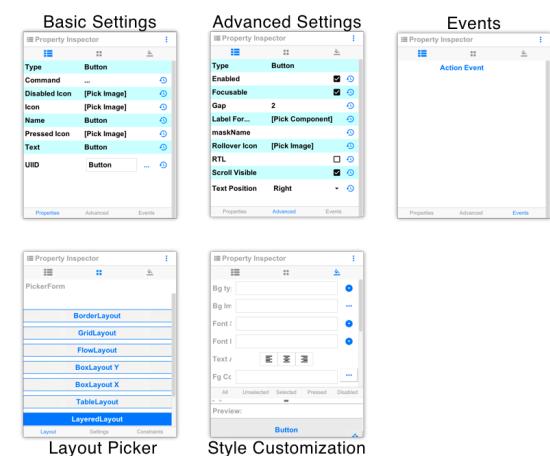


Figure 80. Properties allow you to customize everything about a component

There are five property sheets per component:

- **Basic Settings** — These include the basic configuration for a component e.g. name, icon, text etc.
- **Advanced Settings** — These include features that aren't as common such as icon gap, mask etc.
- **Events** — By clicking a button in this tab a method will be added to the source file with a callback matching your component name. This will let you bind an event to a button, text field etc.
- **Layout** — You can determine the layout of the parent **Container** here. For auto-layout this should stay as layered layout, however you can nest other layout types in here
- **Style Customization** — This isn't a theme, if you want to customize the style of a specific component you can do that through this UI. The theme works on a more global/reusable level and this is designed for a specific component only

For things like setting the text on the component we can use a convenient "long click" on the

component to edit the text in place as such:



Figure 81. Use the long click to edit the text "in place"

Events

When a component supports broadcasting events you can bind such events by selecting it, then selecting the events tab and clicking the button matching the event type



Figure 82. The events tab is listed below supported event types can be bound above

Once an event is bound the IDE will open to the event code e.g.:

```
public void onButton_1.ActionEvent(com.codename1.ui.events.ActionEvent ev) {  
}
```



Some IDE's only generate the project source code after you explicitly build the project so if your code needs to access variables etc. try building first

Within the code you can access all the GUI components you defined with the `gui_` prefix e.g. `Button_1` from the UI is represented as:

```
private com.codename1.ui.Button gui_Button_1 = new com.codename1.ui.Button();
```

Underlying XML

Saving the project generates an XML file representing the UI into the res directory in the project, the GUI file is created in a matching hierarchy in the project under the `res/guibuilder` directory:

Name	Date Modified	Size
build.xml	10 Jun 2016, 5:03 PM	22 KB
codenameone_settings.properties	10 Jun 2016, 5:03 PM	951 bytes
CodeNameOneBuildClient.jar	10 Jun 2016, 5:03 PM	257 KB
icon.png	10 Jun 2016, 5:03 PM	168 KB
iosCerts	19 Jun 2016, 2:15 PM	--
JavaSE.jar	10 Jun 2016, 5:03 PM	17.7 MB
lib	20 Jun 2016, 6:24 AM	--
manifest.mf	10 Jun 2016, 5:03 PM	85 bytes
native	10 Jun 2016, 5:03 PM	--
nbproject	10 Jun 2016, 5:03 PM	--
override	10 Jun 2016, 5:03 PM	--
res	Today, 2:25 PM	--
guibuilder	Today, 2:25 PM	--
com	Today, 2:25 PM	--
mycompany	Today, 2:25 PM	--
myapp	10 Jun 2016, 5:05 PM	--
MyForm.gui	Today, 12:44 PM	366 bytes
src	Today, 2:25 PM	--
com	Today, 2:25 PM	--
mycompany	Today, 2:26 PM	--
myapp	10 Jun 2016, 5:05 PM	--
MyApplication.java	10 Jun 2016, 5:08 PM	1 KB
MyForm.java	17 Jun 2016, 1:25 PM	3 KB
theme.res	10 Jun 2016, 5:03 PM	172 KB
test	10 Jun 2016, 5:08 PM	--

Figure 83. The java and GUI files in the hierarchy



If you refactor (rename or move) the java file it's connection with the GUI file will break. You need to move/rename both

You can edit the GUI file directly but changes won't map into the GUI builder unless you reopen it. These files should be under version control as they are the main files that change. The GUI builder file for the button and label code looks like this:

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

<component type="Form" layout="FlowLayout" flowLayoutFillRows="false" flowLayoutAlign="1" flowLayoutValign="0" title="My new title" name="MyForm">
  <component type="Button" text="Button" name="Button_1" actionEvent="true">
    </component>
  <component type="Label" text="Hi World" name="Label_1">
    </component>
  </component>
```

This format is relatively simple and is roughly the same format used by the old GUI builder which makes the migration to the new GUI builder possible. This file triggers the following Java source file:

```
package com.mycompany.myapp;

/**
 * GUI builder created Form
```

```

/*
 * @author shai
 */
public class MyForm extends com.codename1.ui.Form {

    public MyForm() {
        this(com.codename1.ui.util.Resources.getGlobalResources());
    }

    public MyForm(com.codename1.ui.util.Resources resourceObjectInstance) {
        initGuiBuilderComponents(resourceObjectInstance);
    }

    //-- DON'T EDIT BELOW THIS LINE!!!
    private com.codename1.ui.Label gui_Label_1 = new com.codename1.ui.Label();
    private com.codename1.ui.Button gui_Button_1 = new com.codename1.ui.Button();

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void guiBuilderBindComponentListeners() {
        EventCallbackClass callback = new EventCallbackClass();
        gui_Button_1.addActionListener(callback);
    }

    class EventCallbackClass implements com.codename1.ui.events.ActionListener,
    com.codename1.ui.events.DataChangedListener {
        private com.codename1.ui.Component cmp;
        public EventCallbackClass(com.codename1.ui.Component cmp) {
            this.cmp = cmp;
        }

        public EventCallbackClass() {
        }

        public void actionPerformed(com.codename1.ui.events.ActionEvent ev) {
            com.codename1.ui.Component sourceComponent = ev.getComponent();
            if(sourceComponent.getParent().getLeadParent() != null) {
                sourceComponent = sourceComponent.getParent().getLeadParent();
            }

            if(sourceComponent == gui_Button_1) {
                onButton_1ActionEvent(ev);
            }
        }

        public void dataChanged(int type, int index) {
        }
    }
    private void initGuiBuilderComponents(com.codename1.ui.util.Resources resourceObjectInstance) {
        guiBuilderBindComponentListeners();
        setLayout(new com.codename1.ui.layoutsFlowLayout());
        setTitle("My new title");
        setName("MyForm");
        addComponent(gui_Label_1);
        addComponent(gui_Button_1);
        gui_Label_1.setText("Hi World");
        gui_Label_1.setName("Label_1");
        gui_Button_1.setText("Click Me");
        gui_Button_1.setName("Button_1");
    }// </editor-fold>

    //-- DON'T EDIT ABOVE THIS LINE!!!
    public void onButton_1ActionEvent(com.codename1.ui.events.ActionEvent ev) {
    }
}

```



Don't touch the code within the DON'T EDIT comments...

The GUI builder uses the "magic comments" approach where code is generated into those areas to match the XML defined in the GUI builder. Various IDE's generate that code at different times. Some will generate it when you run the app while others will generate it as you save the GUI in the builder.

You can write code freely within the class both by using the event mechanism, by writing code in the constructors or thru overriding functionality in the base class.

2.3.2. Auto-Layout Mode

As of version 3.7, new forms created with the GUI Builder will use auto-layout mode. In this mode you can move and resize your components exactly as you see fit. You aren't constrained to the positions dictated by the form's layout manager.



All forms designed in auto-layout mode use LayeredLayout

Auto-Layout Mode is built upon the inset support in LayeredLayout. Component positioning uses **insets and reference components**, not absolute positioning

As an example, let's drag a button onto a blank form and see what happens. The button will be "selected" initially after adding, it so you'll see its outline, and resize handles for adjusting its size and position. You'll also see four floating labels (above, below, to the left, and to the right) that show the corresponding side's inset values and allow you to adjust them.

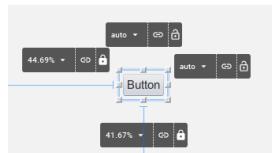


Figure 84. A button selected on the canvas in auto-layout mode. You can drag it to reposition it, or resize it using the resize handles.

Press the mouse inside the bounds of the button and drag it around to reposition it. You will notice that the inset labels change to reflect the new inset values. If you drag the button close to the edge of the form, the corresponding inset value will change to millimetres. If you move farther away from the edge, it will change to percentage values.

The Inset Control

Let's take a closer look at the inset control (the inset controls are the black buttons that appear to the top, bottom, left, and right of the selected component).



Figure 85. The inset control allows you to change the inset size and units, toggle it between fixed and flexible, and link it to another component.

Each control has three sections:

1. **The inset value drop-down menu.** This shows the current value of the inset (e.g. 0mm, 25%,

auto, etc...). If you click on this, it will open a menu that will allow you to change the units. If the inset is currently in millimetres, it will have options for pixels, and percent. If the inset is in percent, it will have options for pixels and millimetres. Etc.. It also includes a text field to enter a an inset value explicitly.



2.



The "Link" Button - If the inset is linked to a reference component, then this button will be highlighted "blue", and hovering over it will highlight the reference component in the UI so that you can clearly see which component it is linked to. Clicking on this button will open a dialog that will allow you to "break" this link. You can drag this button over any component in the form to "link".

3.



The "Lock" Button - This button allows you to toggle the inset between "flexible" (i.e. auto) and "fixed" (i.e. millimetres or percent).

Auto Snap

Notice the "auto-snap" checkbox that appears in the top-right corner of the designer.

Smart Insets Auto Snap

Figure 86. Auto-snap checkbox

Auto-snap does exactly what it sounds like: It automatically snaps two components together when you drag them near each other. This is handy for linking components together without having to explicitly link them (using the "link" button). This feature is turned on by default. If auto-snap is turned off, you can still initiate a "snap" by holding down the ALT/Option key on your keyboard during the drag.

Smart Insets

Beside the "auto-snap" checkbox is another checkbox named "Smart Insets".

Smart Insets Auto Snap

Figure 87. Smart insets checkbox

Smart Inset uses some heuristics during a drag to try to determine how the insets should be linked. Currently the heuristics are quite basic (it tries to link to the nearest neighbor component in most cases), but we will be working on improving this for future releases. This feature is turned off by default while it is still being refined. The goal is to improve this to the point where it **always** makes the correct link choices - at which time you will be able to use the designer without having any knowledge of insets or reference components.

The Widget Control Pad

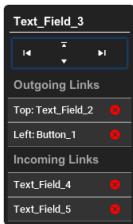


Figure 88. Widget control pad

When a component is selected, you should see a black floating panel appear in the lower right of the screen.

This is the widget control pad, and it provides an alternative view of the component's links. It also provides a useful list of incoming links (i.e. components that "depend on" this component's positioning). In some cases, you may want to disconnect incoming links so that you can drag the component without affecting the position of dependent components.

This control pad also includes game-pad-like controls (up, down, left, right), that allow you to "tab" the component to the next guide in that direction. Tab positions exist at component edges in the form. This is useful for aligning components with each other.

Keyboard Short-Cuts

- Arrow Keys** - Use the up/down/left/right arrow keys to nudge the currently selected component a little bit at a time. This is a convenient way to move the component to a position that is more precise than can easily be achieved with a mouse drag.
- Arrow Keys + SHIFT** - Hold down the SHIFT key while pressing an arrow key and it will "tab" the component to the next tab marker. The form has implicit tab markers at the edge of each component on the form.
- ALT/Option Key + Click or Drag** - Holding down the option/alt key while clicking or dragging a component will result in "snapping" behaviour even if auto-snap is turned off.

Sub-Containers

In some cases, you may need to add sub-containers to your form to aid in grouping your components together. You can drag a container onto your form using the "Container" palette item (under "Core Components"). The default layout the subcontainer will be LayeredLayout so that you are able to position components within the sub-container with precision, just like on the root container.

You can also change the layout of subcontainers to another classical layout manager (e.g. grid layout, box layout, etc..) and drag components directly into it just as you did with the old designer. This is very useful if parts of your form lend themselves. As an example, let's drag a container onto the canvas that uses BoxLayout Y. (You can find this under the "Containers" section of the component palette).

Drag the button (that was previously on the form) over that container, and you should see a drop-zone become highlighted.



Figure 89. Dropping container on child container with box layout y

You can drop the button directly there. You can As you drag more components into the sub-container, you'll see them automatically laid out vertically.

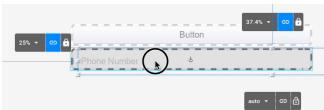


Figure 90. Box Layout Y dropping 2nd child

The Canvas Resize Tool

When designing a UI with the new designer it is **very** important that you periodically test the form's "resizing" behavior so that you know how it will behave on different devices. Components may appear to be positioned correctly when the canvas is one size, but become out of whack when the container is resized. After nearly every manipulation you perform, it is good practice to drag the canvas resize tool (the button in the lower right corner of the designer) smaller and bigger so you can see how the positions are changed. If things grow out of whack, you may need to toggle an inset between fixed and auto, or add a link between some of the components so that the resizing behavior matches your expectations.

3. Theme Basics

This chapter covers the creation of a simple hello world style theme and its visual customization. It uses the Codename One Designer tool to demonstrate basic concepts in theme creation such as 9-piece borders, selectors and style types. We would recommend reviewing this even if you end up using CSS.

3.1. Understanding Codename One Themes

Codename One themes are pluggable CSS like elements that allow developers to determine/switch the look of the application in runtime. A theme can be installed via the [UIManager class](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html>] and themes can be layered one on top of the other (like CSS). A theme can be generated from CSS as well, we'll cover this (and the supported CSS syntax) soon.

By default, Codename One themes derive the native operating system themes, although this behavior is entirely optional.

A theme initializes the [Style](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html>] objects, which are then used by the components to render themselves or by the [LookAndFeel](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/LookAndFeel.html>] and [DefaultLookAndFeel](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/DefaultLookAndFeel.html>] classes to create the appearance of the application.

Codename One themes have some built-in defaults. E.g. borders for buttons and padding/margin-opacity for various components. These are a set of “common sense” defaults that can be overridden within the theme.

Codename One themes are effectively a set of UIID's mapped to a [Style](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html>] object. Codename One applications always have a theme, you can modify it to suit your needs and you can add multiple themes within the main resource file.

You can also add multiple resource files to a project and work with them. In code a theme is initialized using this code in your main class:

```
private Resources theme;

public void init(Object context) {
    theme = UIManager.initFirstTheme("/theme");
}
```

The [initFirstTheme](#) method is a helper method that hides some [try/catch](#) logic as well as some verbosity. This could be expressed as:

```

try {
    theme = Resources.openLayered("/theme");
    UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()[0]));
} catch(IOException e){
    e.printStackTrace();
}

```

3.2. Customizing Your Theme

We can launch the designer tool by double clicking on the `theme.res` file found in typical Codename One applications. In the left side you can see the section picker and within it the Theme section.

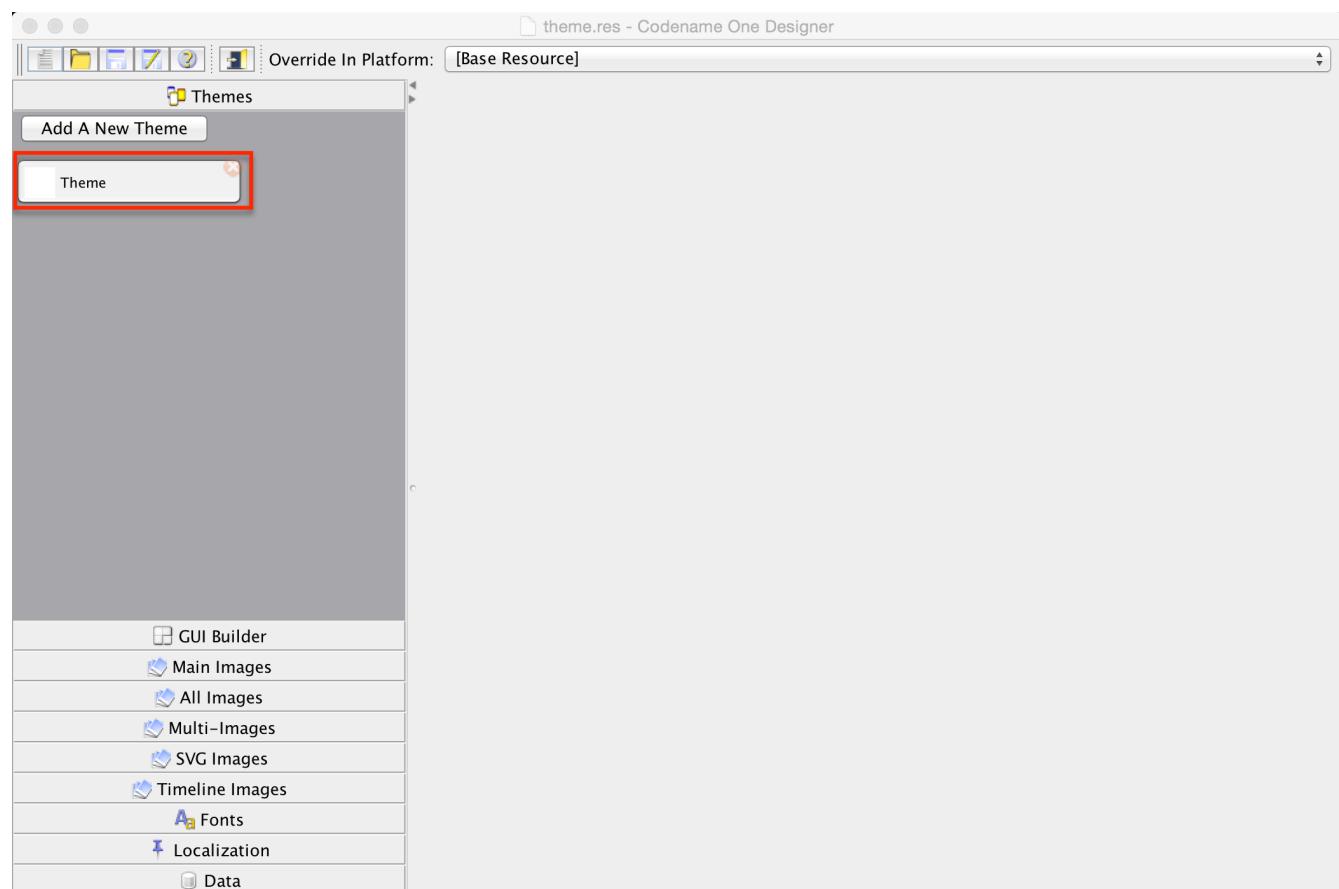


Figure 91. The theme area in the designer

When you select the theme you will see the theme default view.

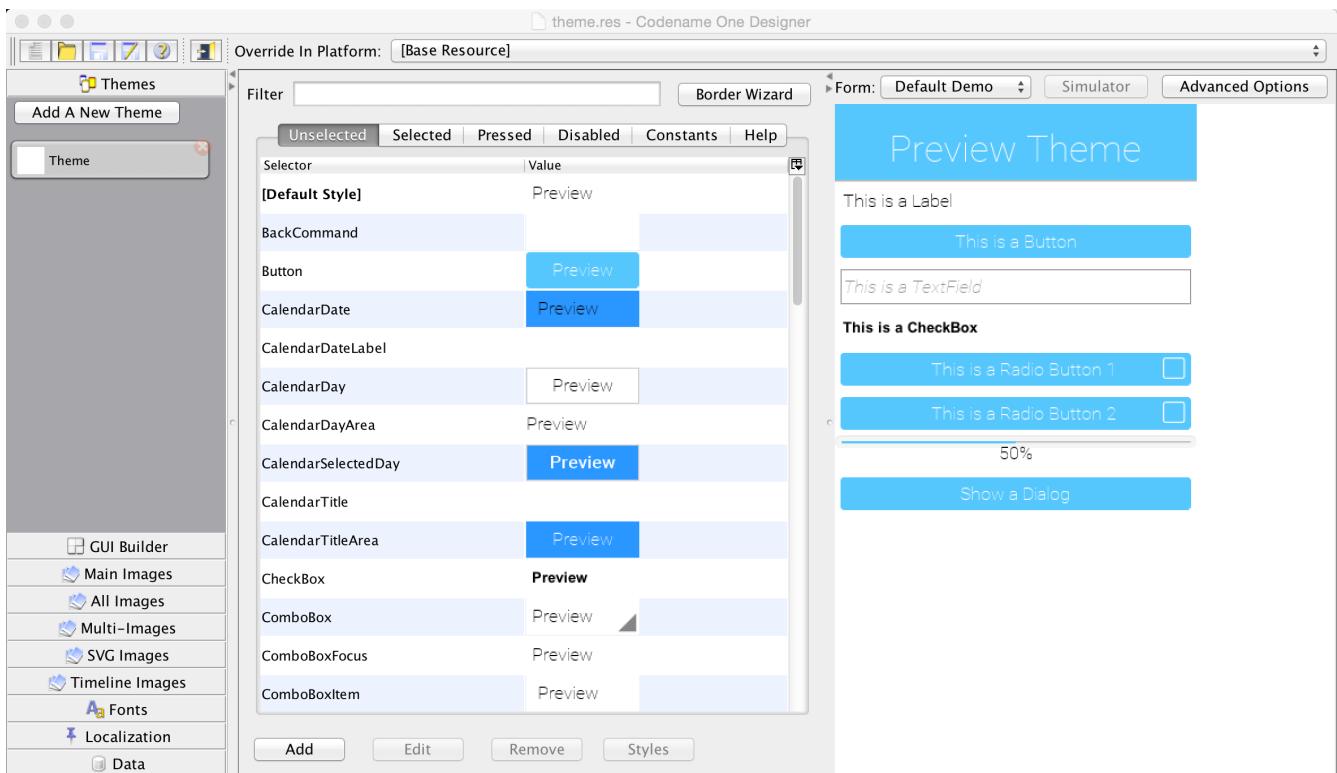


Figure 92. Theme default view

There are several interesting things to notice here the preview section allows us to instantly see the changes we make to the theme data.

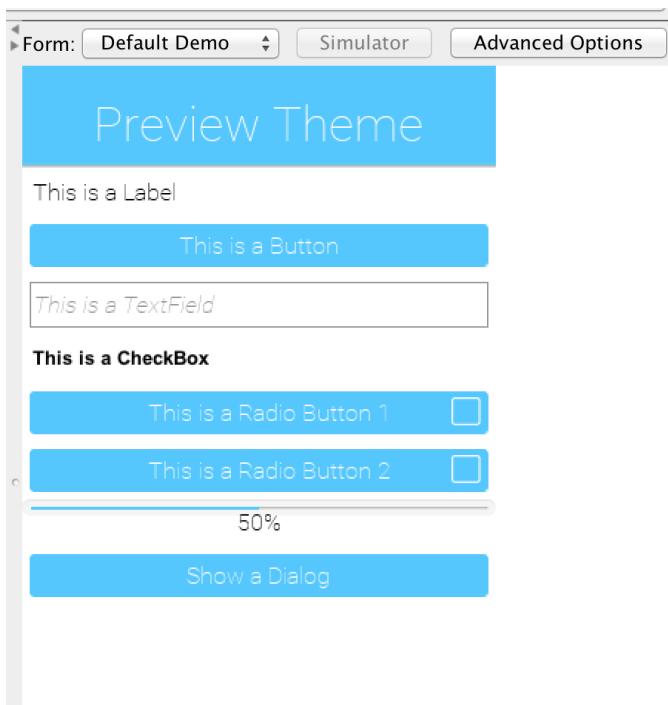


Figure 93. Theme preview section

The theme state tabs and constant tabs allow us to pass between the various editing modes for the theme and also add theme constants.

We discussed styles before, you can pick the right style mode through the tabs.



Figure 94. You can use these tabs to add the various types of styles and theme constants

The most important section is the style section. It allows us to add/edit/remove style UIID's.

Notice the **Default Style** section, it allows us to customize global defaults for the styles. Use it with caution as changes here can have wide implications.

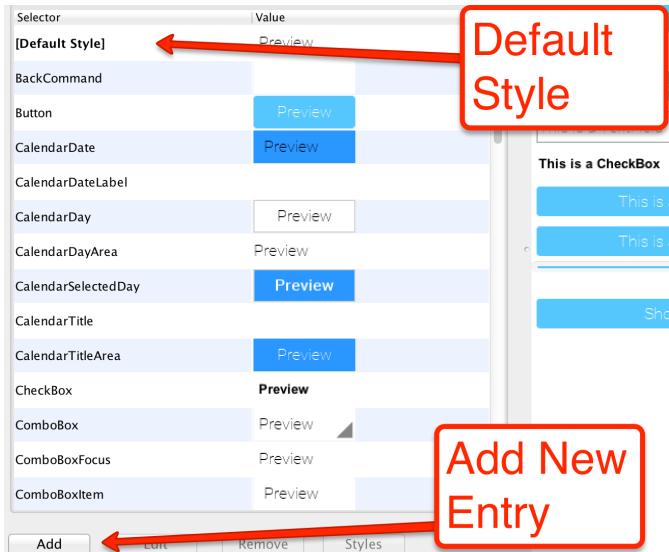


Figure 95. The theme selection area allows us to add, edit and delete entries. Notice the default style entry which is a unique special case

When we add an entry to the style we can just type the desired UIID into the box at the top of the dialog. We can also pick a UIID from the combo box but that might not include all potential options.



You can use the Component Inspector tool in the simulator to locate a component and its UIID in a specific **Form** [<https://www.codenameone.com/javadoc/com/codenname1/ui/Form.html>]

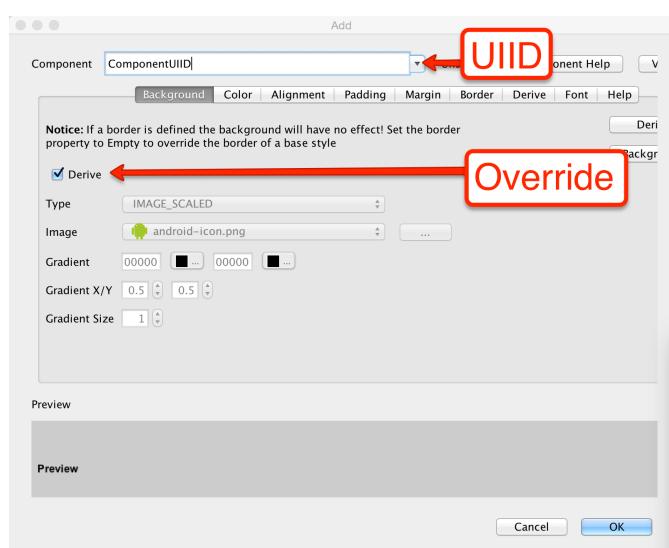


Figure 96. When pressing the Add/Edit entry we can edit a specific style entry UIID

When we add/edit an entry an important piece of the puzzle is the **Derive** check box that appears next to all of the UIID entries. All styles derive from the base style and usually from the native

theme defaults, so when this flag is checked the defaults will be used.

When you uncheck that checkbox the fields below it become editable and you can override the default behavior. To restore the default just recheck that flag.



A common oddity for developers is that when they press **Add** and don't derive any entry nothing is actually added. The entries in the theme are essentially key/value pairs so when you don't add anything there are no keys so the entry doesn't show up

3.3. Customizing The Title

The title is a great target for customization since it includes a few interesting "complexities".

The **Title** is surrounded by a **TitleArea** container that encloses it, above the title you will also see the **StatusBar** UIID that prevents the status details from drawing on top of the title text.

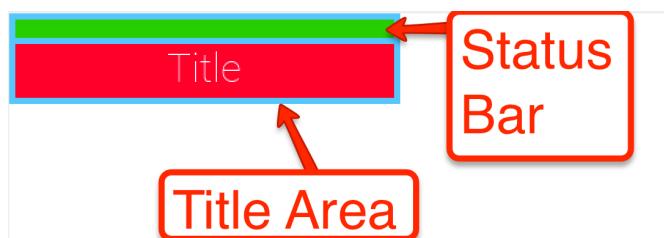


Figure 97. Title Area UIID's



The **StatusBar** UIID is a special case that is only there on iOS. In iOS the application needs to render the section under the status bar (which isn't the case for other OS's) and the **StatusBar** UIID was added so developers can ignore that behavior.

3.3.1. Background Priorities and Types

A slightly confusing aspect of styles in Codename One is the priorities of backgrounds. When you define a specific type of background it will override prior definitions, this even applies to inheritance.

E.g. if the theme defined a border for the **Button** UIID (a very common case) if you will try to define the background image or the background color of **Button** those will be ignored!



The solution is to derive the border and select the **Empty** border type

The order for UIID settings for background is as follows:

1. Border - if the component has a border it can override everything. Image borders always override all background settings you might have.
2. Image or gradient - an image or gradient background overrides background color/transparency settings.
3. Background Color/Transparency - If transparency is larger than 0 then this takes effect.

3.3.2. The Background Behavior and Image

Lets start in the first page of the style entry, we'll customize the background behavior for the **Title** UIID and demonstrate/explain some of the behaviors.

The pictures below demonstrate the different types of background image behaviors.

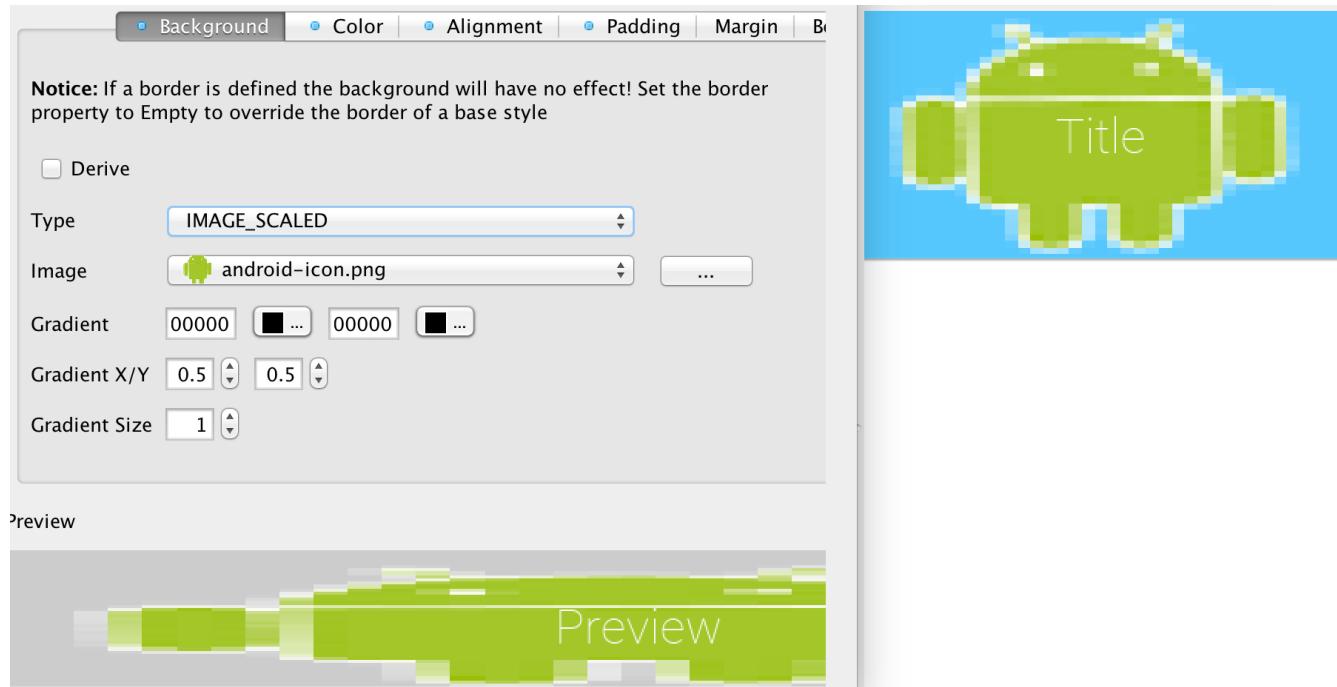


Figure 98. *IMAGE_SCALED* scales the image without preserving aspect ratio to fit the exact size of the component

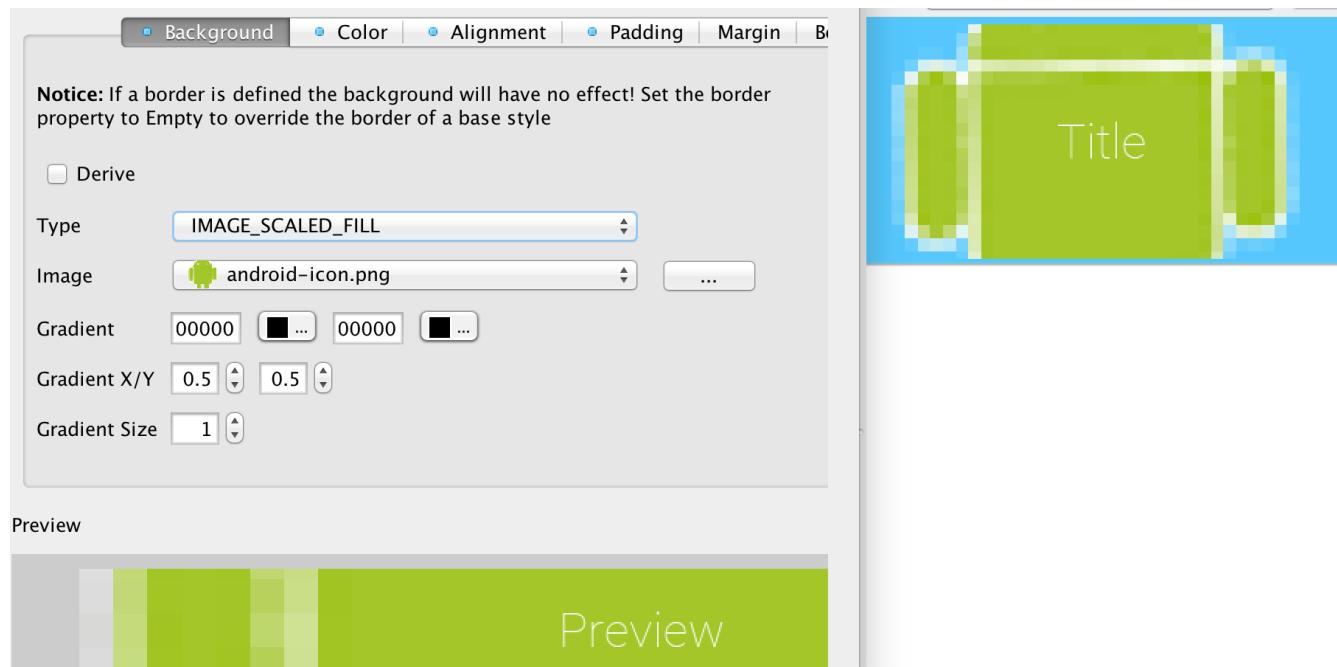


Figure 99. *IMAGE_SCALED_FILL* scales the image while preserving aspect ratio so it fills the entire space of the component



Aspect ratio is the ratio between the width and the height of the image. E.g. if the image is **100x50** pixels and we want the width to be 200 pixels preserving the aspect ratio will require the height to also double to **200x100**.

We highly recommend preserving the aspect ratio to keep images more "natural".

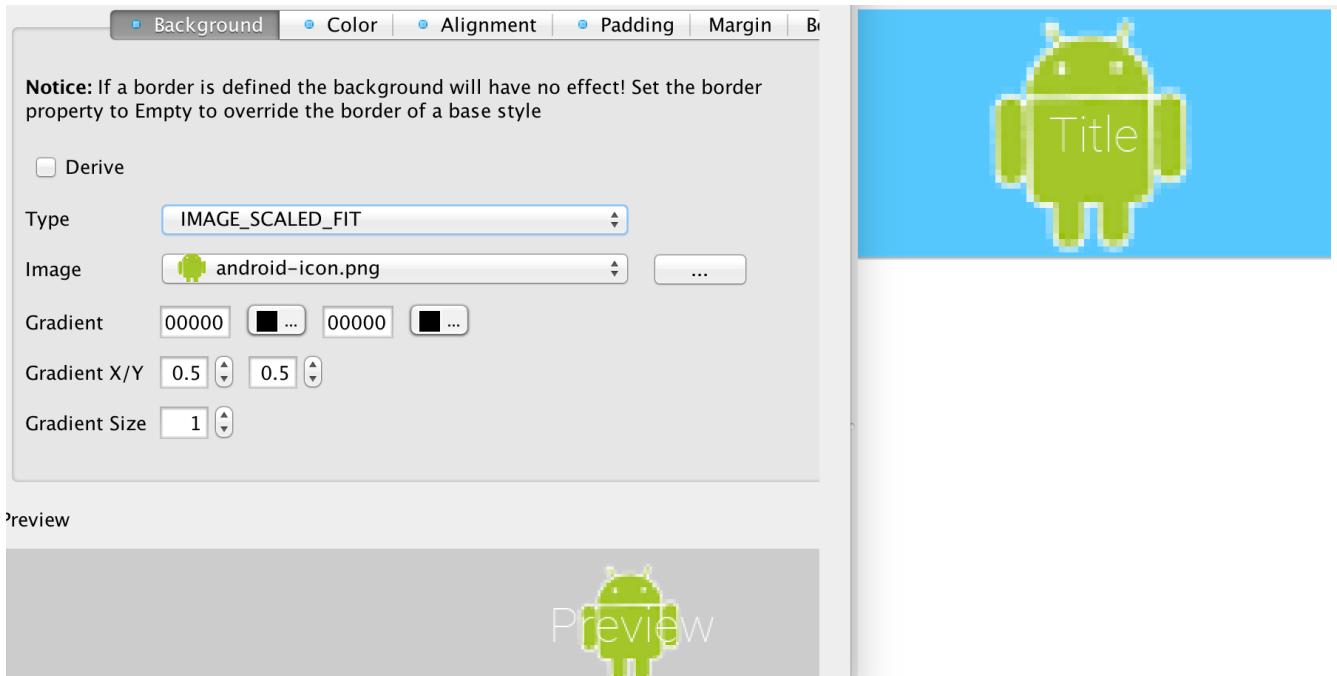


Figure 100. **IMAGE_SCALED_FIT** scales the image while preserving aspect ratio so it fits within the component

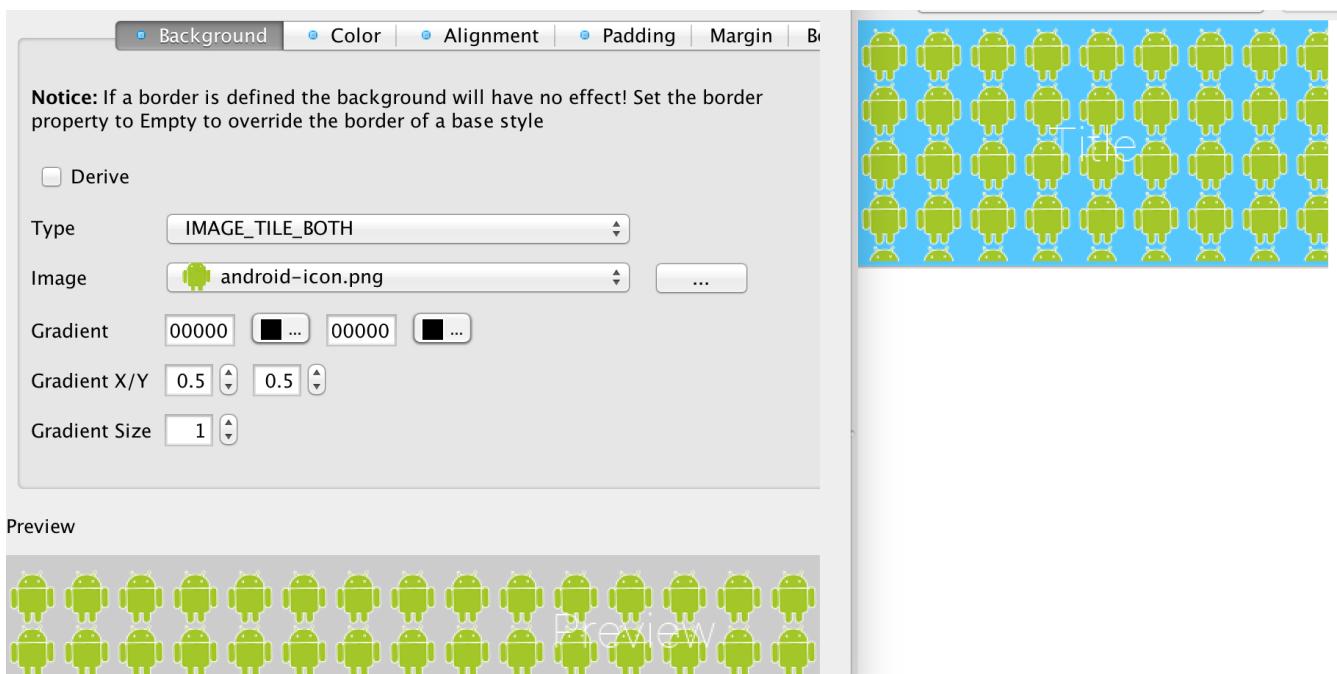


Figure 101. **IMAGE_TILE_BOTH** tiles the image on both axis of the component

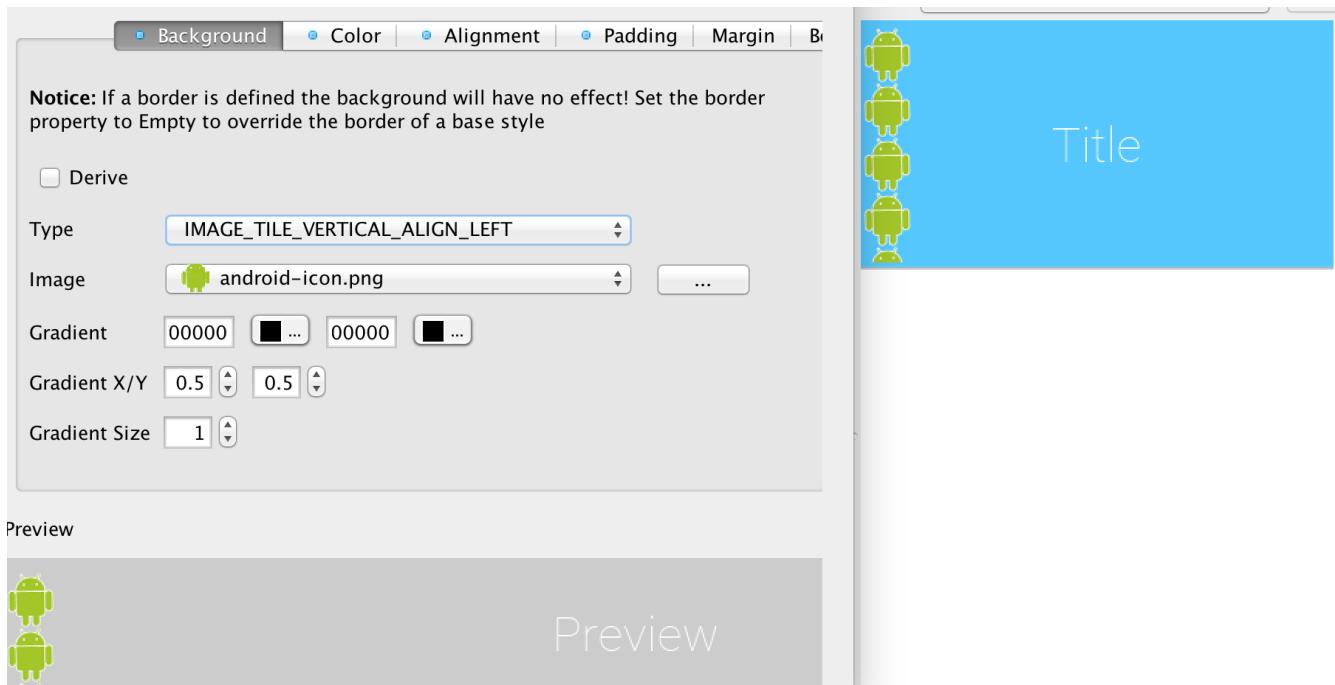


Figure 102. IMAGE_TILE_VERTICAL_ALIGN_LEFT tiles the image on the left side of the component

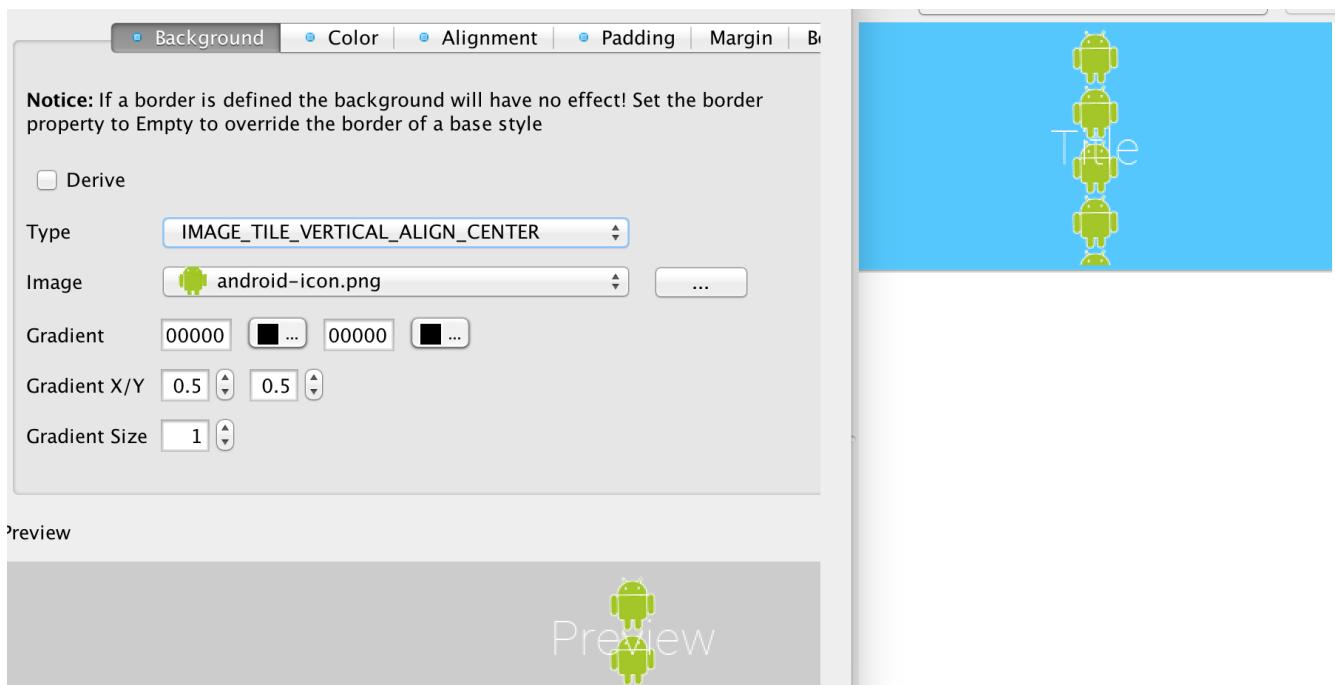


Figure 103. IMAGE_TILE_VERTICAL_ALIGN_CENTER tiles the image in the middle of the component

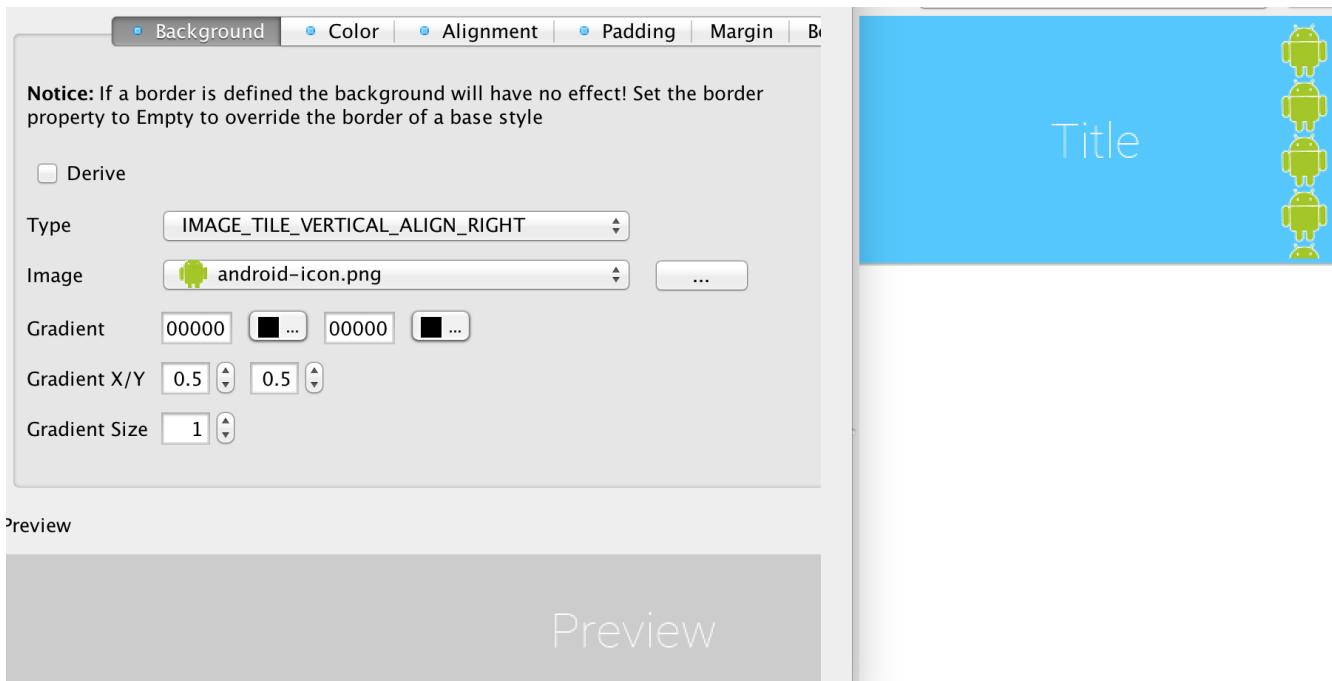


Figure 104. IMAGE_TILE_VERTICAL_ALIGN_RIGHT tiles the image on the right side of the component

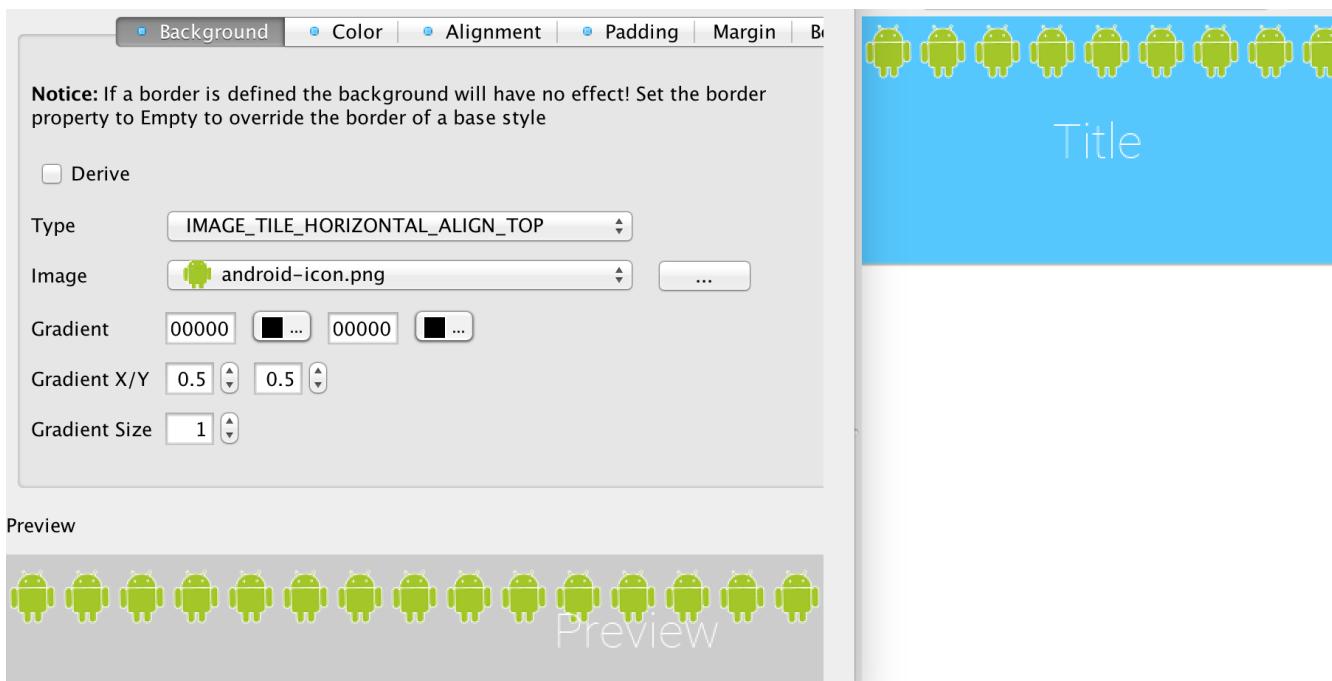


Figure 105. IMAGE_TILE_HORIZONTAL_ALIGN_TOP tiles the image on the top of the component

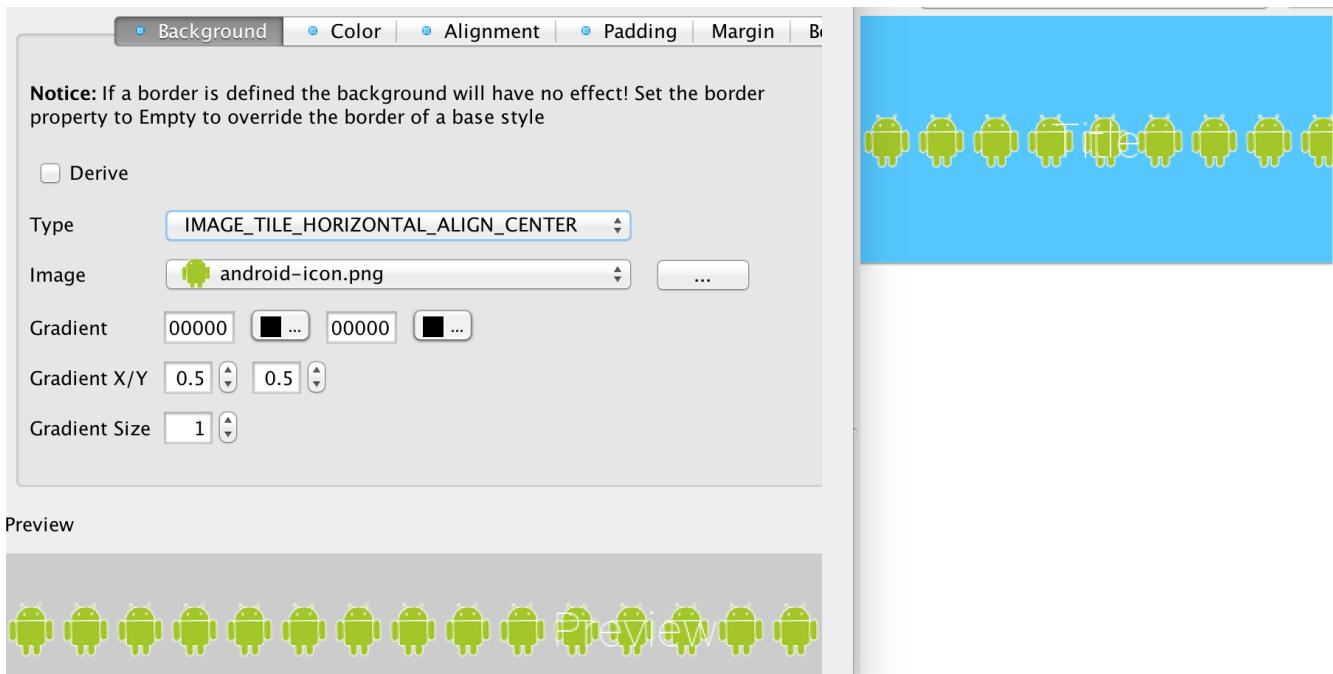


Figure 106. IMAGE_TILE_HORIZONTAL_ALIGN_CENTER tiles the image in the middle of the component

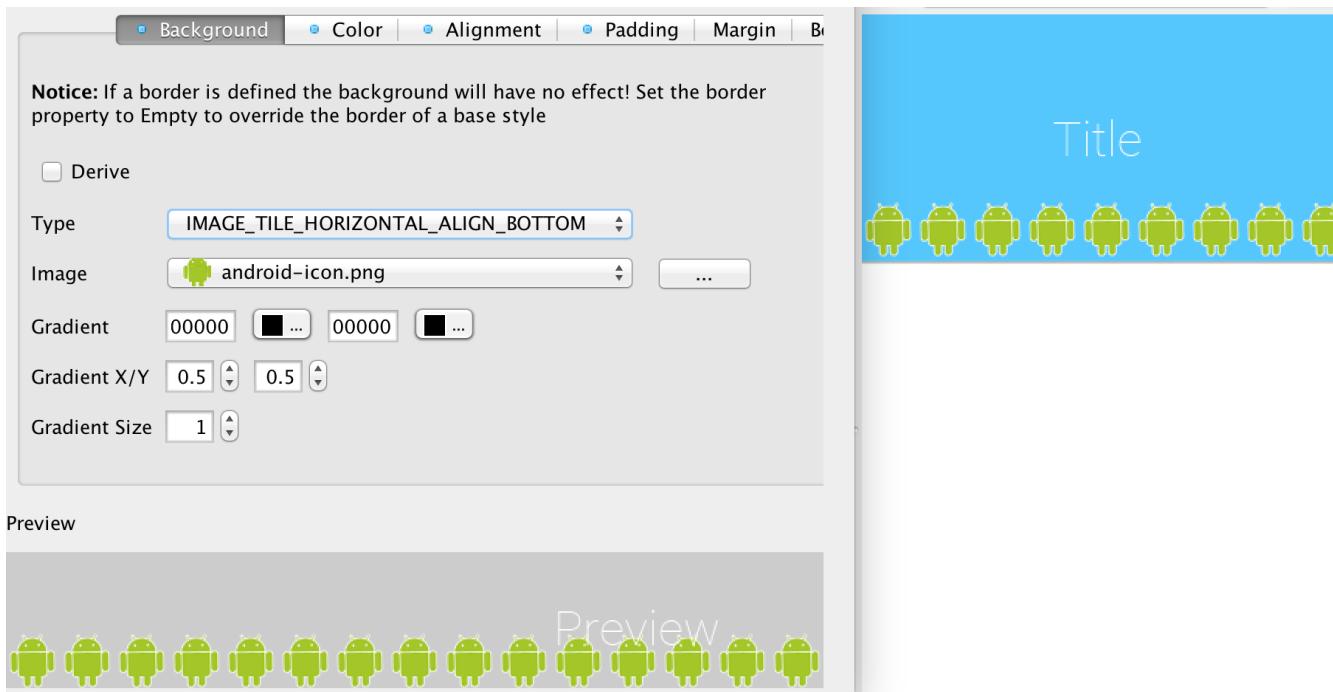


Figure 107. IMAGE_TILE_HORIZONTAL_ALIGN_BOTTOM tiles the image to the bottom of the component

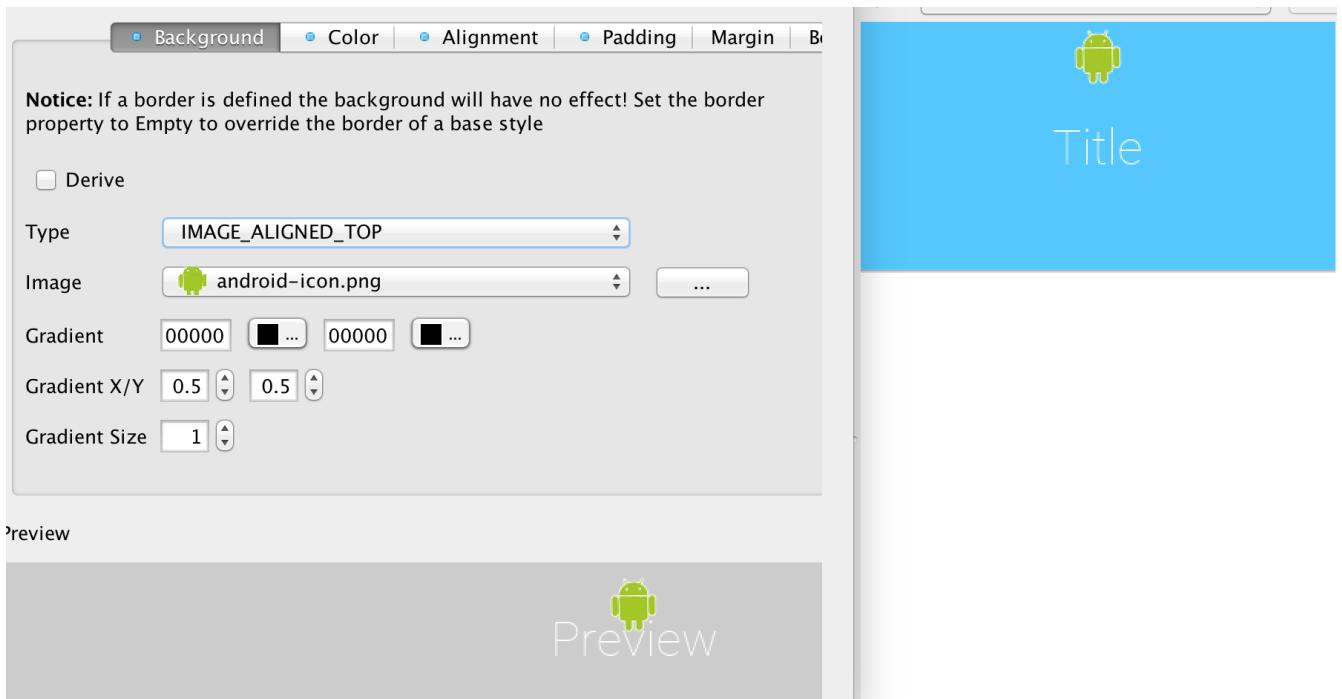


Figure 108. IMAGE_ALIGNED_TOP places the image centered at the top part of the component

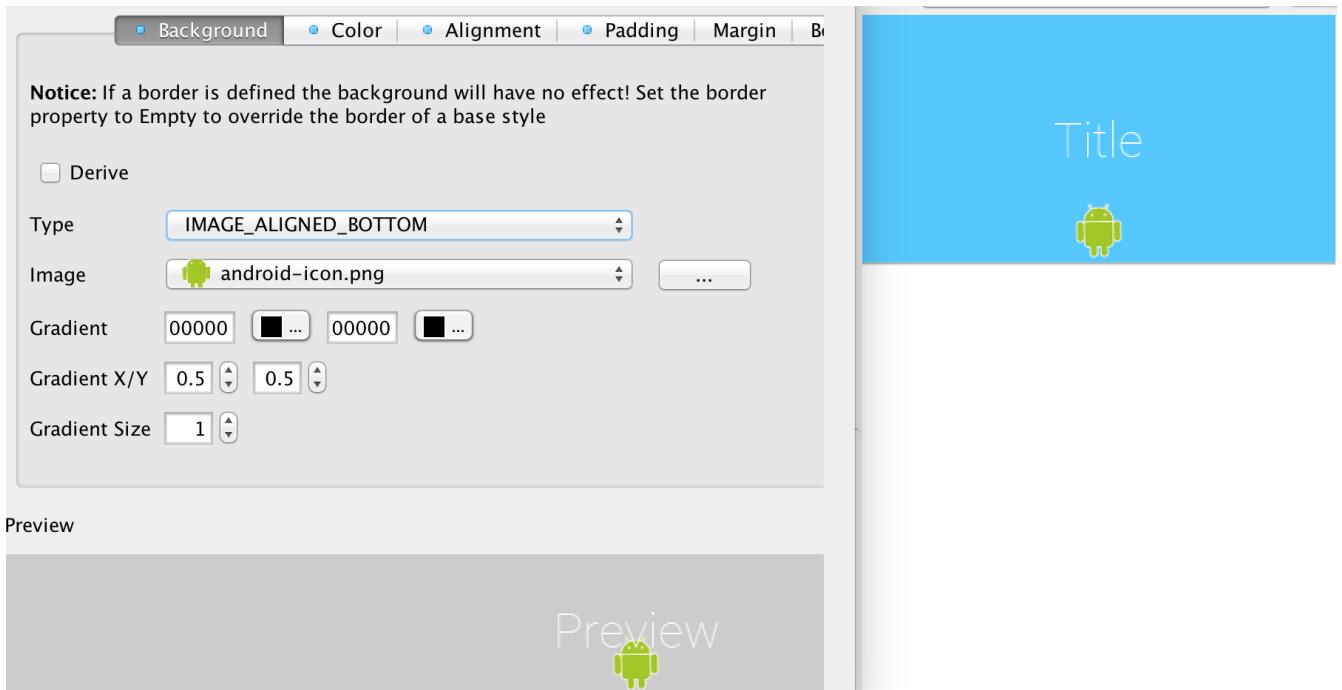


Figure 109. IMAGE_ALIGNED_BOTTOM places the image centered at the bottom part of the component

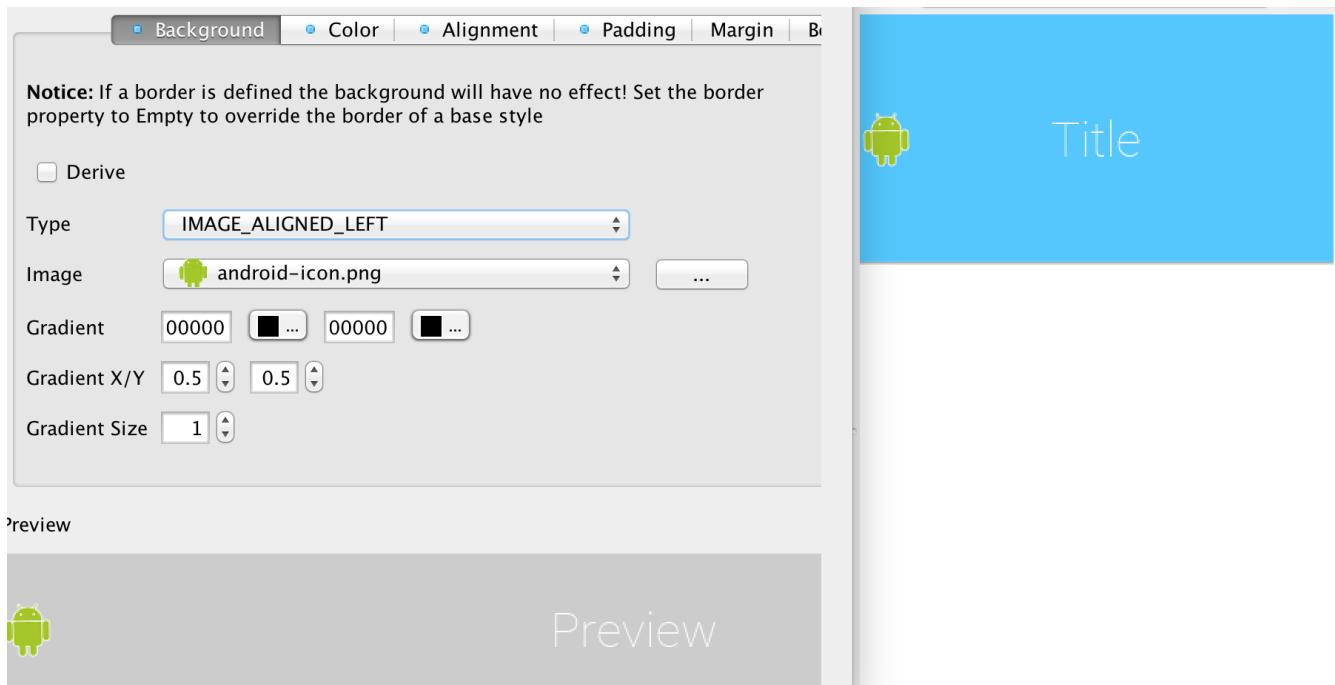


Figure 110. IMAGE_ALIGNED_LEFT places the image centered at the left part of the component

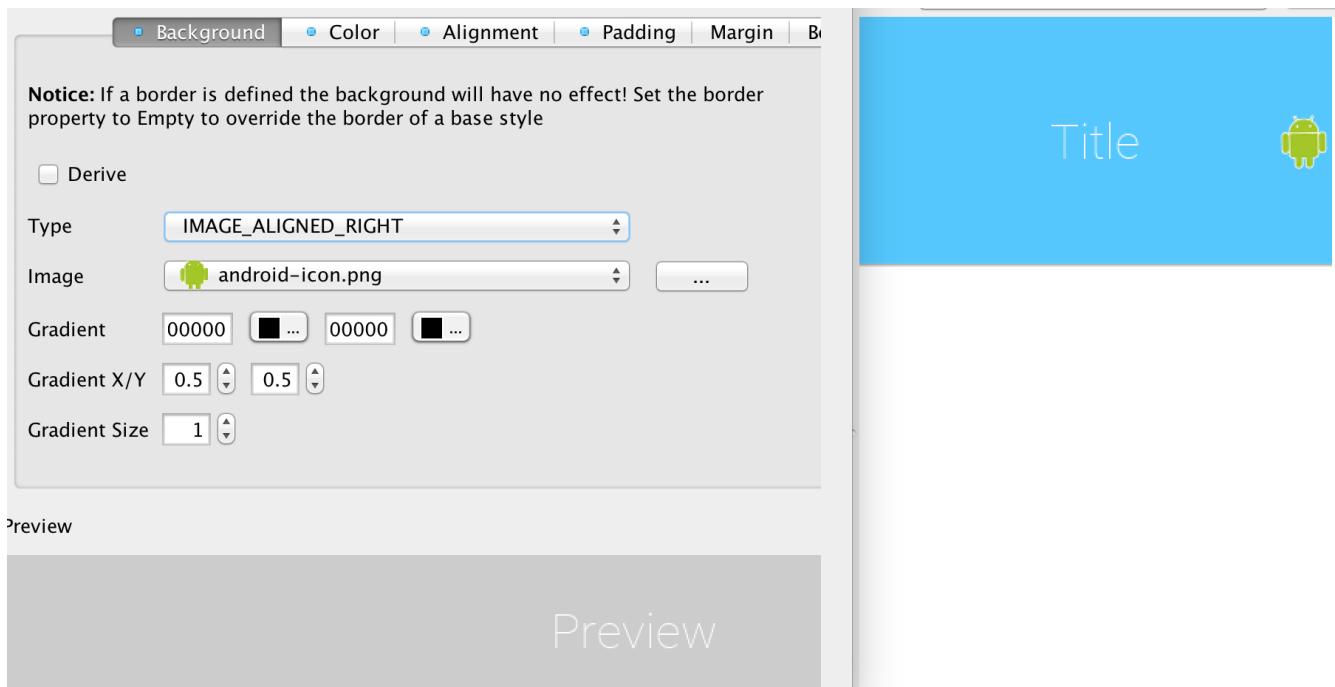


Figure 111. IMAGE_ALIGNED_RIGHT places the image centered at the right part of the component

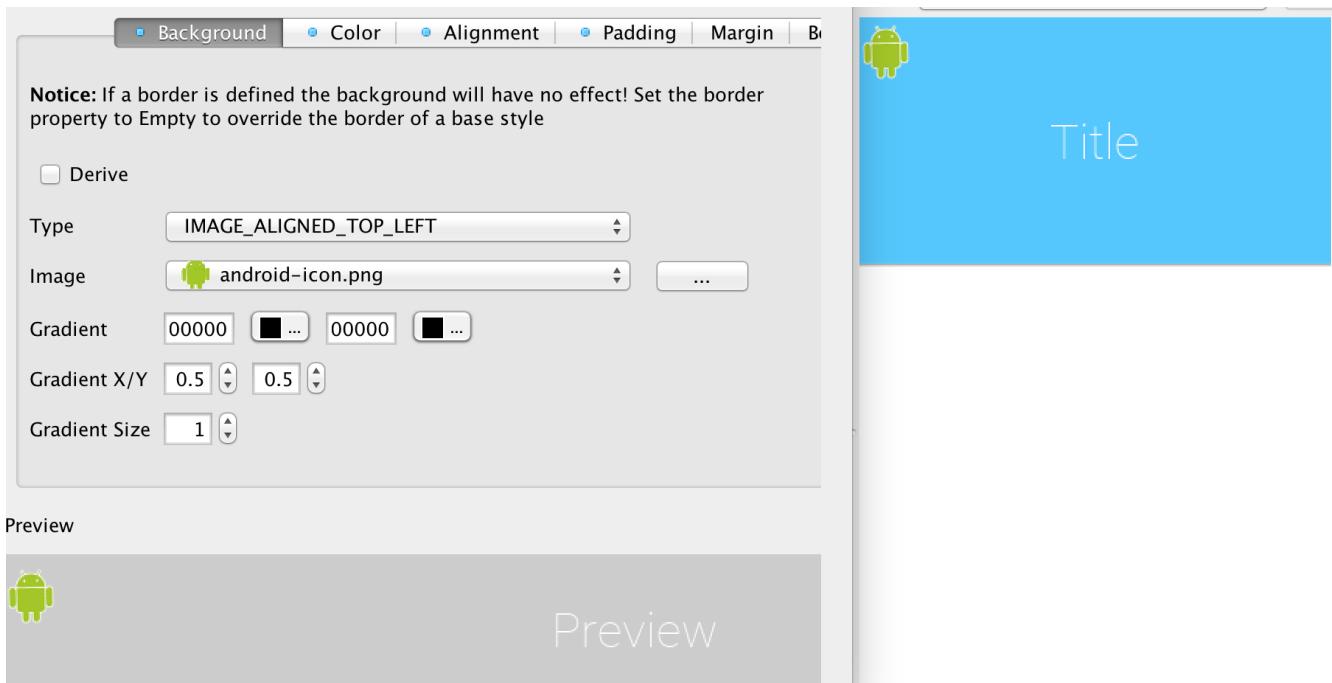


Figure 112. IMAGE_ALIGNED_TOP_LEFT places the image at the top left corner

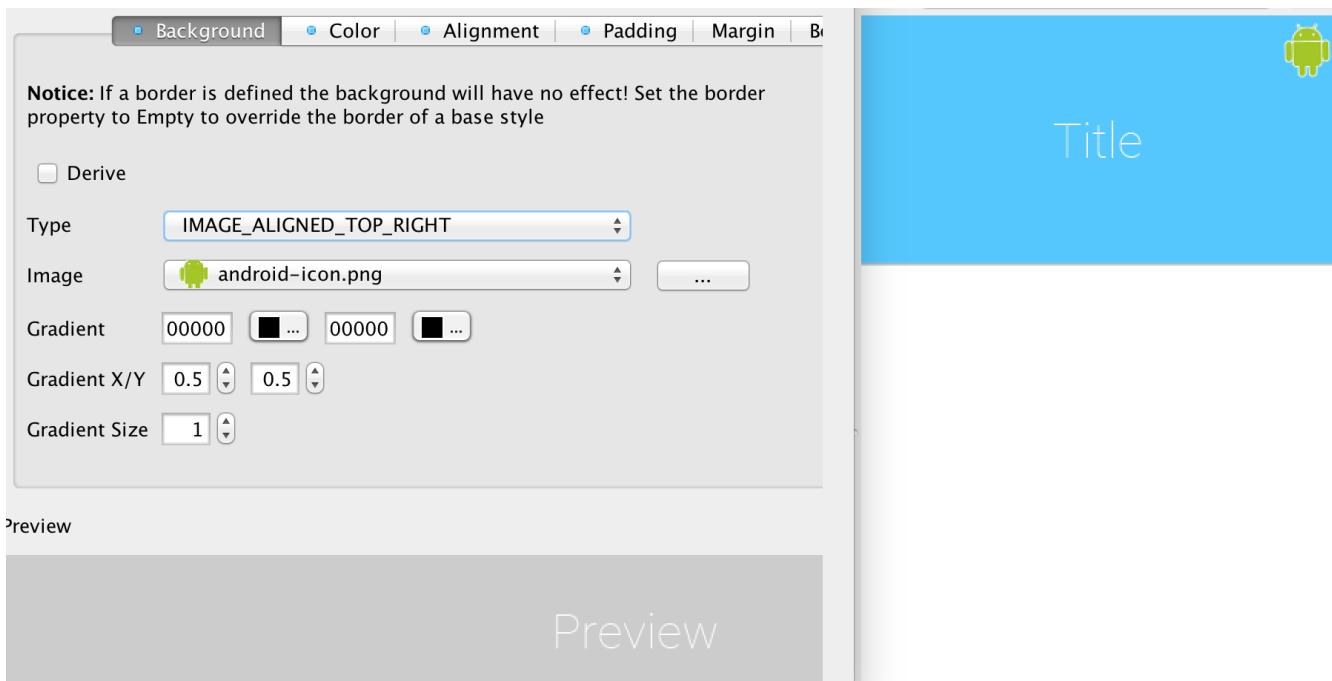


Figure 113. IMAGE_ALIGNED_TOP_RIGHT places the image at the top right corner

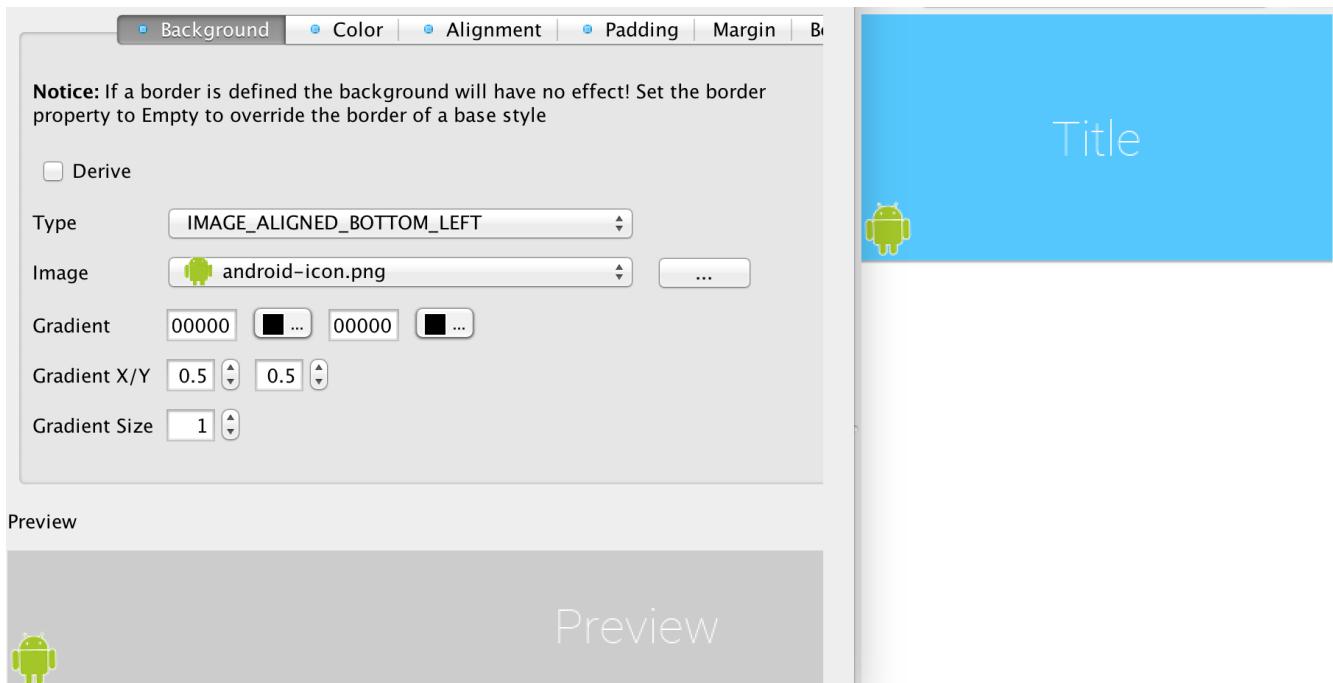


Figure 114. IMAGE_ALIGNED_BOTTOM_LEFT places the image at the bottom left corner

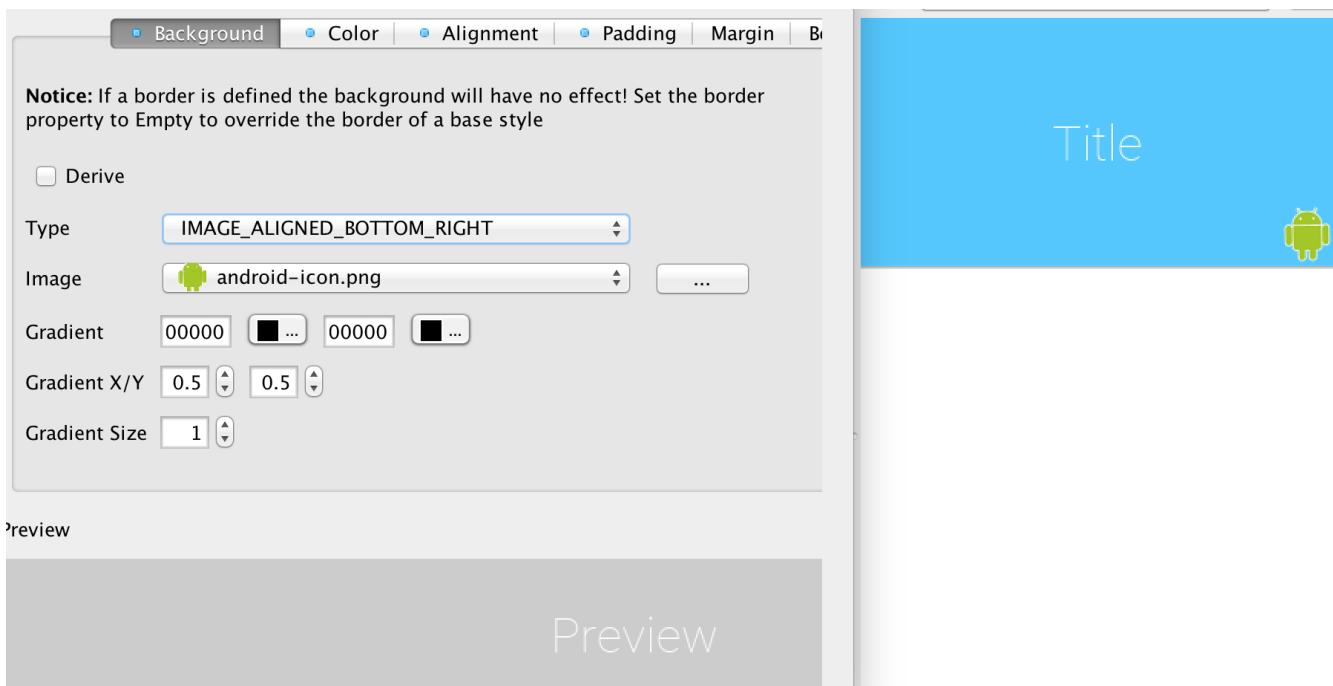


Figure 115. IMAGE_ALIGNED_BOTTOM_RIGHT places the image at the bottom right corner

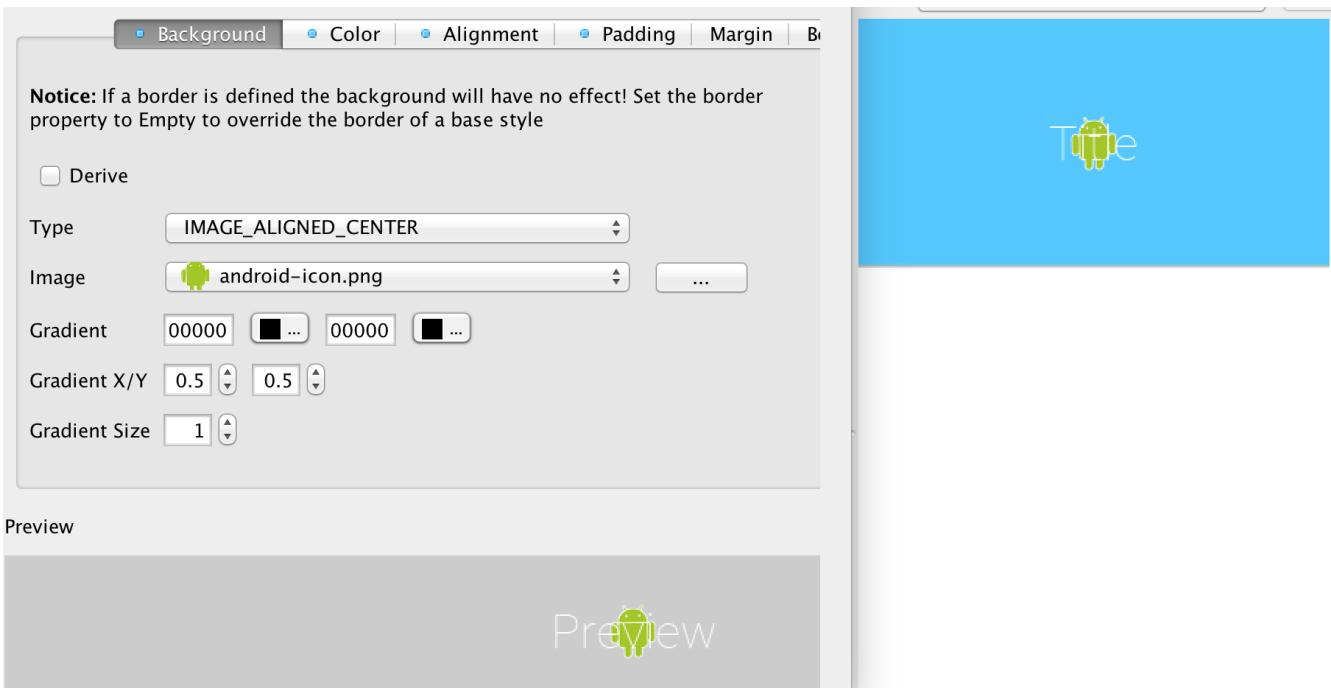


Figure 116. IMAGE_ALIGNED_CENTER places the image in the middle of the component

3.3.3. The Color Settings

The color settings are much simpler than the background behavior. As explained above the priority for color is at the bottom so if you have a border, image or gradient defined the background color settings will be ignored.

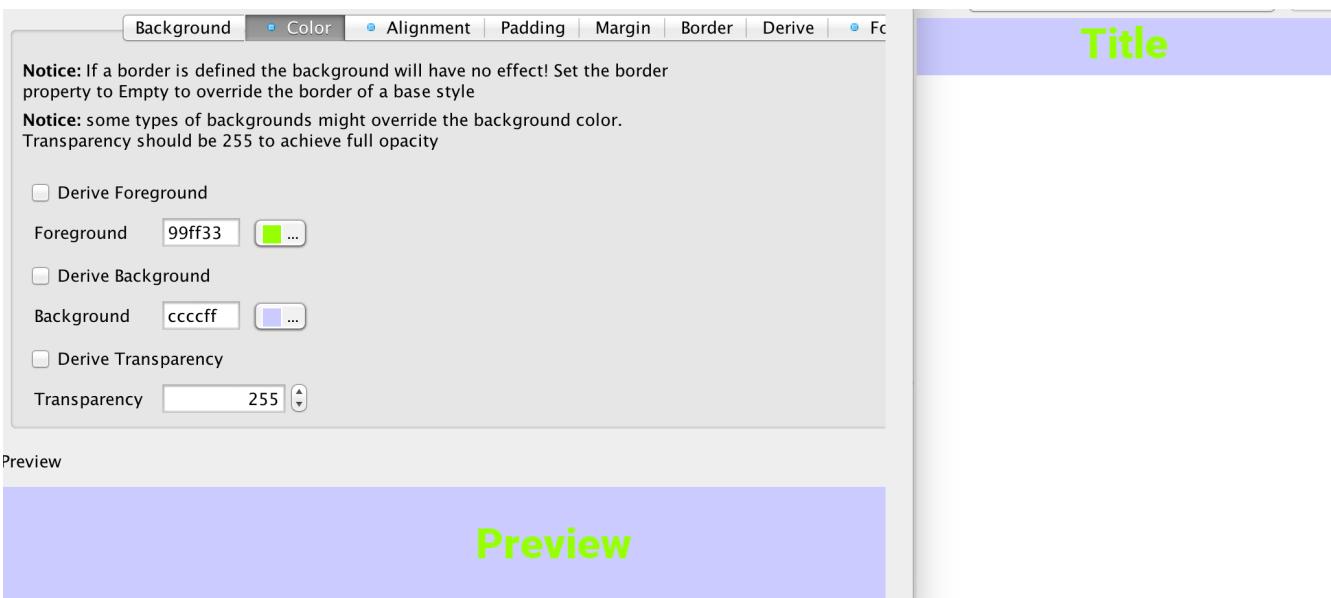


Figure 117. Add theme entry color settings

There are three color settings:

- Foreground color is the RRGGBB color that sets the style foreground color normally used to draw the text of the component. You can use the color picker button on the side to pick a color
- Background same as foreground only determines the background color of the component
- Transparency represents the opacity of the component background as a value between 0 (transparent) and 255 (opaque)



Setting the background will have no effect unless transparency is higher than 0. If you don't explicitly define this it might have a different value based on the native theme

3.3.4. Alignment

Not all component types support alignment and even when they do they don't support it for all elements. E.g. a [Label](https://www.codenameone.com/javadoc/com/codename1/ui/Label.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Label.html] and its subclasses support alignment but will only apply it to the text and not the icon.

Notice that [Container](https://www.codenameone.com/javadoc/com/codename1/ui/Container.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Container.html] doesn't support alignment. You should use the layout manager to tune component positioning.

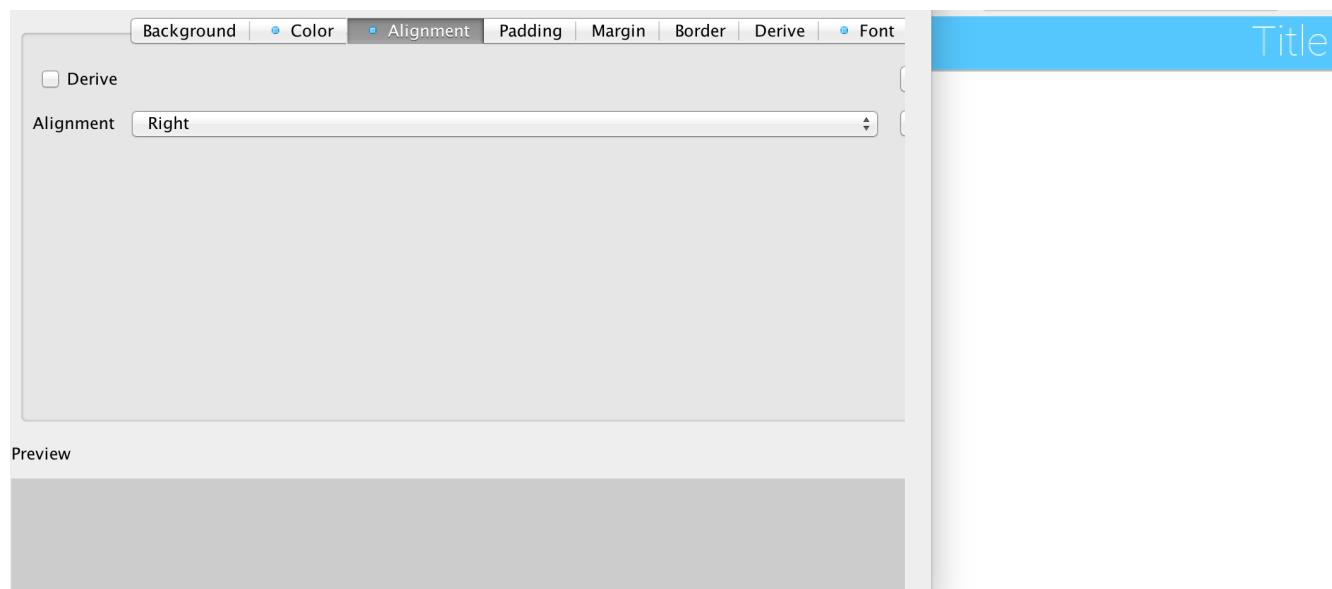


Figure 118. Alignment of the text within some component types



Aligning text components to anything other than the default alignment might be a problem if they are editable. The native editing capabilities might collide with the alignment behavior.



Bidi/RtL layout reverses the alignment value so left becomes right and visa versa

3.3.5. Padding and Margin

Padding and margin are concepts derived from the CSS box model. They are slightly different in Codename One, where the border spacing is part of the padding, but other than that they are pretty similar:

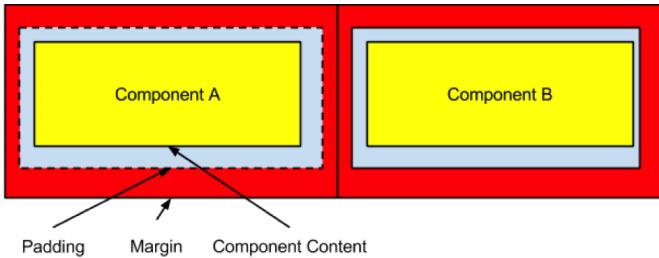


Figure 119. Padding and Margin/Box Model

In the diagram, we can see the component represented in yellow occupying its preferred size. The padding portion in gray effectively increases the components size. The margin is the space between components, it allows us to keep whitespace between multiple components. Margin is represented in red in the diagram.

The theme allows us to customize the padding/margin, and specify them for all 4 sides of a component. They can be specified in pixels, millimeters/dips, or screen percentage:

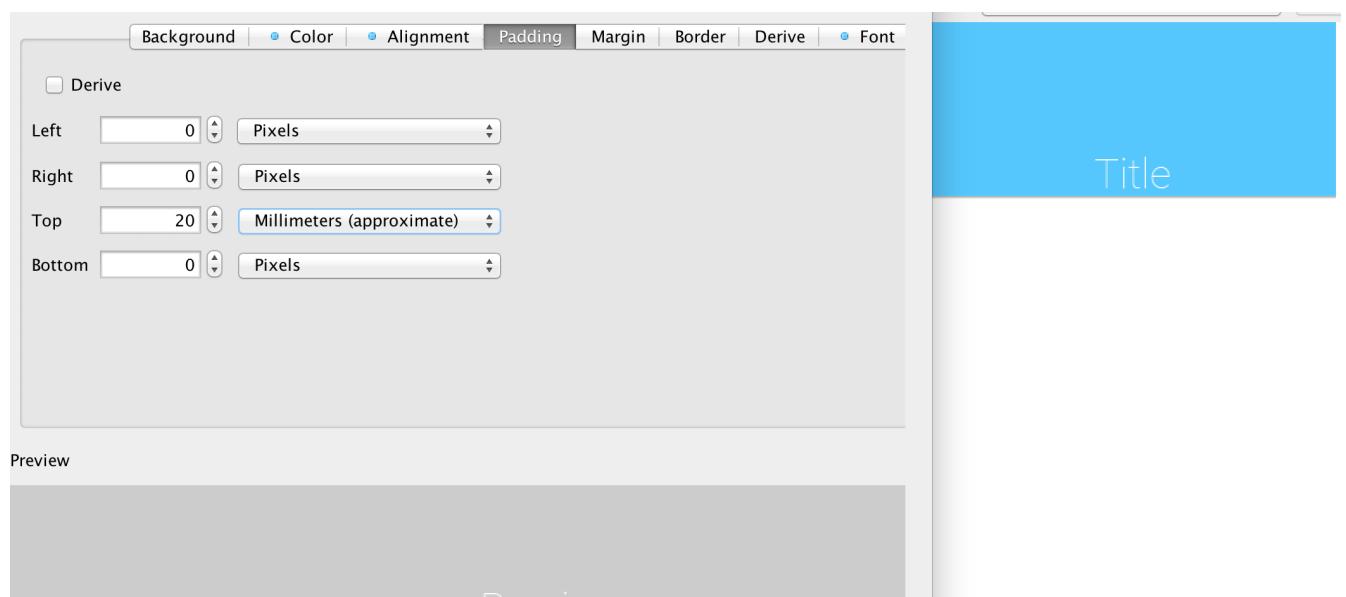


Figure 120. Padding tab



We recommend using millimeters for all spacing to make it look good for all device densities. Percentages make sense only in very extreme cases

3.3.6. Borders

Borders are a big subject in their own right, the UI for their creation is also a bit confusing:



Figure 121. Border entry in the theme

3.3.7. 9-Piece Image Border

A common border type is the 9-piece image border, to facilitate that border type we have a special **Image Border Wizard**.

A 9 piece image border is a common convention in UI theming that divides a border into 9 pieces 4 representing corners, 4 representing the sides and one representing the middle.



Android uses a common variation on the 9-piece border: 9-patch. The main difference between the 9-piece border and 9-patch is that 9-piece borders tile the sides/center whereas 9-patch scales them

9-piece image borders work better than background images for many use cases where the background needs to "grow/shrink" extensively and might need to change aspect ratio.

They don't work well in cases where the image is asymmetric on both axis. E.g. a radial gradient image. 9-piece images in general don't work very well with complex gradients.

The image border wizard simplifies the process of generating a 9-piece image border using a 3 stage process.

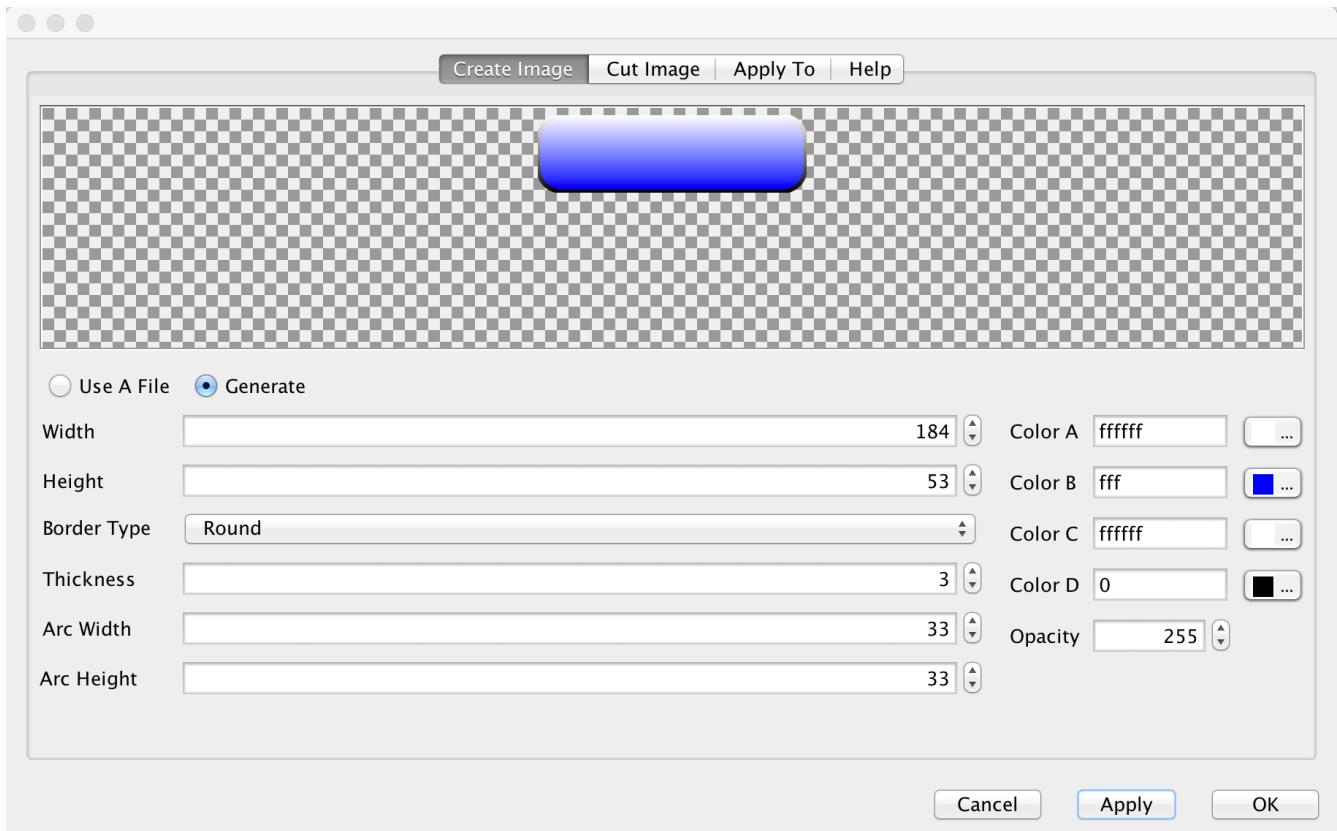


Figure 122. Stage 1: create or pick an image from an existing PNG file that we will convert to a 9-piece image



Use an image that's designed for a high DPI device

For your convenience you can create a rudimentary image with the create image stage but for a professional looking application you would usually want to use a design by a professional designer.

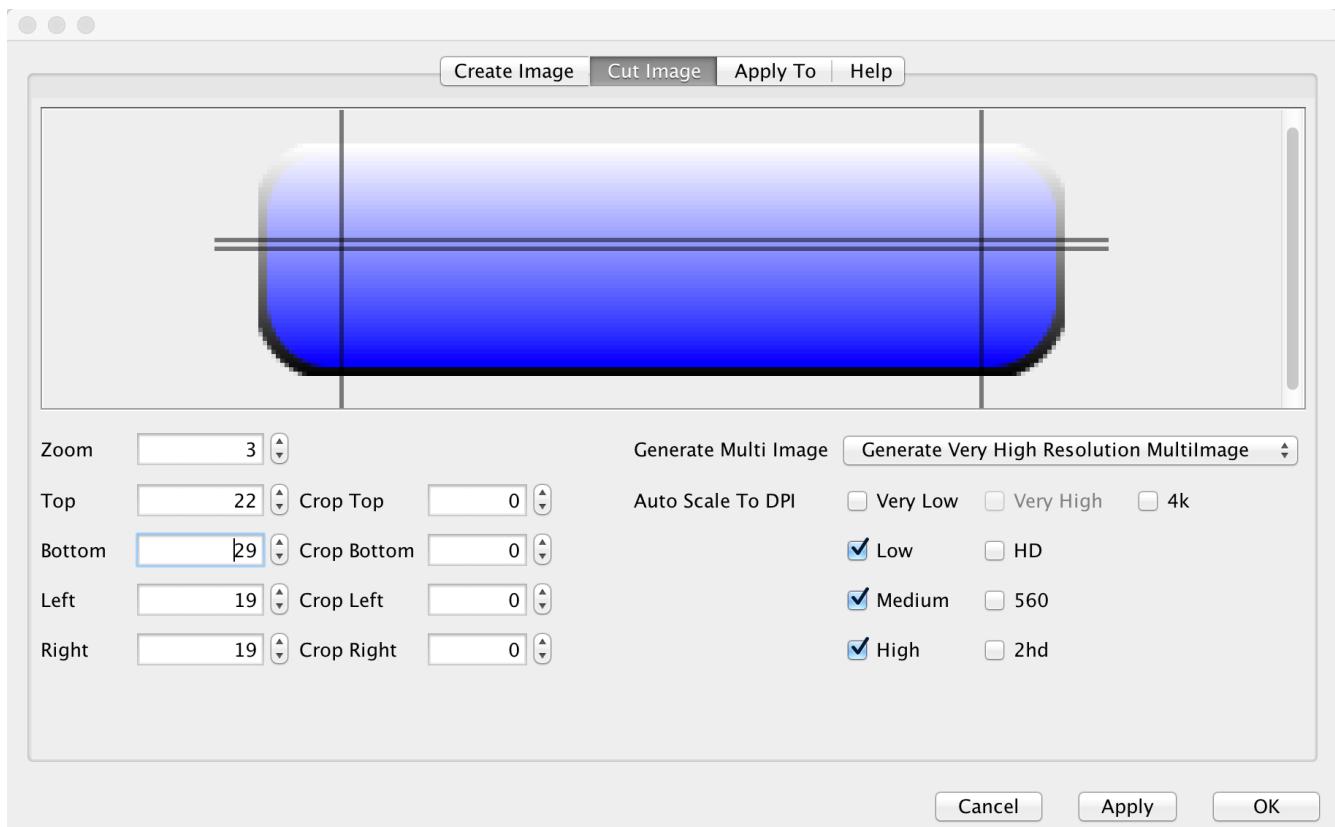


Figure 123. Stage 2: Cutting the image and adapting it to the DPI's

The second stage is probably the hardest and most important one in this wizard!

You can change the values of the top/bottom/left/right spinners to move the position of the guide lines that indicate the various 9 pieces. The image shows the correct cut for this image type with special attention to the following:

- The left/right position is high enough to fit in the rounded corners in their entirety. Notice that we didn't just leave 1 pixel as that performs badly, we want to leave as much space as possible!
- The top and bottom lines have exactly one pixel between them. This is to avoid breaking the gradient. E.g. if we set the lines further apart we will end up with this:



Figure 124. This is why it's important to keep the lines close when a gradient is involved, notice the tiling effect...



Figure 125. When the lines are close together the gradient effect grows more effectively

- The elements on the right hand side include the **Generate Multi Image** options. Here you can indicate the density of the source image you are using (e.g. if its for iPhone 5 class device pick Very High). You can then select in the checkboxes below the densities that should be generated automatically for you. This allows fine detail on the border to be maintained in the various high/low resolution devices.



We go into a lot of details about multi images in the advanced theming section.

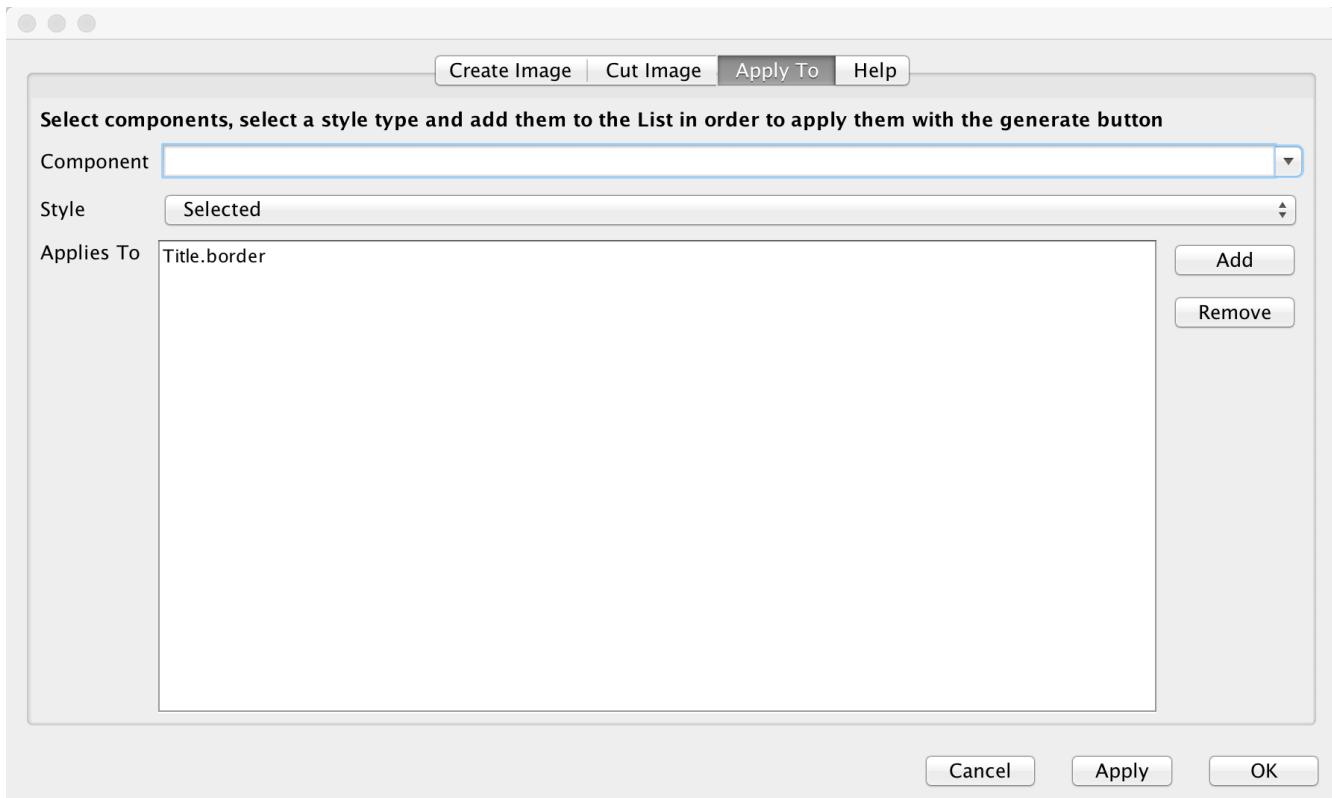


Figure 126. Stage 3: Styles to which the border is applied

The last page indicates the styles to which the wizard will apply the border. Under normal usage you don't really need to touch this as its properly filled out.

You can define the same border for multiple UIIDs from here though.

Border Minimum Size

A common oddity when using the image borders is the fact that even when padding is removed the component might take a larger size than the height of the text within it.

The reason for that is the border. Because of the way borders are implemented they can't be drawn to be smaller than the sum of their corners. E.g. the minimum height of a border would be the height of the bottom corner + the height of the top corner. The minimum width would be the width of the left + right corners.

This is coded into the common preferred size methods in Codename One and components generally don't shrink below the size of the image border even if padding is 0.

Customizing The 9-Piece Border

Normally we can just use the 9-piece border wizard but we can also customize the border by pressing the "..." button on the border section in the theme.

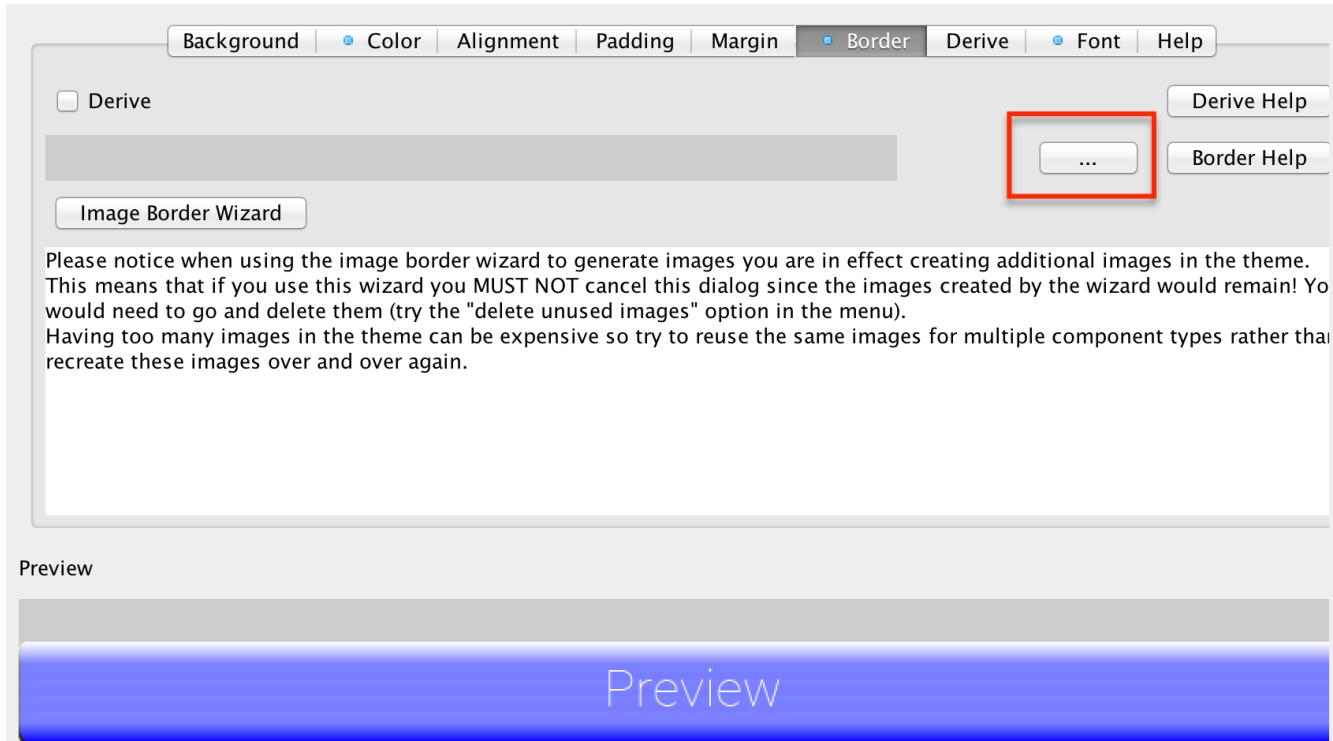


Figure 127. Press this to customize borders

The UI for the 9-piece border we created above looks like [this](#).

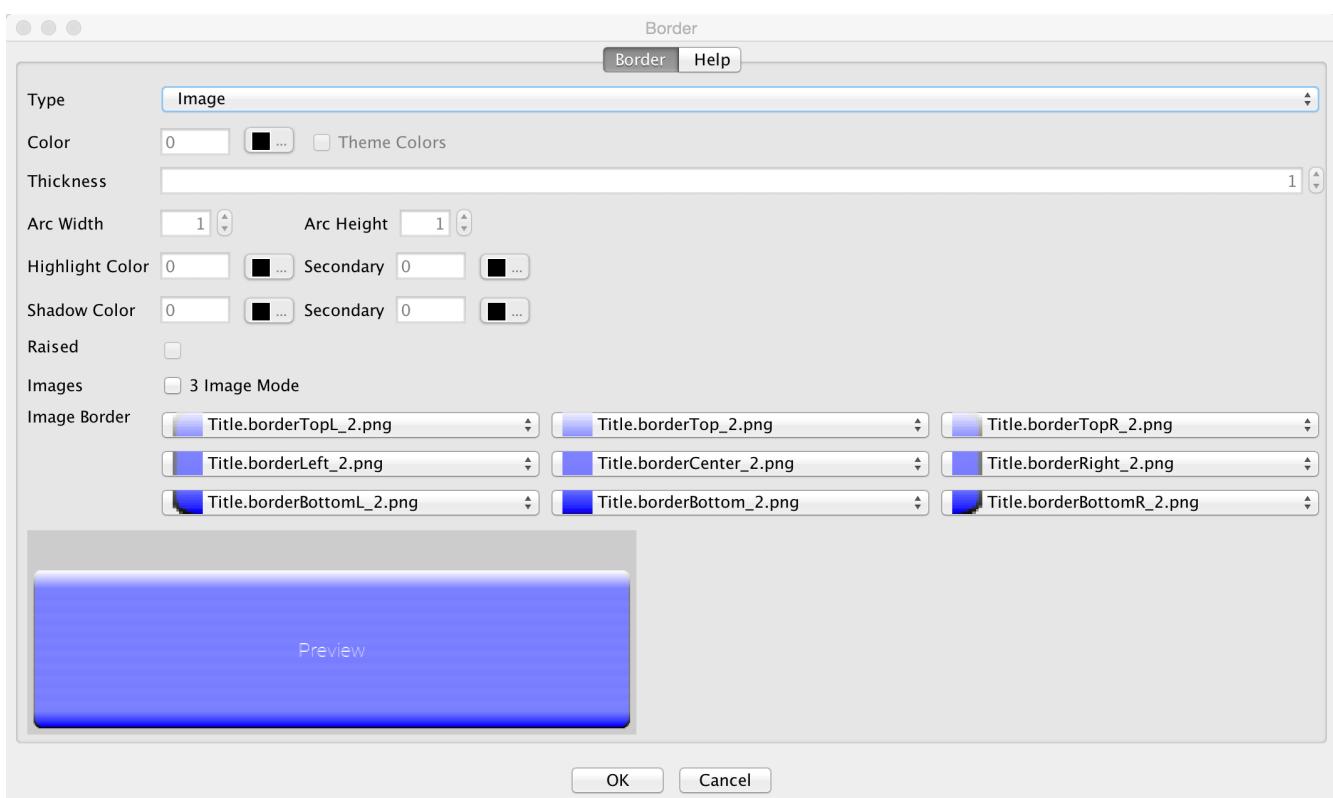


Figure 128. 9-piece border in edit mode

You can pick the image represented by every section in the border from the combo boxes. They are organized in the same way the border is with the 9-pieces placed in the same position they would occupy when the border is rendered.



Notice that the other elements in the UI are disabled when the image border type is selected.

3 Image Mode

The 9-piece border has a (rarely used) special case: 3 image mode. In this mode a developer can specify the top left corner, the top image and the center image to produce a 9 piece border. The corner and top piece are then rotated dynamically to produce a standard 9-piece border on the device.

This is useful for reducing application code size but isn't used often as it requires a more symetric UI.



Don't confuse the 3-image mode for the 9-piece border with the horizontal/vertical image border below

3.3.8. Horizontal/Vertical Image Border

The 9-piece border is the workhorse of borders in Codename One, however there are some edge cases of UI elements that should grow on one axis and not on another. A perfect example of this is the iOS 6 style back button. If we tried to cut it into a 9-piece border the arrow effect would be broken.

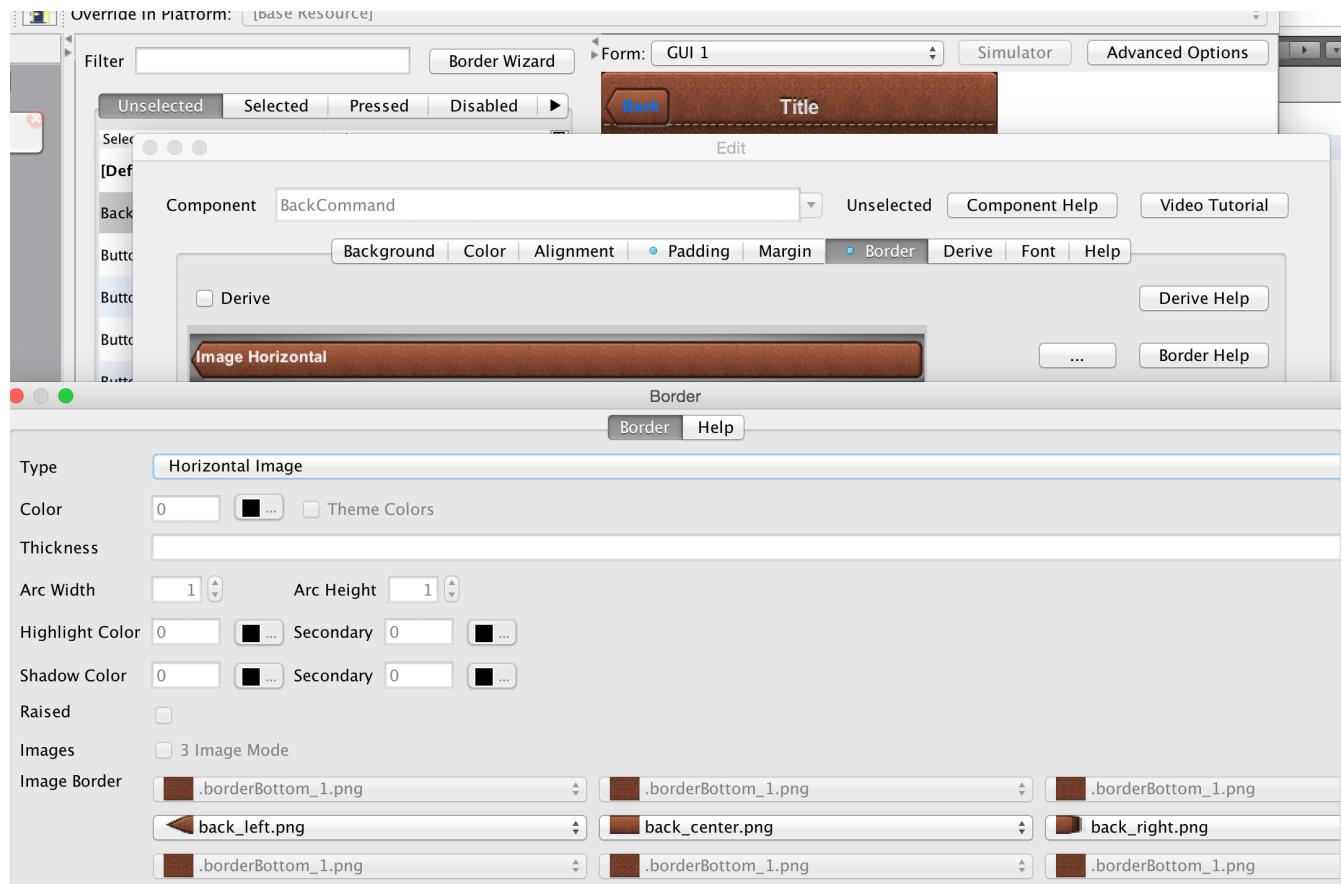


Figure 129. Horizontal image border is commonly used for UI's that can't grow vertically e.g. the iOS 6 style back button

The horizontal and vertical image borders accept 3 images of their respective AXIS and build the border by placing one image on each side and tiling the center image between them. E.g. A horizontal border will never grow vertically.



In RTL/Bidi ^[1] modes the borders flip automatically to show the reverse direction. An iOS style back button will point to the right in such languages.

3.3.9. Empty Border

Empty borders enforce the removal of a border. This is important if you would like to block a base style from having a border.

E.g. Buttons have borders by default. If you would like to create a [Button](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Button.html>] that is strictly of solid color you could just define the border to be empty and then use the solid color as you see fit.



There is a null border which is often confused with an empty border. You should use empty border and not null border

3.3.10. Round Border

Circles and completely round border sides are problematic for multi-resolutions. You need to draw them dynamically and can't use image borders which can't be tiled/cut to fit round designs (due to physical constraints of the round shape).

We designed the [RoundBorder](#) to enable two distinct types of borders:

- Circular borders - e.g. Android floating action
- Rectangles with round (not rounded) sides

Round Border is a bit confusing since we already support a **rounded** border type. The rounded border type is a rectangle with rounded corners whereas the round border has completely round sides or appears as a circle.

To make matters worse the round border has a ridiculous number of features/configurations that would have made the already cluttered UI darn near impossible to navigate. To simplify this we split the UI into 3 tabs for standard borders, image borders and round border.

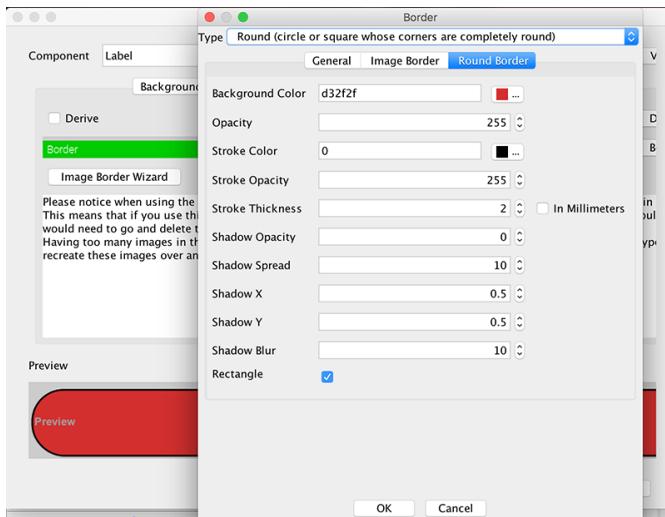


Figure 130. Round Border

3.3.11. Rounded Rectangle Border

The **RoundRectBorder** was developed based on the **RoundBorder** and has similar features. It produces a rounded rectangle UI.



Don't confuse the Rounded Rectangle border with the deprecated **Rounded** border...

It's a pretty simple border type akin to the **RoundBorder**.

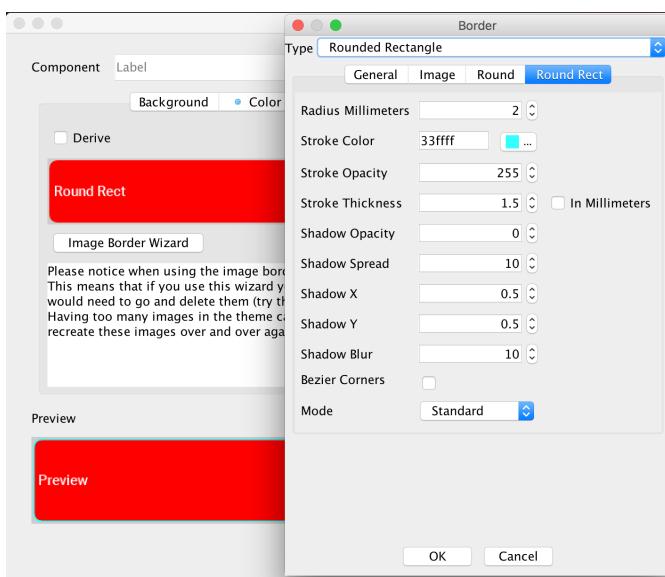


Figure 131. Rounded Rectangle Border

3.3.12. Bevel/Etched Borders

We generally recommend avoiding bevel/etched border types as they aren't as efficient and look a bit dated in todays applications. We cover them here mostly for completeness.



Figure 132. Bevel border

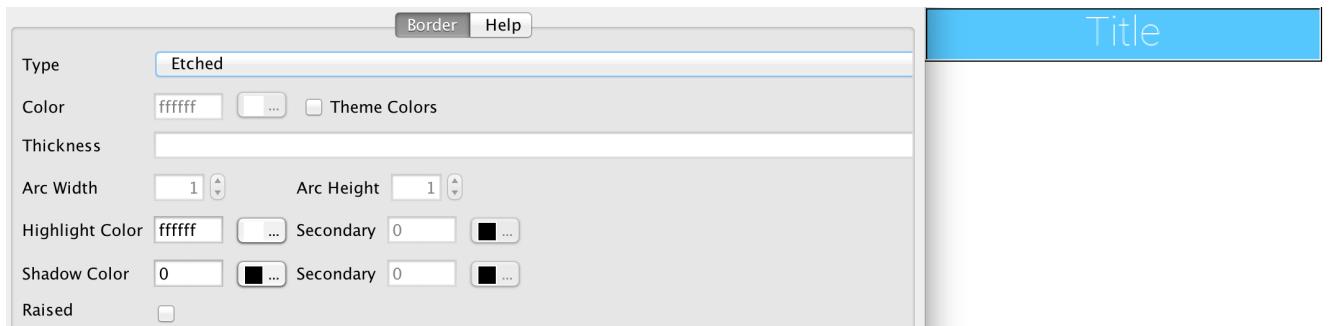


Figure 133. Etched border

3.3.13. Derive

Derive allows us to inherit the behavior of a UIID and extend it with some customization.

E.g. Lets say we created a component that's supposed to look like a title, we could do something like:

```
cmp.setUIID("Title");
```

But title might sometimes be aligned to the left (based on theme) and we always want our component to be center aligned. However, we don't want that to affect the actual titles in the app...

To solve this we can define a **MyTitle** UIID and derive the **Title** UIID. Then just customize that one attribute.

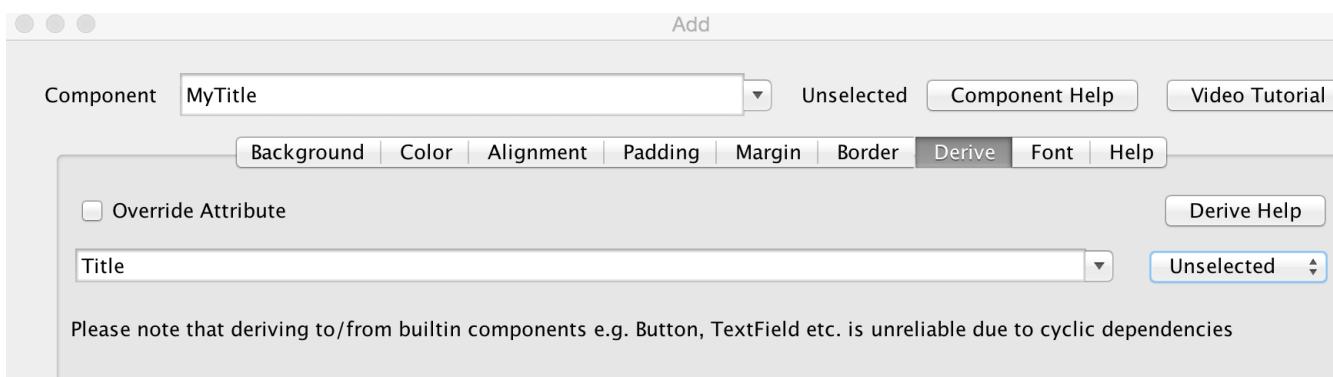


Figure 134. Derive title

Issues With Derive

Style inheritance is a problematic topic in every tool that supports such behavior. Codename One styles start from a global default then have a system default applied and on top of that have the native OS default applied to them.

At that point a developer can define the style after all of the user settings are in place. Normally this works reasonably well, but there are some edge cases where inheriting a style can fail.

When you override an existing style such as `Button` and choose to derive from `Button` in a different selection mode or even a different component altogether such as `Label` you might trigger a recursion effect where a theme setting in the base theme depends on something in a base triggering an infinite loop.

To avoid this always inherit only from UIID's you defined e.g. `MyButton`.

3.3.14. Fonts

Codename One currently supports 3 font types:

- **System fonts**—these are very simplistic builtin fonts. They work on all platforms and come in one of 3 sizes. However, they are ubiquitous and work in every platform in all languages.
- **TTF files**—you can just place a TTF file in the src directory of the project and it will appear in the **True Type** combo box.
- **Native fonts**—these aren't supported on all platforms but generally they allow you to use a set of platform native good looking fonts. E.g. on Android the devices Roboto font will be used and on iOS San Francisco or Helvetica Neue will be used. **This is the recommended font type we suggest for most use cases!**



If you use a TTF file **MAKE SURE** not to delete the file when there **MIGHT** be a reference to it. This can cause hard to track down issues!



Notice that a TTF file must have the ".ttf" extension, otherwise the build server won't be able to recognize the file as a font and set it up accordingly (devices need fonts to be defined in very specific ways). Once you do that, you can use the font from code or from the theme

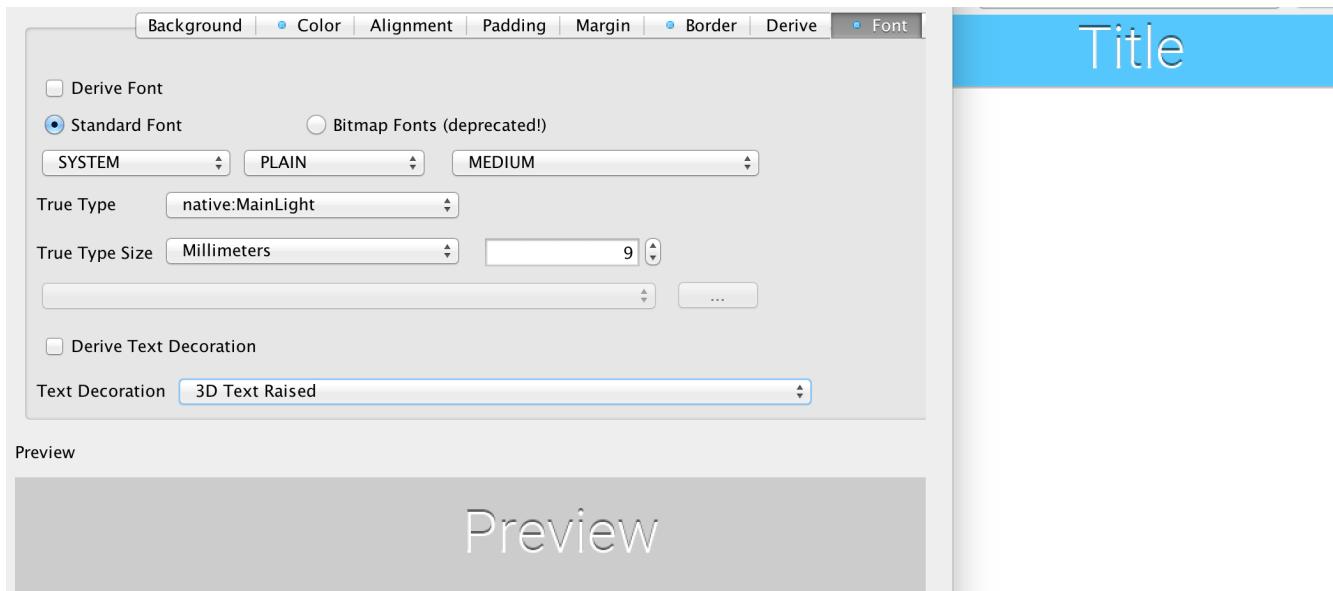


Figure 135. Font Theme Entry



System fonts are always defined even if you use a TTF or native font. If the native font/TTF is unavailable in a specific platform the system font will be used instead.

You can size native/TTF fonts either via pixels, millimeters or based on the size of the equivalent system fonts:

1. **System font size** - the truetype font will have the same size as a small, medium or large system font. This allows the developer to size the font based on the device DPI
2. **Millimeter size** - allows sizing the font in a more DPI aware size
3. **Pixels** - useful for some unique cases, but highly problematic in multi-DPI scenarios



You should notice that font sizing is very inconsistent between platforms we recommend using millimeters for sizing

You can load a truetype font from code using:

```
if(Font.isTrueTypeFileSupported()) {
    Font myFont = Font.createTrueTypeFont(fontName, fontFileName);
    myFont = myFont.derive(sizeInPixels, Font.STYLE_PLAIN);
    // do something with the font
}
```

Notice that, in code, only pixel sizes are supported, so it's up to you to decide how to convert that. We recommend using millimeters with the `convertToPixels` method. You also need to derive the font with the proper size, unless you want a 0 sized font which isn't very useful.

The font name is the difficult bit, iOS requires the name of the font in order to load the font. This font name doesn't always correlate to the file name making this task rather "tricky". The actual font name is sometimes viewable within a font viewer. It isn't always intuitive, so be sure to test that on the device to make sure you got it right.



due to copyright restrictions we cannot distribute Helvetica and thus can't simulate it. In the simulator you will see Roboto and not the device font unless you are running on a Mac

The code below demonstrates all the major fonts available in Codename One with the handlee ttf file posing as a standin for arbitrary TTF:

```
private Label createForFont(Font fnt, String s) {
    Label l = new Label(s);
    l.getUnselectedStyle().setFont(fnt);
    return l;
}

public void showForm() {
    GridLayout gr = new GridLayout(5);
    gr.setAutoFit(true);
    Form hi = new Form("Fonts", gr);

    int fontSize = Display.getInstance().convertToPixels(3);

    // requires Handlee-Regular.ttf in the src folder root!
    Font ttfFont = Font.createTrueTypeFont("Handlee", "Handlee-Regular.ttf").
        derive(fontSize, Font.STYLE_PLAIN);

    Font smallPlainSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_SMALL);
    Font mediumPlainSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
    Font largePlainSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_LARGE);
    Font smallBoldSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_SMALL);
    Font mediumBoldSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM);
    Font largeBoldSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_LARGE);
    Font smallItalicSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_ITALIC, Font.SIZE_SMALL);
    Font mediumItalicSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_ITALIC, Font.SIZE_MEDIUM);
    Font largeItalicSystemFont = Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_ITALIC, Font.SIZE_LARGE);

    Font smallPlainMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN, Font.SIZE_SMALL);
    Font mediumPlainMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
    Font largePlainMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN, Font.SIZE_LARGE);
    Font smallBoldMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD, Font.SIZE_SMALL);
    Font mediumBoldMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD, Font.SIZE_MEDIUM);
    Font largeBoldMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD, Font.SIZE_LARGE);
    Font smallItalicMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_ITALIC, Font.SIZE_SMALL);
    Font mediumItalicMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_ITALIC, Font.SIZE_MEDIUM);
    Font largeItalicMonospaceFont = Font.createSystemFont(Font.FACE_MONOSPACE, Font.STYLE_ITALIC, Font.SIZE_LARGE);

    Font smallPlainProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_PLAIN, Font.SIZE_SMALL);
    Font mediumPlainProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
    Font largePlainProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_PLAIN, Font.SIZE_LARGE);
    Font smallBoldProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD, Font.SIZE_SMALL);
    Font mediumBoldProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD, Font.SIZE_MEDIUM);
    Font largeBoldProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD, Font.SIZE_LARGE);
    Font smallItalicProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_ITALIC, Font.SIZE_SMALL);
    Font mediumItalicProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_ITALIC, Font.SIZE_MEDIUM);
    Font largeItalicProportionalFont = Font.createSystemFont(Font.FACE_PROPORTIONAL, Font.STYLE_ITALIC, Font.SIZE_LARGE);

    String[] nativeFontTypes = {
        "native:MainThin", "native:MainLight",
        "native:MainRegular", "native:MainBold",
        "native:MainBlack", "native:ItalicThin",
        "native:ItalicLight", "native:ItalicRegular",
        "native:ItalicBold", "native:ItalicBlack"};
    }

    for(String s : nativeFontTypes) {
        Font tt = Font.createTrueTypeFont(s, s).derive(fontSize, Font.STYLE_PLAIN);
```

```

        hi.add(createForFont(tt, s));
    }

    hi.add(createForFont(ttfFont, "Handlee TTF Font"));
    add(createForFont(smallPlainSystemFont, "smallPlainSystemFont"));
    add(createForFont(mediumPlainSystemFont, "mediumPlainSystemFont"));
    add(createForFont(largePlainSystemFont, "largePlainSystemFont"));
    add(createForFont(smallBoldSystemFont, "smallBoldSystemFont"));
    add(createForFont(mediumBoldSystemFont, "mediumBoldSystemFont"));
    add(createForFont(largeBoldSystemFont, "largeBoldSystemFont"));
    add(createForFont(smallPlainSystemFont, "smallItalicSystemFont"));
    add(createForFont(mediumItalicSystemFont, "mediumItalicSystemFont"));
    add(createForFont(largeItalicSystemFont, "largeItalicSystemFont"));

    add(createForFont(smallPlainMonospaceFont, "smallPlainMonospaceFont"));
    add(createForFont(mediumPlainMonospaceFont, "mediumPlainMonospaceFont"));
    add(createForFont(largePlainMonospaceFont, "largePlainMonospaceFont"));
    add(createForFont(smallBoldMonospaceFont, "smallBoldMonospaceFont"));
    add(createForFont(mediumBoldMonospaceFont, "mediumBoldMonospaceFont"));
    add(createForFont(largeBoldMonospaceFont, "largeBoldMonospaceFont"));
    add(createForFont(smallItalicMonospaceFont, "smallItalicMonospaceFont"));
    add(createForFont(mediumItalicMonospaceFont, "mediumItalicMonospaceFont"));
    add(createForFont(largeItalicMonospaceFont, "largeItalicMonospaceFont"));

    add(createForFont(smallPlainProportionalFont, "smallPlainProportionalFont"));
    add(createForFont(mediumPlainProportionalFont, "mediumPlainProportionalFont"));
    add(createForFont(largePlainProportionalFont, "largePlainProportionalFont"));
    add(createForFont(smallBoldProportionalFont, "smallBoldProportionalFont"));
    add(createForFont(mediumBoldProportionalFont, "mediumBoldProportionalFont"));
    add(createForFont(largeBoldProportionalFont, "largeBoldProportionalFont"));
    add(createForFont(smallItalicProportionalFont, "smallItalicProportionalFont"));
    add(createForFont(mediumItalicProportionalFont, "mediumItalicProportionalFont"));
    add(createForFont(largeItalicProportionalFont, "largeItalicProportionalFont"));

    hi.show();
}

```

Fonts		
native:MainThin	native:MainLight	native:MainRegular
native:MainBold	native:MainBlack	native:ItalicThin
native:ItalicLight	native:ItalicRegular	native:ItalicBold
native:ItalicBlack	Handlee TTF Font	smallPlainSystemFont
mediumPlainSystemFont	largePlainSystemFont	smallBoldSystemFont
mediumBoldSystemFont	largeBoldSystemFont	smallItalicSystemFont
mediumItalicSystemFont	largeItalicSystemFont	smallPlainMonospaceFont
mediumPlainMonospaceFont	largePlainMonospaceFont	smallBoldMonospaceFont
mediumBoldMonospaceFont	largeBoldMonospaceFont	smallItalicMonospaceFont
mediumItalicMonospaceFont	largeItalicMonospaceFont	smallPlainProportionalFont
mediumPlainProportionalFont	largePlainProportionalFont	smallBoldProportionalFont
mediumBoldProportionalFont	largeBoldProportionalFont	smallItalicProportionalFont
mediumItalicProportionalFont	largeItalicProportionalFont	

Figure 136. The fonts running on the ipad simulator on a Mac, notice that this will look different on a PC

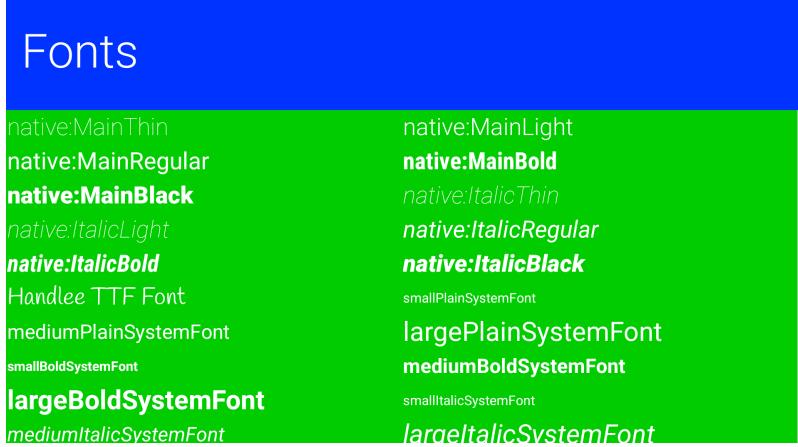


Figure 137. The same demo running on a OnePlus One device with Android 5.1

Font Effects

You can define an effect to be applied to a specific font, specifically:

- Underline
- Strike thru
- 3d text raised/lowered
- 3d shadow north

The "3d" effects effectively just draw the text twice, with a slight offset and two different colors to create a "3d" feel.

All of the effects are relatively simple and performant.

[1] Languages that are written from right to left such as Hebrew, Arabic etc.

4. Advanced Theming

Before we go into CSS there are a few advanced theme concepts. Notice this still applies to CSS as features such as theme constants are used there as well...

4.1. Working With UIID's

UIID's (User Interface IDentifier) are unique qualifiers given to UI components that associate a set of theme definitions with a specific set of components. E.g. we can associate the **Button** UIID with a component and then define the look for the **Button** in the theme.

One of the biggest advantages with UIID's is the ability to change the UIID of a component. E.g. to create a multiline label, one can use something like:

```
TextArea t = ...;  
t.setUIID("Label");  
t.setEditable(false);
```



This is pretty much how components such as **SpanLabel** [<https://www.codenameone.com/javadoc/com/codename1/components/SpanLabel.html>] are implemented internally

UIID's can be customized via the GUI builder and allow for powerful customization of individual components.



The class name of the component is commonly the same as the UIID, but they are in essence separate entities

4.2. Theme Layering

There are two use cases in which you would want to use layering:

- You want a **slightly** different theme in one platform
- You want the ability to customize your theme for a specific use case, e.g. let a user select larger fonts

This is actually pretty easy to do and doesn't require re-doing the entire theme. You can do something very similar to the cascading effect of CSS where a theme is applied "on top" of another theme. To do that just add a new theme using the **Add Theme** button.



Make sure to remove the **includeNativeBool** constant in the new theme!

In the new theme define the changes e.g. if you just want a larger default font define only that property for all the relevant UIID's and ignore all other properties!

For a non-gui builder app the theme loading looks like this by default:

```
theme = UIManager.initFirstTheme("/theme");
```

You should fix it to look like this:

```
theme = UIManager.initNamedTheme("/theme", "Theme");
```



This assumes the name of your main theme is "Theme" (not the layer theme you just added).

The original code relies on the theme being in the 0 position in the theme name array which might not be the case!

When you want to add the theme layer use:

```
UIManager.getInstance().addThemeProps(theme.getTheme("NameOfLayerTheme"));
```

The `addThemeProps` call will layer the secondary theme on top of the primary "Theme" and keep the original UIID's defined in the "Theme" intact.

If you apply theme changes to a running application you can use `Form's `refreshTheme()` to update the UI instantly and provide visual feedback for the theme changes.

4.3. Override Resources In Platform

When we want to adapt the look of an application to different OS conventions one of the common requirements is to use different icons. Sometimes we want to change behavior based on device type e.g. have a different UI structure for a Tablet.

Codename One allows you to override a resource for a specific platform when doing this you can redefine a resource differently for that specific platform and also add platform specific resources.

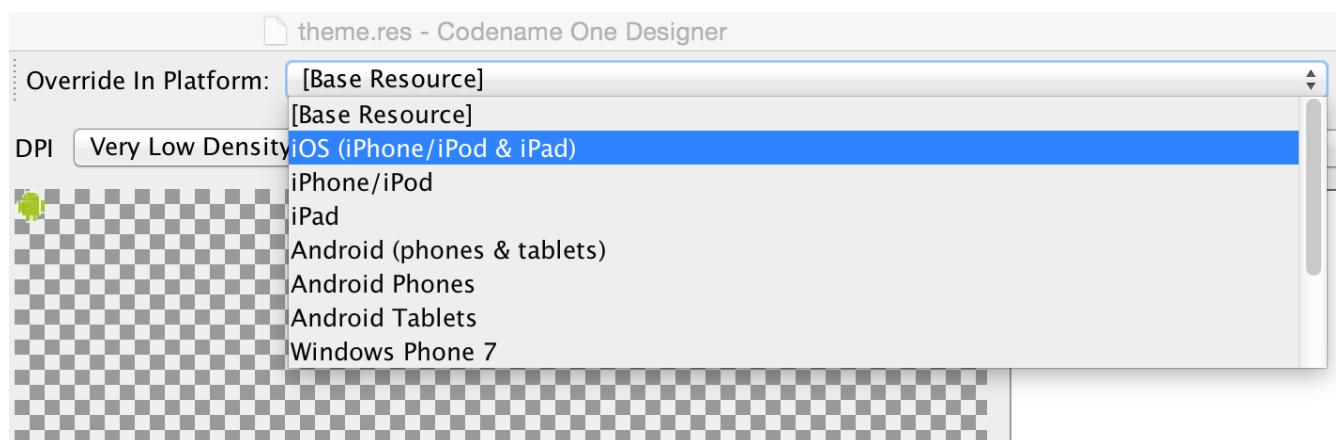


Figure 138. Override resources for specific platform

Overridden resources take precedence over embedded resources thus allowing us to change the look or even behavior (when overriding a GUI builder element) for a specific platform/OS.



Overriding the theme is dangerous as a theme has external dependencies (e.g. image borders). The solution is to use [theme layering](#) and override the layer!

To override select the platform where overriding is applicable

Figure 139. Override for platform, allows us to override the checked resources and replace them with another resource

You can then click the green checkbox to define that this resource is specific to this platform. All resources added when the platform is selected will only apply to the selected platform. If you change your mind and are no longer interested in a particular override just delete it in the override mode and it will no longer be overridden.

4.4. Theme Constants

The Codename One Designer has a tab for creating constants which can be used to add global values of various types and behavior hints to Codename One and its components. Constants are always strings. There are some conventions that allow the UI to adapt to input types e.g. if a constant ends with the word Bool it is treated as a boolean (true/false) value. Such a value will display as a checkbox. Similarly an **Int** suffix will display a numeric picker and an **Image** suffix will show a combo box to pick an image.



The combo box in the designer for adding a theme constant is editable, you can just type in any value you want!

To use a constant one can use the [UIManager](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html>]'s methods to get the appropriate constant type specifically:

- `getThemeConstant`
- `isThemeConstant`
- `getThemeImageConstant`

Internally, Codename One has several built in constants and the list is constantly growing. As we add features to Codename One, we try to keep this list up to date but the very nature of theme constants is "adhoc" and some might not make it here.

Table 3. Theme Constants

Constant	Description/Argument
alwaysTensileBool	Enables tensile drag even when there is no scrolling in the container (only for scrollable containers)
backGestureThresholdInt	The threshold for the back gesture in the SwipeBackSupport [https://www.codenameone.com/javadoc/com/codename1/ui/util/SwipeBackSupport.html] class, defaults to 5

Constant	Description/Argument
backUsesTitleBool	Indicates to the GUI builder that the back command should use the title of the previous form and not just the word "Back"
buttonRippleBool	<code>true</code> to activate the material design ripple effect on the buttons. This effect draws a growing circle from the point of touch onwards. This is <code>false</code> by default except for Android where it defaults to <code>true</code> . This is equivalent to <code>setButtonRippleEffectDefault(bool)</code> in the <code>Button</code> class
capsButtonTextBool	<code>true</code> to activate the caps text mode in the buttons. When activated <code>setText</code> on <code>Button</code> and all the constructors will invoke <code>upcase()</code> on all the strings effectively making the application buttons use uppercase exclusively. This is <code>false</code> by default except for Android where it defaults to <code>true</code> . It's equivalent to <code>Button.setCapsTextDefault(boolean)</code> and can be tuned to an individual <code>Component</code> via <code>Component.setRippleEffect(boolean)</code>
capsButtonUiids	A list of the UIID's that should be capitalized by default (in supported platforms) other than the <code>Button</code> and <code>RaisedButton</code> which are already capitalized. This list can be separated by spaces or commas e.g. <code>capsButtonUiids=UpcaseButton,OtherCustomButton</code>
defaultCommandImage	Image to give a command with no icon
dialogButtonCommandsBool	Place commands in the dialogs as buttons
dialogBlurRadiusInt	Sets the default Gaussian blur radius for the background of the dialogs. The default value is -1 indicating no blur
dialogPosition	Place the dialog in an arbitrary border layout position (e.g. North, South, Center, etc.)
centeredPopupBool	Popup of the combo box will appear in the center of the screen
changeTabOnFocusBool	Useful for feature phones, allows changing the tab when the focus changes immediately, without pressing a key
checkBoxCheckDisImage	CheckBox image to use instead of Codename One drawing it on its own
checkBoxCheckedImage	CheckBox image to use instead of Codename One drawing it on its own

Constant	Description/Argument
checkBoxOppositeSideBool	Indicates the check box should be drawn on the opposite side to the text and not next to the text
checkBoxUncheckDisImage	CheckBox image to use instead of Codename One drawing it on its own
checkBoxUncheckedImage	CheckBox image to use instead of Codename One drawing it on its own
comboImage	Combo image to use instead of Codename One drawing it on its own
commandBehavior	Deprecated: Don't use this constant as it conflicts with the Toolbar . Indicates how commands should act, as a touch menu, native menu etc. Possible values: SoftKey, Touch, Bar, Title, Right, Native
ComponentGroupBool	Enables component group, which allows components to be logically grouped together, so the UIID's of components would be modified based on their group placement. This allows for some unique styling effects where the first/last elements have different styles from the rest of the elements. It's disabled by default, thus leaving its usage up to the designer
dialogTransitionIn	Default transition for dialog
dialogTransitionInImage	Default transition Image [https://www.codenameone.com/javadoc/com/codename1/ui/Image.html] for dialog, causes a Timeline [https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html] transition effect
dialogTransitionOut	Default transition for dialog
defaultCommandImage	An image to place on a command if none is defined, only applies to touch commands
defaultEmblemImage	The emblem painted on the side of the multibutton, by default this is an arrow on some platforms
dialogTransitionOutImage	Default transition Image [https://www.codenameone.com/javadoc/com/codename1/ui/Image.html] for dialog, causes a Timeline [https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html] transition effect
disabledColor	Color to use when disabling entries by default

Constant	Description/Argument
dlgButtonCommandUUID	The UIID used for dialog button commands
dlgCommandButtonSizeInt	Minimum size to give to command buttons in the dialog
dlgCommandGridBool	Places the dialog commands in a grid for uniform sizes
dlgInvisibleButtons	Includes an RRGGBB color for the line separating dialog buttons, as is the case with Android 4 and iOS 7 buttons in dialogs
dlgSlideDirection	Slide hints
dlgSlideInDirBool	Slide hints
dlgSlideOutDirBool	Slide hints
drawMapPointerBool	Indicates whether a pointer should appear in the center of the map component
fadeScrollBarBool	Boolean indicating if the scrollbar should fade when there is inactivity
fadeScrollEdgeBool	Places a fade effect at the edges of the screen to indicate that it's possible to scroll until we reach the edge (common on Android)
fadeScrollEdgeInt	Amount of pixels to fade out at the edge
firstCharRTLBool	Indicates to the GenericListCellRenderer [https://www.codenameone.com/javadoc/com/codename1/ui/list/GenericListCellRenderer.html] that it should determine RTL status based on the first character in the sentence
noTextModeBool	Indicates that the on/off switch in iOS shouldn't draw text on top of the switch, which is the case for iOS 7+ but not for prior versions
fixedSelectionInt	Number corresponding to the fixed selection constants in List [https://www.codenameone.com/javadoc/com/codename1/ui/List.html]
formTransitionIn	Default transition for form
formTransitionInImage	Default transition Image [https://www.codenameone.com/javadoc/com/codename1/ui/Image.html] for form, causes a Timeline [https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html] transition effect
formTransitionOut	Default transition for form

Constant	Description/Argument
formTransitionOutImage	Default transition Image [https://www.codenameone.com/javadoc/com/codename1/ui/Image.html] for form, causes a Timeline [https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html] transition effect
globalToobarBool	Indicates that the Toolbar API should be on/off by default for all forms
hasRaisedButtonBool	Is true in platforms where the theme has the RaisedButton UIID defined. This is currently only true in the native Android theme to allow some material design guidelines
hideBackCommandBool	Hides the back command from the side menu when possible
hideEmptyTitleBool	Indicates that a title with no content should be hidden even if the border for the title occupies space
hideLeftSideMenuBool	Hides the side menu icon that appears on the left side of the UI
ignorListFocusBool	Hide the focus component of the list when the list doesn't have focus
infiniteImage	The image used by the infinite progress component, the component will rotate it as needed
includeNativeBool	True to derive from the platform native theme, false to create a blank theme that only uses the basic defaults
labelGap	Positive floating point value representing the default gap value between the label text and the icon in millimeters
listItemGapInt	Built-in item gap in the list, this defaults to 2, which predicated padding/margin in Codename One
listLongPressBool	Indicates whether a list should handle long press events, defaults to true
mapTileLoadingImage	An image to preview while loading the MapComponent [https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html] tile

Constant	Description/Argument
mapTileLoadingText	The text of the tiles in the MapComponent [https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html] during loading, defaults to "Loading..."
mapZoomButtonsBool	Indicates whether buttons should be drawn on the map component
mediaBackImage	Media icon used by the media player class
mediaFwdImage	Media icon used by the media player class
mediaPauseImage	Media icon used by the media player class
mediaPlayImage	Media icon used by the media player class
menuButtonBottomBool	When set to true this flag aligns the menu button to the bottom portion of the title. Defaults to false
menuButtonTopBool	When set to true this flag aligns the menu button to the top portion of the title. Defaults to false
menuHeightPercent	Allows positioning and sizing the menu
menuImage	The three dot menu image used in Android and the Toolbar [https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html] to show additional command entries
menuImageSize	The size in millimeters (floating point value) of the generated side menu image, this is only used if you don't supply a custom image. The default value is 4.5.
menuPrefSizeBool	Allows positioning and sizing the menu
menuSlideDirection	Defines menu entrance effect
menuSlideInDirBool	Defines menu entrance effect
menuSlideOutDirBool	Defines menu entrance effect
menuTransitionIn	Defines menu entrance effect
menuTransitionInImage	Defines menu entrance effect
menuTransitionOut	Defines menu exit effect
menuTransitionOutImage	Defines menu entrance effect
menuWidthPercent	Allows positioning and sizing the menu
minimizeOnBackBool	Indicates whether the form should minimize the entire application when the physical back button is pressed (if available) and no command is defined as the back command. Defaults to true

Constant	Description/Argument
onOffIOSModeBool	Indicates whether the on/off switch should use the iOS or Android mode
otherPopupRendererBool	Indicates that a separate renderer UIID/instance should be used to the list within the combo box popup
PackTouchMenuBool	Enables preferred sized packing of the touch menu (true by default), when set to false this allows manually determining the touch menu size using percentages
paintsTitleBarBool	Indicates that the StatusBar UIID should be added to the top of the form to space down the title area, as is the case on iOS 7+ where the status bar is painted on top of the UI
popupCancelBodyBool	Indicates that a cancel button should appear within the combo box popup
PopupDialogArrowBool	Indicates whether the popup dialog has an arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowBottomImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowTopImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowLeftImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowRightImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
popupNoTitleAddPaddingInt	Adds padding to a popup when no title is present
popupTitleBool	Indicates that a title should appear within the combo box popup
pullToRefreshImage	The arrow image used to draw the pullToRefresh animation
pureTouchBool	Indicates the pure touch mode
radioOppositeSideBool	Indicates the radio button should be drawn on the opposite side to the text and not next to the text
radioSelectedDisImage	Radio button image

Constant	Description/Argument
radioSelectedImage	Radio button image
radioUnselectedDisImage	Radio button image
radioUnselectedImage	Radio button image
radioSelectedDisFocusImage	Radio button image
radioSelectedFocusImage	Radio button image
radioUnselectedDisFocusImage	Radio button image
radioUnselectedFocusImage	Radio button image
releaseRadiusInt	Indicates the distance from the button with dragging, in which the button should be released, defaults to 0
rendererShowsNumbersBool	Indicates whether renderers should render the entry number
reverseSoftButtonsBool	Swaps the softbutton positions
rightSideMenuImage	Same as sideMenuImage only for the right side, optional and defaults to sideMenuImage
rightSideMenuPressImage	Same as sideMenuPressImage only for the right side, optional and defaults to sideMenuPressImage
scrollVisibleBool	true/false default is platform dependent. Toggles whether the scroll bar is visible
showBackCommandOnTitleBool	Used by the Toolbar [https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html] API to indicate whether the back button should appear on the title
shrinkPopupTitleBool	Indicates the title of the popup should be set to 0 if it's missing
sideMenuAnimSpeedInt	The speed at which a sidemenu moves defaults to 300 milliseconds
sideMenuFoldedSwipeBool	Indicates the side menu could be opened via swiping
sideMenuImage	The image representing the side menu, three lines (Hamburger menu)
sideMenuPressImage	Optional pressed version of the sideMenuImage
sideMenuScrollVisibleBool	Indicates whether the scroll bar on the side menu should be visible or not, defaults to hidden
sideMenuShadowBool	Indicates whether the shadow for the side menu should be drawn

Constant	Description/Argument
sideMenuShadowImage	The image used when drawing the shadow (a default is used if this isn't supplied)
sideMenuSizeTabPortraitInt	The size of the side menu when expanded in a tablet in portrait mode
sideMenuSizePortraitInt	The size of the side menu when expanded in a phone in portrait mode
sideMenuSizeTabLandscapeInt	The size of the side menu when expanded in a tablet in landscape mode
sideMenuSizeLandscapeInt	The size of the side menu when expanded in a phone in landscape mode
sideMenuTensileDragBool	Enables/disables the tensile drag behavior within the opened side menu
sideSwipeActivationInt	Indicates the threshold in the side menu bar at which a swipe should trigger activation, defaults to 15 (percent)
sideSwipeSensitiveInt	Indicates the region of the screen that is sensitive to side swipe in the side menu bar, defaults to 10 (percent)
slideDirection	Default slide transition settings
slideInDirBool	Default slide transition settings
slideOutDirBool	Default slide transition settings
sliderThumbImage	The thumb image that can appear on the sliders
snapGridBool	Snap to grid toggle
statusBarScrollsUpBool	Indicates that a tap on the status bar should scroll up the UI, only relevant in OS's where paintsTitleBarBool is true
switchButtonPadInt	Indicates the padding in the on/off switch, defaults to 16
switchMaskImage	Indicates the mask image used in iOS mode to draw on top of the switch
switchOnImage	Indicates the on image used in iOS mode to draw the on/off switch
switchOffImage	Indicates the off image used in iOS mode to draw the on/off switch
TabEnableAutoImageBool	Indicates images should be filled by default for tabs
TabSelectedImage	Default selected image for tabs (if TabEnableAutoImageBool=true)

Constant	Description/Argument
TabUnselectedImage	Default unselected image for tabs (if TabEnableAutoImageBool=true)
tabPlacementInt	The placement of the tabs in the Tabs [https://www.codenameone.com/javadoc/com/codename1/ui/Tabs.html] component: TOP = 0, LEFT = 1, BOTTOM = 2, RIGHT = 3
tabsSlideSpeedInt	The time of the animation that occurs (in milliseconds) between between releasing a swiped tab and reaching the next tab. Currently defaults to 200
tabsFillRowsBool	Indicates if the tabs should fill the row using flow layout
tabsGridBool	Indicates whether tabs should use a grid layout thus forcing all tabs to have identical sizes
tabsOnTopBool	Indicates the tabs should be drawn on top of their content in a layered UI, this allows a tab to intrude into the content of the tabs
textCmpVAlignInt	The vertical alignment of the text component: TOP = 0, CENTER = 4, BOTTOM = 2
textComponentErrorColor	A hex RGB color which defaults to null in which case this has no effect. When defined this will change the color of the border and label to the given color to match the material design styling. This implements the red border underline in cases of error and the label text color change
textComponentOnTopBool	Toggles the on top mode which makes things look like they do on Android. This defaults to true on Android and false on other OS's. This can also be manipulated via the onTopMode(boolean) method in InputComponent however the layout will only use the theme constant
textComponentAnimBool	toggles the animation mode which again can be manipulated by a method in InputComponent . If you want to keep the UI static without the floating hint effect set this to false. Notice this defaults to true only on Android

Constant	Description/Argument
textComponentFieldUIID	sets the UIID of the text field to something other than TextField this is useful for platforms such as iOS where the look of the text field is different within the text component. This allows us to make the background of the text field transparent when it's within the TextComponent and make it different from the regular text field
textFieldCursorColorInt	The color of the cursor as an integer (not hex)
tickerSpeedInt	The speed of label/button etc. (in milliseconds)
tintColor	The aarrggbb hex color to tint the screen when a dialog is shown
topMenuSizeTabPortraitInt	The size of the side menu when expanded and attached to the top in a tablet in portrait mode
topMenuSizePortraitInt	The size of the side menu when expanded and attached to the top in a phone in portrait mode
topMenuSizeTabLandscapeInt	The size of the side menu when expanded and attached to the top in a tablet in landscape mode
topMenuSizeLandscapeInt	The size of the side menu when expanded and attached to the top in a phone in landscape mode
touchCommandFillBool	Indicates how the touch menu should layout the commands within
touchCommandFlowBool	Indicates how the touch menu should layout the commands within
transitionSpeedInt	Indicates the default speed for transitions
treeFolderImage	Picture of a folder for the Tree class [https://www.codenameone.com/javadoc/com/codename1/ui/tree/Tree.html] class
treeFolderOpenImage	Picture of a folder expanded for the Tree class
treeNodeImage	Picture of a file node for the Tree class
tensileDragBool	Indicates that tensile drag should be enabled/disabled. This is usually set by platform themes

Dynamic Theme Swapping & Theme Constants

Once a theme constant is set by a theme, it isn't removed on a refresh when replacing the theme.

E.g. if one would set the `comboImage` constant to a specific value in theme A and then switch to theme B, that doesn't define the `comboImage`, the original theme A `comboImage` might remain!

The reason for this is simple: when extracting the constant values, components keep the values in cache locally and just don't track the change in value. Furthermore, since the components allow manually setting values, it's impractical for them to track whether a value was set by a constant or explicitly by the user.

The solution for this is to either manually reset undesired values before replacing a theme (e.g. for the case, above by calling the default look and feel method for setting the combo image with a null value), or defining a constant value to replace the existing value.

4.5. Native Theming

Codename One uses a theme constant called `includeNativeBool`, when that constant is set to `true` Codename One starts by loading the native theme first and then applying all the theme settings. This effectively means your theme "derives" the style of the native theme first, similar to the cascading effect of CSS. Internally this is exactly what the [theme layering](#) section covered.

By avoiding this flag you can create themes that look *EXACTLY* the same on all platforms.



If you avoid the native theming you might be on your own. A few small device oddities such as the iOS status bar are abstracted by native theming. Without it you will need to do everything from scratch

You can simulate different OS platforms by using the native theme menu option

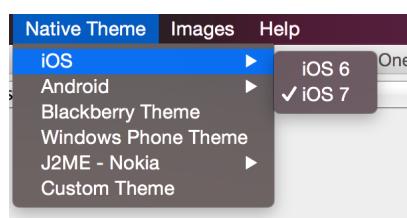


Figure 140. The native theme menu option

Developers can pick the platform of their liking and see how the theme will appear in that particular platform by selecting it and having the preview update on the fly.

4.6. Under the Hood of the Theme Engine

To truly understand a theme we need to understand what it is. Internally a theme is just a [Hashtable](#) key/value pair between UIID based keys and their respective values. E.g. the key:

```
Button.fgColor=ffffff
```

Will set the foreground color of the [Button](https://www.codenameone.com/javadoc/com/codename1/ui/Button.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Button.html] UIID to white.

When a Codename One Component [https://www.codenameone.com/javadoc/com/codename1/ui/Component.html] is instantiated it requests a [Style](https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html) [https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html] object from the [UIManager](https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html) [https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html] class. The [Style](#) object is based on the settings within the theme and can be modified thru code or by using the theme.

We can replace the theme dynamically in runtime and refresh the styles assigned to the various components using the [refreshTheme\(\)](https://www.codenameone.com/javadoc/com/codename1/ui/Component.html#refreshTheme--) [https://www.codenameone.com/javadoc/com/codename1/ui/Component.html#refreshTheme--] method.



It's a common mistake to invoke [refreshTheme\(\)](#) without actually changing the theme. We see developers doing it when all they need is a [repaint\(\)](#) or [revalidate\(\)](#). Since [refreshTheme\(\)](#) is **very** expensive we recommend that you don't use it unless you really need to...

A theme [Hashtable](#) key is comprised of:

[\[UIID.\]\[type#\]attribute](#)

The *UIID*, corresponds to the component's UIID e.g. [Button](#), [CheckBox](#) [https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html] etc. It is optional and may be omitted to address the global default style.

The type is omitted for the default unselected type, and may be one of *sel* (selected type), *dis* (disabled type) or *press* (pressed type). The attribute should be one of:

- [derive](#) - the value for this attribute should be a string representing the base component.
- [bgColor](#) - represents the background color for the component, if applicable, in a web hex string format RRGGBB e.g. ff0000 for red.
- [fgColor](#) - represents the foreground color, if applicable.
- [border](#) - an instance of the border class, used to display the border for the component.
- [bgImage](#) - an [Image](#) [https://www.codenameone.com/javadoc/com/codename1/ui/Image.html] object used in the background of a component.
- [transparency](#) - a [String](#) containing a number between 0-255 representing the alpha value for the background. This only applies to the [bgColor](#).
- [margin](#) - the margin of the component as a [String](#) containing 4 comma separated numbers for top,bottom,left,right.
- [padding](#) - the padding of the component, it has an identical format to the margin attribute.
- [font](#) - A [Font](#) [https://www.codenameone.com/javadoc/com/codename1/ui/Font.html] object instance.
- [alignment](#) - an [Integer](#) object containing the LEFT/RIGHT/CENTER constant values defined in

Component.

- **textDecoration** - an `Integer` value containing one of the `TEXT_DECORATION_*` constant values defined in `Style`.
- **backgroundType** - a `Byte` object containing one of the constants for the background type defined in `Style` [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html>] under `BACKGROUND_*`.
- **backgroundGradient** - contains an `Object` array containing 2 integers for the colors of the gradient. If the gradient is radial it contains 3 floating points defining the x, y & size of the gradient.

So to set the foreground color of a selected button to red, a theme will define a property like:

```
Button.sel#fgColor=ff0000
```

This information is mostly useful for understanding how things work within Codename One, but it can also be useful in runtime.

E.g. to increase the size of all fonts in the application, we can do something like:

```
Hashtable h = new Hashtable();
h.put("font", largeFont);
UIManager.getInstance().addThemeProps(h);
Display.getInstance().getCurrent().refreshTheme();
```

4.7. Understanding Images and Multi-Images



This section provides a very high level overview of images. We dive deeper into the various types of images in the [graphics section](#) [<https://www.codenameone.com/manual/graphics.html#deep-into-images-section>].

When working with a theme, we often use images for borders or backgrounds. We also use images within the GUI for various purposes and most such images will be extracted from the resource file.

Adding a standard JPEG/PNG image to the resource file is straight forward, and the resulting image can be viewed within the images section. However, due to the wide difference between device types, an image that would be appropriate in size for an iPhone 3gs would not be appropriate in size for a Nexus device or an iPhone 4 (but perhaps, surprisingly, it will be just right for iPad 1 and iPad 2).

The density of the devices varies significantly and Codename One tries to simplify the process by unifying everything into one set of values to indicate density. For simplicity's sake, density is sometimes expressed in terms of pixels, however it is mapped internally to actual screen measurements where possible.

A multi-image is an image that has multiple varieties for different densities, and thus looks sharp in all the densities. Since scaling on the device can't interpolate the data (due to performance considerations), significant scaling on the device becomes impractical. However, a multi-image will just provide the "right" resolution image for the given device type.

From the programming perspective this is mostly seamless, a developer just accesses one image and has no ability to access the images in the different resolutions. Within the designer, however, we can explicitly define images for multiple resolutions and perform high quality scaling so the “right” image is available.

We can use two basic methods to add a multi-image: quick add and standard add.

Both methods rely on understanding the source resolution of the image, e.g. if you have an icon that you expect to be 128x128 pixels on iPhone 4, 102x102 on nexus one and 64x64 on iPhone 3gs. You can provide the source image as the 128 pixel image and just perform a quick add option while picking the **Very High** density option.

This will indicate to the algorithm that your source image is designed for the “very high” density and it will scale for the rest of the densities accordingly.



This relies on the common use case of asking your designer to design for one high end device (e.g. iPhone X) then you can take the resources and add them as “HD” resources. They will automatically adapt to the lower resolutions

Alternatively, you can use the standard add multi-image dialog and set it like this:



Notice that we selected the square image option, essentially eliminating the height option. Setting values to 0 prevents the system from generating a multi-image entry for that resolution, which will mean a device in that category will fall on the closest alternative.

The percentage value will change the entire column, and it means the percentage of the screen. E.g. We know the icon is 128 for the very high resolution, we can just move the percentage until we reach something close to 128 in the “Very High” row and the other rows will represent a size that should be pretty close in terms of physical size to the 128 figure.

At runtime, you can always find the host device’s approximate pixel density using the `Display.getDeviceDensity()` method. This will return one of:

Table 4. Densities

Constant	Density	Example Device
<code>Display.DENSITY_VERY_LOW</code>	~ 88 ppi	
<code>Display.DENSITY_LOW</code>	~ 120 ppi	Android ldpi devices
<code>Display.DENSITY_MEDIUM</code>	~ 160 ppi	iPhone 3GS, iPad, Android mdpi devices
<code>Display.DENSITY_HIGH</code>	~ 240 ppi	Android hdpi devices
<code>Display.DENSITY_VERY_HIGH</code>	~ 320 ppi	iPhone 4, iPad Air 2, Android xhdpi devices
<code>Display.DENSITY_HD</code>	~ 540 ppi	iPhone 6+, Android xxhdpi devices
<code>Display.DENSITY_560</code>	~ 750 ppi	Android xxxhdpi devices
<code>Density.DENSITY_2HD</code>	~ 1000 ppi	
<code>Density.DENSITY_4K</code>	~ 1250ppi	

4.8. Use Millimeters for Padding/Margin and Font Sizes

When configuring your styles, you should almost never use "Pixels" as the unit for padding, margins, font size, and border thickness because the results will be inconsistent on different densities. Instead, you should use millimeters for all non-zero units of measurement.

As we now understand the [complexities of DPI](#) it should be clear why this is important.

4.8.1. Fractions of Millimeters

Sometimes millimeters don't give you enough precision for what you want to do. Currently the designer only allows you to specify integer values for most units. However, you can achieve more precise results when working directly in Java. The `Display.convertToPixels()` method will allow you to convert millimeters (or DIPS) to pixels. It also only takes an integer input, but you can use it to obtain a multiplier that you can then use to convert any millimeter value you want into pixels.

E.g.

```
double pixelsPerMM = ((double)Display.getInstance().convertToPixels(10, true)) / 10.0;
```

And now you can set the padding on an element to 1.5mm. E.g.

```
myButton.getAllStyles().setPaddingUnit(Style.UNIT_TYPE_PIXELS);
int pixels = (int)(1.5 * pixelsPerMM);
myButton.getAllStyles().setPadding(pixels, pixels, pixels, pixels);
```

4.9. Creating a Great Looking Side Menu

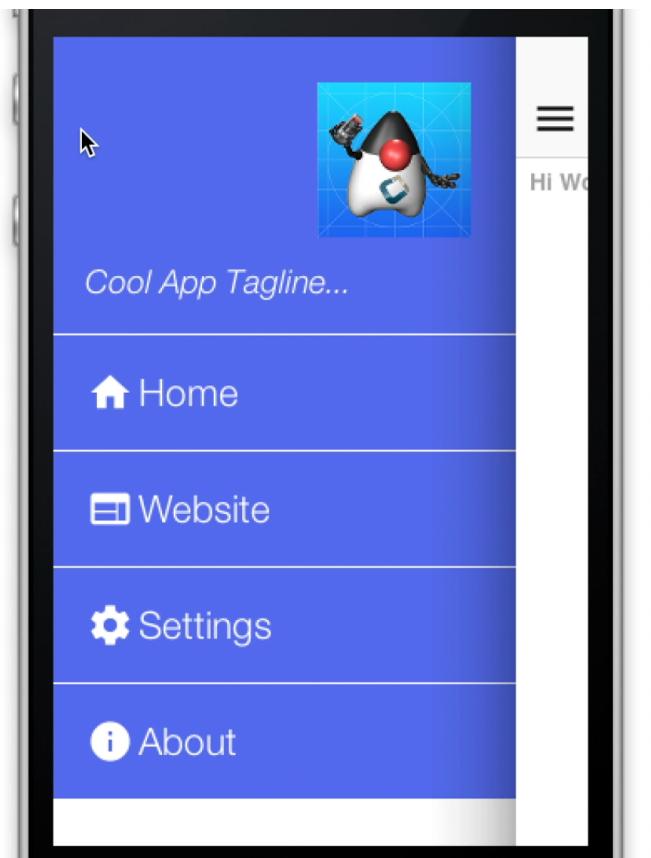


Figure 141. Side Menu final result

A side menu is a crucial piece of an elegant application. We'll explain how one creates a simple side menu that's elegant, portable and easy to build. This is a good "starting point" side menu from which you can build more elaborate designs.

To get this result we will start from a native theme and a bare bones application to keep things simple.

The code for the side menu is this:

```

Form hi = new Form("Hi World");

Toolbar tb = hi.getToolbar();
Image icon = theme.getImage("icon.png"); ①
Container topBar = BorderLayout.east(new Label(icon));
topBar.add(BorderLayout.SOUTH, new Label("Cool App Tagline...", "SidemenuTagline")); ②
topBar.setUIID("SideCommand");
tb.addComponentToSideMenu(topBar);

tb.addMaterialCommandToSideMenu("Home", FontImage.MATERIAL_HOME, e -> {});
tb.addMaterialCommandToSideMenu("Website", FontImage.MATERIAL_WEB, e -> {});
tb.addMaterialCommandToSideMenu("Settings", FontImage.MATERIAL_SETTINGS, e -> {});
tb.addMaterialCommandToSideMenu("About", FontImage.MATERIAL_INFO, e -> {});

hi.addComponent(new Label("Hi World"));
hi.show();

```

- ① This is the icon which was used in lieu of a logo it appears in the top right of the side menu
- ② This is the top bar containing the tagline and the icon it's styled as if it's a command but you can put anything here e.g. an image etc.
- ③ The commands are added as usual to the side menu with no styling or functionality, the entire look is determined by the theme

Next we'll open the designer tool to style the UI

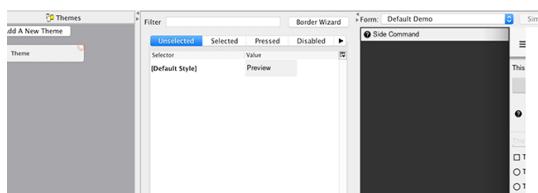


Figure 142. Open the side menu so we will get the right values in the combo box on add

Now when we press **Add** the side menu entries will appear in the combo box (you can type them but this is more convenient). We'll start with the **SideNavigationViewPanel** style:

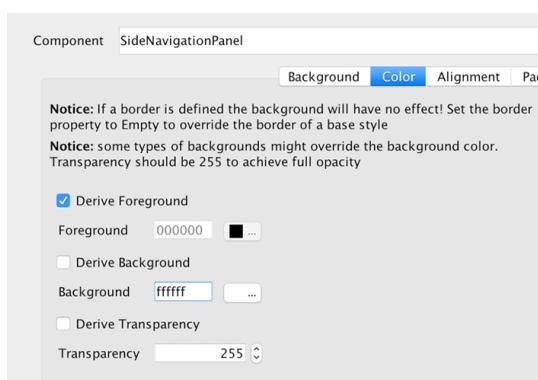


Figure 143. The SideNavigationViewPanel has an opaque white background

The **SideCommand** style is a bit more elaborate, we start with a white foreground and an opaque bluish/purple color:

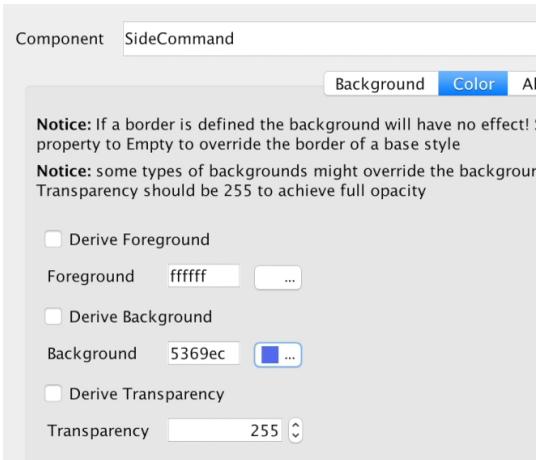


Figure 144. The SideCommand has a white foreground and opaque bluish background

We'll set padding to 3 millimeters which gives everything a good feel and spacing. This is important for finger touch sensitivity.

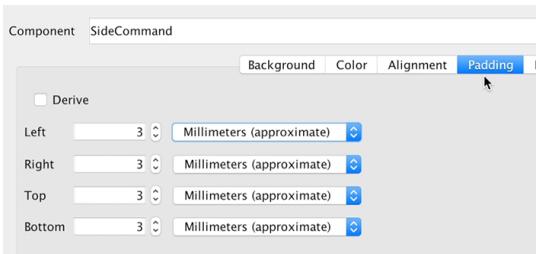


Figure 145. Padding is 3mm so it will feel spacious and touch friendly

We'll set margin to 0 except for the bottom one pixel which will leave a nice white line by showing off the background. This means the commands will have a space between them and the white style we gave to the **SideNavigationViewPanel** will appear thru that space.

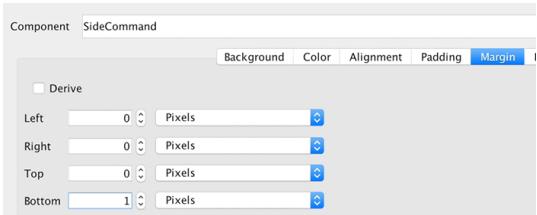


Figure 146. Margin is 0 except for a thin line below each command

Setting the border to empty is crucial!

The iOS version of the side command inherits a border style so we must "remove" it by defining a different border in this case an empty border. Since borders take precedence over color this would have prevented the color changes we made from appearing.

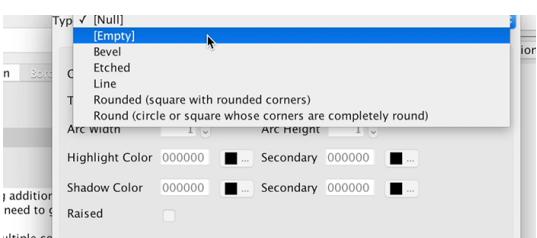


Figure 147. Border must be defined as Empty

Next we need to pick a good looking font and make sure it's large enough. We use millimeters size it correctly for all OS's and override the derived text decoration which has a value in the iOS native theme so it can impact the final look.

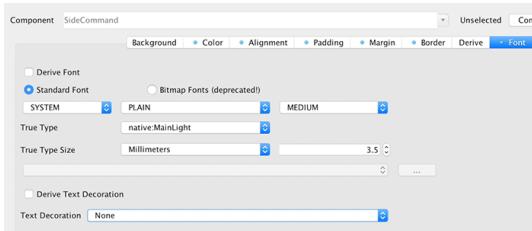


Figure 148. Pick a good looking font for the side command

Next we need to move to the selected tab and add a new side command entry that derives from the unselected version. We'll pick a new color that's slightly deeper and will make the selected style appear selected. We'll also copy and paste this selected style to the pressed style.



Figure 149. Selected & Pressed SideCommand

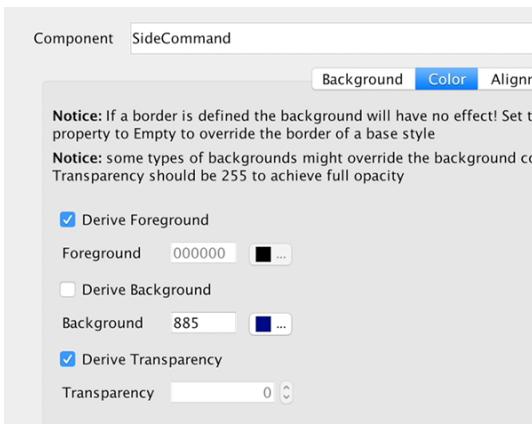


Figure 150. Color for the Selected/Pressed SideCommand

The **SidemenuTagline** is just a **SideCommand** style that was slightly adapted. We'll remove the padding and margin because the whole section is wrapped in a side command and we don't want double padding. We'll leave 1mm padding at the top for a bit of spacing from the logo.

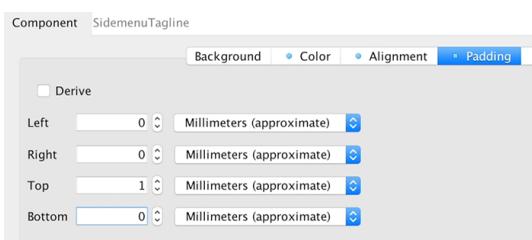


Figure 151. Padding of the SidemenuTagline

We'll also update the font to a smaller size and italic styling so it will feel like a tagline.

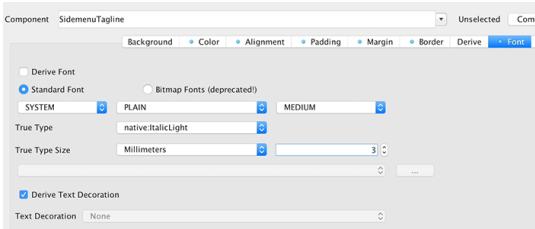


Figure 152. Font for the SideMenuTagline is slightly smaller and italic

The last change for the theme is for the **StatusBarSideMenu** UIID which is a spacing on the top of the sidemenu. This spacing is there for iOS devices which render the clock/battery/reception symbols on top of the app. We'll set the padding to 0.

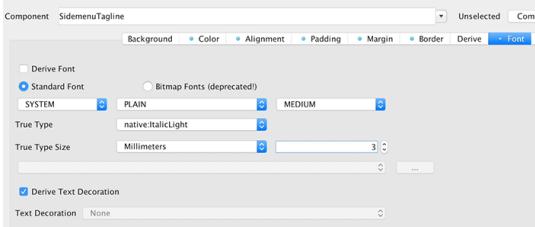


Figure 153. StatusBarSideMenu padding for the top of the side menu

Finally, we'll add the icon image (or a logo if you have it) into the theme as a multi image so we can use it within the side menu as a good looking logo. A relatively large icon image works as a 2HD multi-image but you can use many strategies to get a fitting image for this spot.



Rounded images work well here, you can round images dynamically using masking

These steps produce the UI above as a side menu, they might seem like a long set of steps but each step is pretty simple as you walk thru each one. This does show off the versatility and power of Codename One as a change to one step can create a radically different UI design.

4.10. Converting a PSD To A Theme

Codename One provides extensive support for designing beautiful user interfaces, but it isn't necessarily obvious to new developers how to achieve their desired results. A common workflow for app design includes a PSD file with mock-ups of the UI, created by a professional designer.



PSD is the Adobe Photoshop file format, it's the most common format for UI designs in the industry

For this tutorial we adapt a very slick looking sign-up form found online and convert it to a Codename One component that can be used inside an application.

The process we followed was:

1. Find the PSD design we want to use: [this PSD file](http://freebiesbug.com/psd-freebies/iphone-6-ui-kit/) [<http://freebiesbug.com/psd-freebies/iphone-6-ui-kit/>] created by [Adrian Chiran](https://dribbble.com/adrianchiran) [<https://dribbble.com/adrianchiran>] (we mirrored it [here](https://www.codenameone.com/files/iOS_UI-Kit.psd) [https://www.codenameone.com/files/iOS_UI-Kit.psd] in case it goes offline):

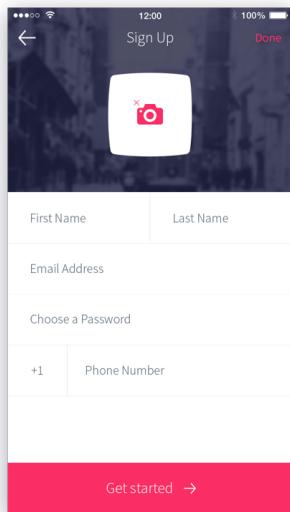


Figure 154. Sign Up form Design

2. Re-create the general structure and layout of the design in a Codename One **Form** using nested components and layout managers. Here is a break-down of how we structured the component hierarchy in the **Form**:

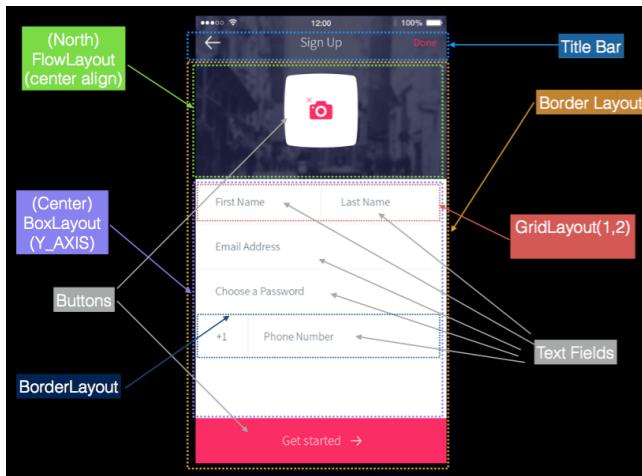


Figure 155. Component hierarchy and layouts

3. Extract the images we needed using Photoshop - this process is often referred to as "cutting"
4. Extract the fonts, colors, and styles we needed to reproduce the design in Codename One
5. Import images into the Codename one project, and define theme styles so that our components match the look of the original design

Here is a screenshot of the resulting component running inside the Codename One simulator:

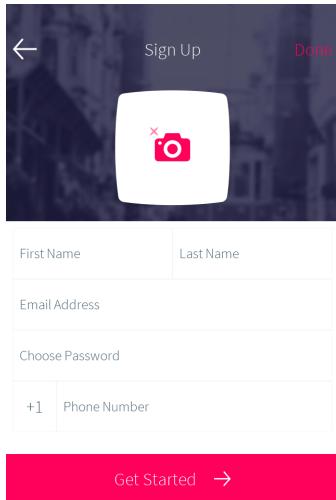


Figure 156. Resulting app in the Codename One simulator

4.10.1. Breaking Down the PSD

Open the PSD you are interested in using Photoshop.



You might be missing fonts in your system so you can either install them or ignore that. Keep in mind that some fonts might not be redistributable with your application

In this PSD we want only one of the screen designs so initially we want to remove everything that isn't related so we can get our bearings more effectively:

- Select the drag tool (the top left tool)
- In the toolbar for the tool (top bar area) check the **Auto Select** mode
- Select the **Layer Mode** for auto selection (in some cases group would actually be better so feel free to experiment)
- Click on the portion in the PSD that you are interested in

You should end up with something like this where a layer is selected in the layers window:

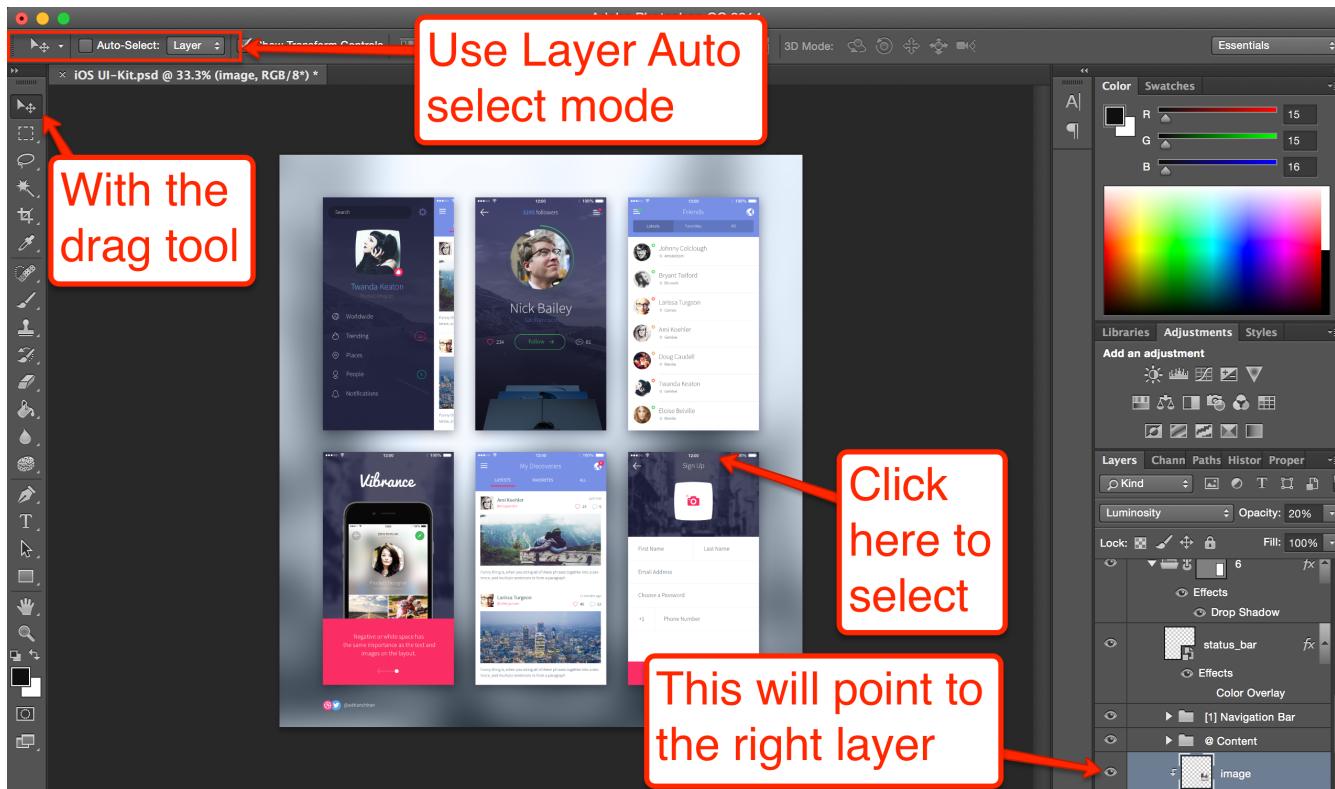


Figure 157. Selecting a layer from the region you are interested in

Scroll up the hierarchy a bit and uncheck/recheck the eye icon on the left until you locate the right element layer.

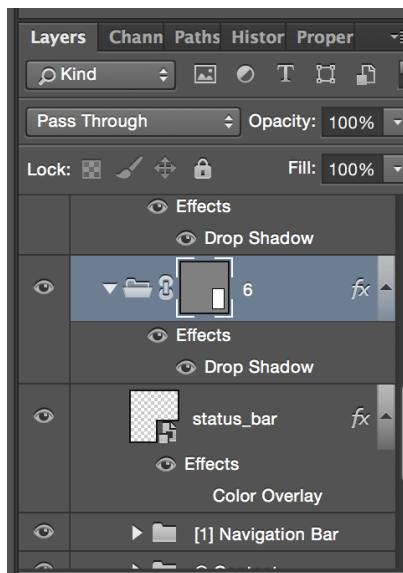


Figure 158. Selecting a layer from the region you are interested in

Right click the layer and select **Convert To Smart Object**.



The right click menu will present different options when you click different areas of the layer, clicking on the left area of the layer works

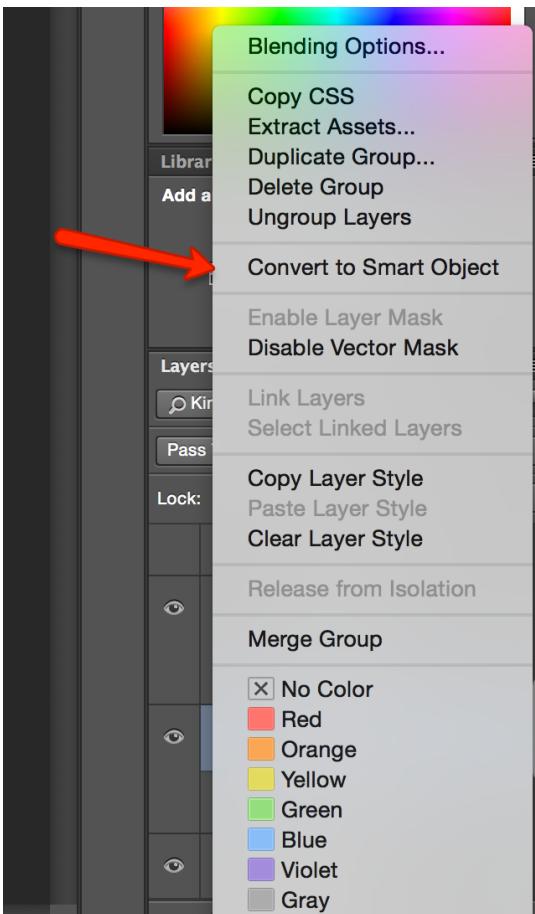


Figure 159. In the right click menu option select "Convert To Smart Object"

Once the layer hierarchy is a smart object you can just double click it which will open the sub hierarchy in a new tab and you now only have the pieces of the image you care about.

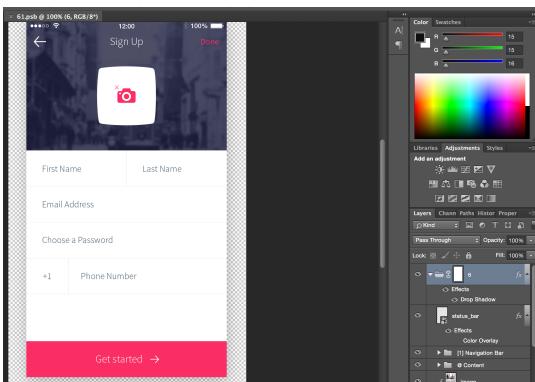


Figure 160. Double clicking the smart object allows us to edit only the form we need

Removing the Noise

The first thing we need to do is remove from the image all of the things that we don't really need. The status bar area on the top is redundant as if is a part of the phones UI. We can select it using the select tool and click the eye icon next to the layer to hide it.

Normally we'd want to have the back arrow but thanks to the material design icons that are a part of Codename One we don't need that icon so we can hide that too.

We don't need the "Sign Up" or "Done" strings in the title either but before removing them we'd like to know the font that is used.

To discover that I can click them to select the layer then switch to the text tool:



Figure 161. The text tool allows us to inspect the font used

Then I can double click the text area layer to find out the font in the top of the UI like this:



Figure 162. The Done toolbar entry uses SourceSansPro Regular



Notice that I don't actually need to have the font installed in this case I don't (hence the square brackets)

Also notice that the color of the font is accessible in that toolbar, by clicking the color element we get this dialog which shows the color value to be **f73267**, this is something we will use later

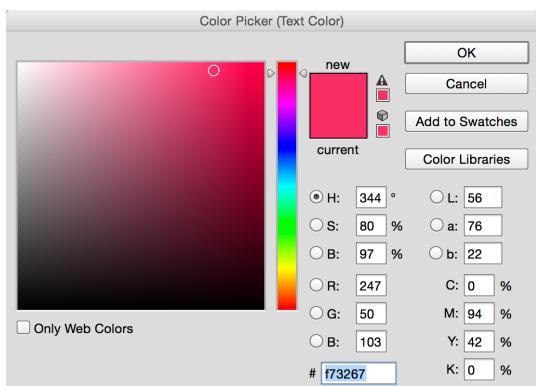


Figure 163. The color dialog lists the hex color at the bottom, we can paste that directly to the designer tool

We can now hide both text layers so they won't pose a problem later.

The Camera Button

The camera button includes an icon and the button background itself. You can just use that as a single image and be done with it, but for the purpose of this tutorial I will take the harder route of separating this into a button background and a foreground image.

When you click on the camera icon you will notice that the camera icon is comprised of two separate layers: the camera and the "x" symbol above it. We can select both layers using **ctrl-click** (command click on the Mac) and convert both to a smart object together using the same method as before:

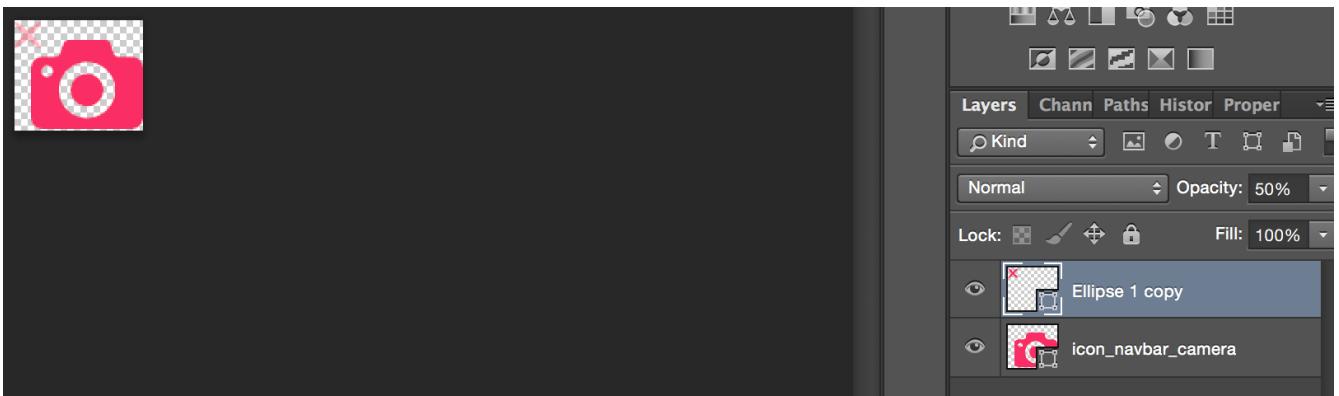


Figure 164. The camera smart object

Since the image is used as an icon we want it to be completely square which isn't the situation here! This is important as a non-square image can trigger misalignment when dealing with icons and the background. So we need to use the **Image** → **Canvas Size** menu and set the values to be the same (the higher value of the two).

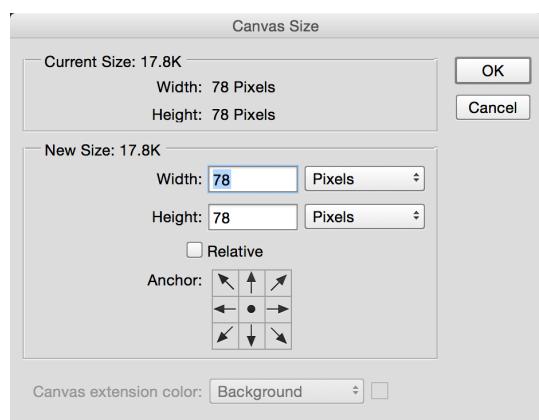


Figure 165. The canvas size dialog for the camera.png file

We can now use **File** → **Export** and save the first image resource we will need into a temporary directory. Make sure to save a PNG file to preserve quality and transparency!



Use **File** → **Export** and never use **File** → **Save As**. The latter can produce a huge size difference as it retains image meta-data

For convenience we'll refer to the file as `camera.png` when we need it later.



Figure 166. The camera icon image

We can follow the exact same procedure with the parent button layer (the white portion) which we can convert to a smart object and export `camera-button.png`.



Figure 167. The camera button image set to a gray background so it will be visible

Now we can hide both of these elements and proceed to get the background image for the title.

Here the "smart object trick" won't work... There is an effects layer in place and the smart object will provide us with the real underlying image instead of the look we actually want. However, solving this is trivial now that we hid all of the elements on top of the image!

We need to switch to the rectangular select tool:

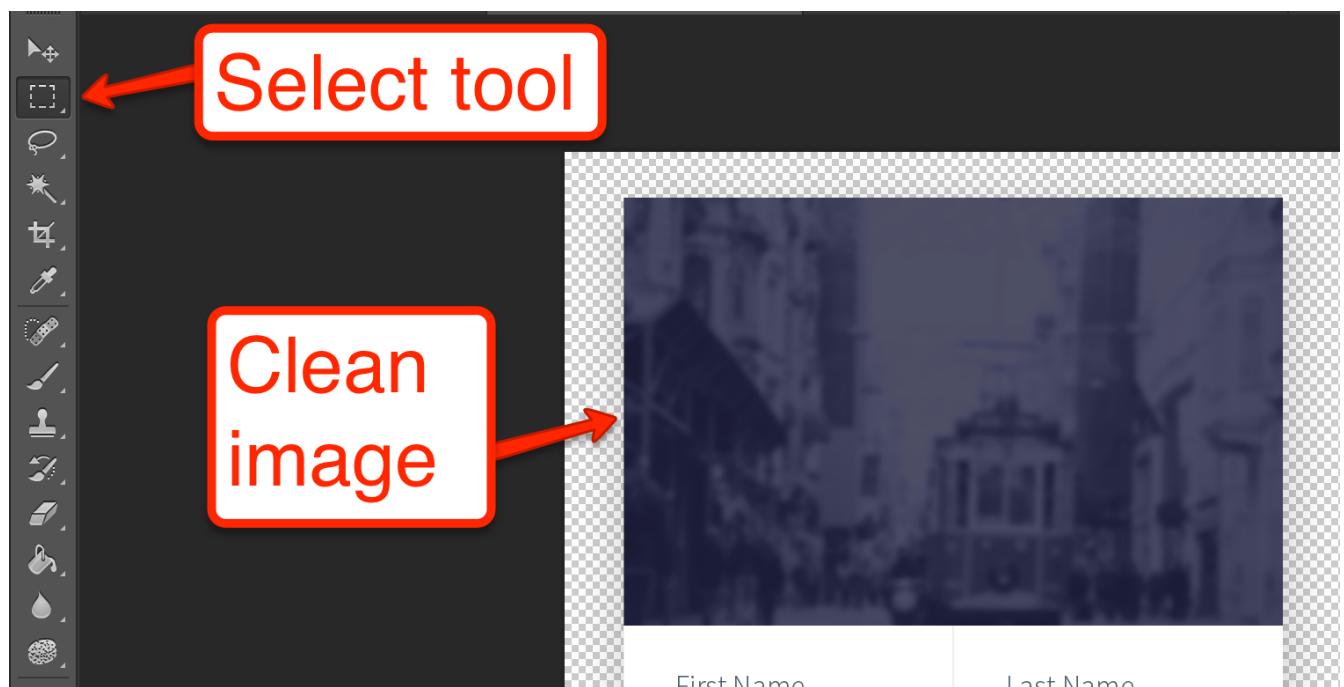


Figure 168. The select tool and the clean image we want to select

Now drag the select tool to select the image don't cross into the white pixels below the image. You can use the zoom value and set it to a very high value to get the selection right.

When the selection is right click **Edit** → **Copy Merged**. Normally **Copy** would only copy a specific layer but in this case we want to copy what we see on the screen!

Now click **File** → **New** it should have the **Presets** set to **Clipboard** which means the newly created image is based on what we just copied (that is seriously great UX). Just accept that dialog and paste (**Ctrl-V** or **Command-V**).

You can now save the image, since it's just a background using JPEG is totally acceptable in this case. We named it **background.jpg**.



Figure 169. The background image

The last thing we need is the colors used in the UI. We can use the "eye drop" tool in a high zoom level to discover the colors of various elements e.g. the text color is `4d606f` and the separator color is `f5f5f5`:

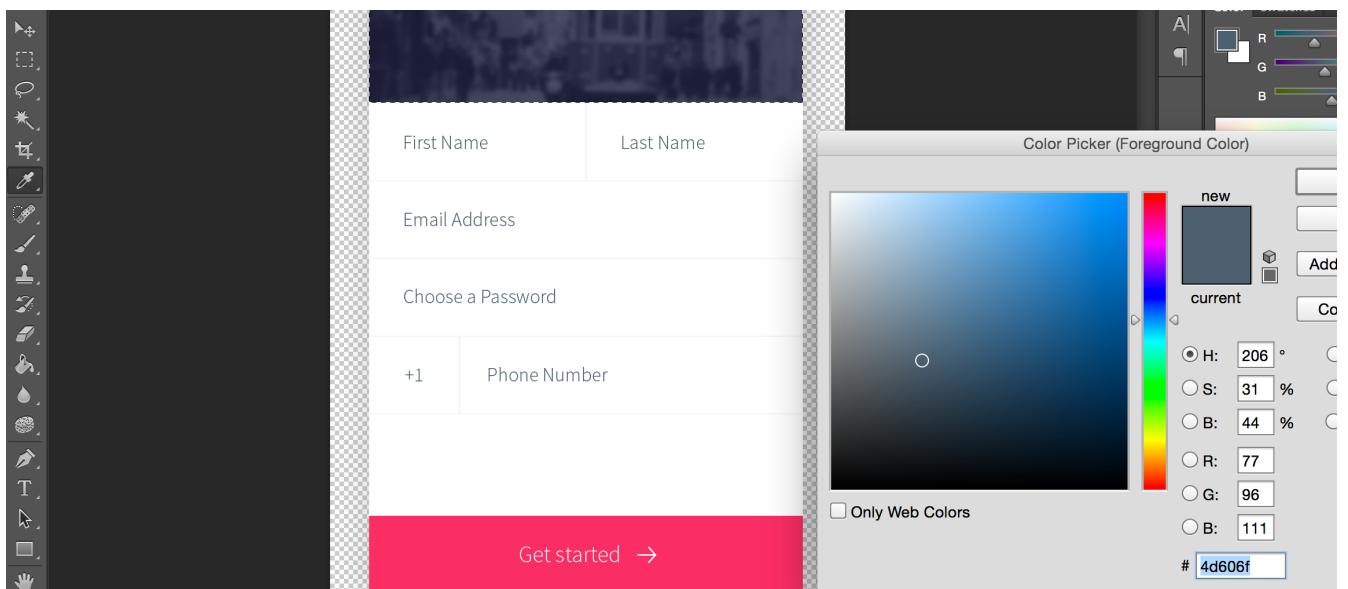


Figure 170. The eye drop tool can be pointed at an area of the image to get the color in that region

4.10.2. The Code

While that was verbose it was relatively simple. We'll create a simple barebones manual application with the native theme.



The reason for this is to avoid "noise", if we use a more elaborate theme it would have some existing settings. This can make the tutorial harder to follow

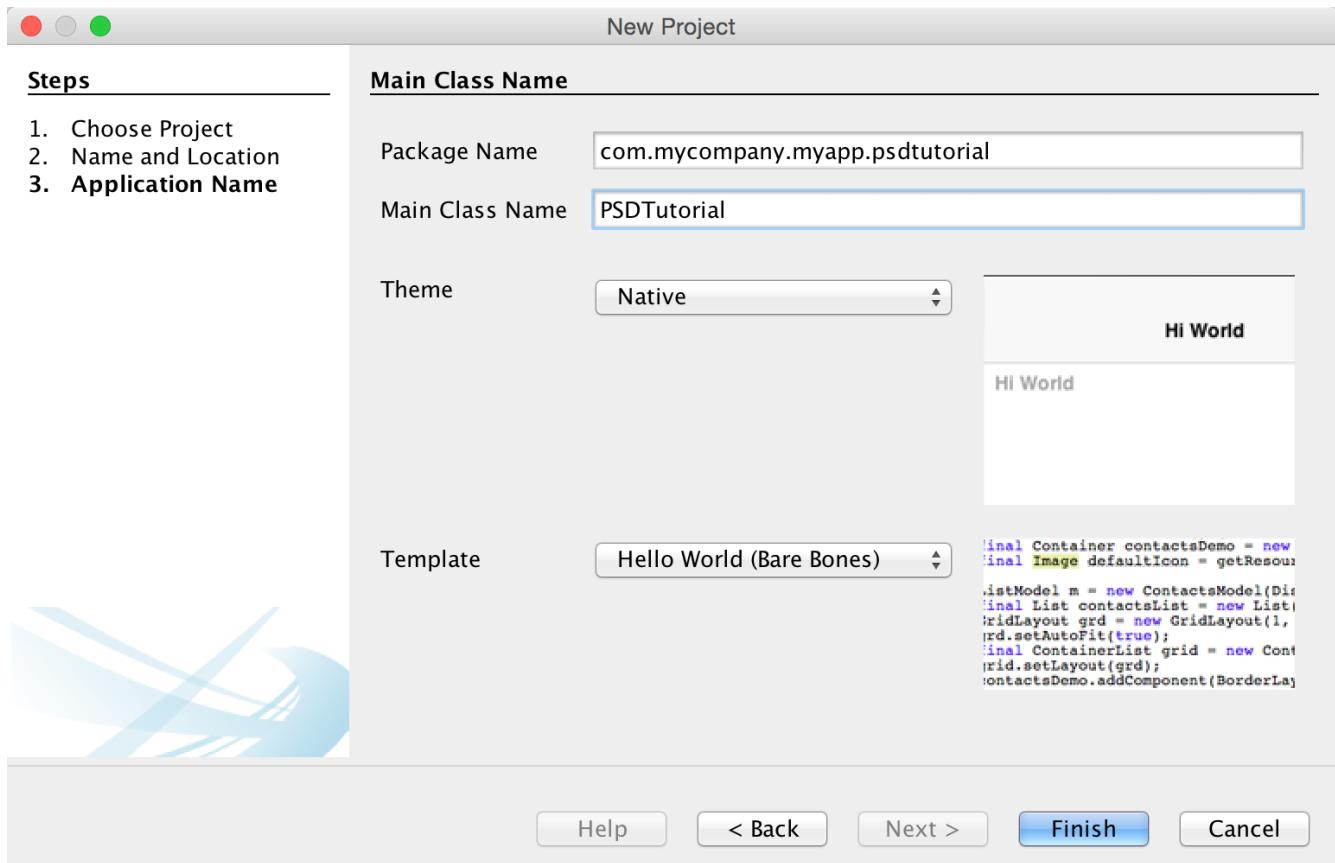


Figure 171. Simple bare bones app settings

Once the project is created double click the `theme.res` file and within the designer select `Images` → `Quick Add Multi Images`. Select the 3 images we created above: `background.jpg`, `camera.png` & `camera-button.png`. Leave the default setting on `Very High` and press `OK`.

Then save the resource file so we can use these images from code.

Here is the source code we used to work with the UI above there are comments within the code explaining some of the logic:

```

private Label createSeparator() {
    Label sep = new Label();
    sep.setUUID("Separator");
    // the separator line is implemented in the theme using padding and background color, by default labels
    // are hidden when they have no content, this method disables that behavior
    sep.setShowEvenIfBlank(true);
    return sep;
}

public void start() {
    if(current != null){
        current.show();
        return;
    }
    // The toolbar uses the layered mode so it resides on top of the background image, the theme makes
    // it transparent so we will see the image below it, we use border layout to place the background image on
    // top and the "Get started" button in the south
    Form psdTutorial = new Form("Signup", new BorderLayout());
    Toolbar tb = new Toolbar(true);
    psdTutorial.setToolbar(tb);

    // we create 4mm material arrow images for the back button and the Get started button

```

```

Style iconStyle = psdTutorial.getUIManager().getComponentStyle("Title");
FontImage leftArrow = FontImage.createMaterial(FontImage.MATERIAL_ARROW_BACK, iconStyle, 4);
FontImage rightArrow = FontImage.createMaterial(FontImage.MATERIAL_ARROW_FORWARD, iconStyle, 4);

// we place the back and done commands in the toolbar, we need to change UIID of the "Done" command
// so we can color it in Red
tb.addCommandToLeftBar("", leftArrow, (e) -> Log.p("Back pressed"));
Command doneCommand = tb.addCommandToRightBar("Done", null, (e) -> Log.p("Done pressed"));
tb.findCommandComponent(doneCommand).setUIID("RedCommand");

// The camera button is comprised of 3 pieces. A label containing the image and the transparent button
// with the camera icon on top. This is all wrapped in the title container where the title background image
// is placed using the theme. We chose to use a Label rather than a background using the cameraLayer so
// the label will preserve the original size of the image without scaling it and take up the space it needs
Button cameraButton = new Button(theme.getImage("camera.png"));
Container cameraLayer = LayeredLayout.encloseIn(
    new Label(theme.getImage("camera-button.png")),
    cameraButton);
cameraButton.setUIID("CameraButton");
Container titleContainer = Container.encloseIn(
    new BorderLayout(BorderLayout.CENTER_BEHAVIOR_CENTER),
    cameraLayer, BorderLayout.CENTER);
titleContainer.setUIID("TitleContainer");

TextField firstName = new TextField("", "First Name");
TextField lastName = new TextField("", "Last Name");
TextField email = new TextField("", "Email Address", 20, TextField.EMAILADDR);
TextField password = new TextField("", "Choose a Password", 20, TextField.PASSWORD);
TextField phone = new TextField("", "Phone Number", 20, TextField.PHONENUMBER);
Label phonePrefix = new Label("+1");
phonePrefix.setUIID("TextField");

// The phone and full name have vertical separators, we use two table layouts to arrange them correctly
// so the vertical separator will be in the right place
TableLayout fullNameLayout = new TableLayout(1, 3);
Container fullName = new Container(fullNameLayout);
fullName.add(fullNameLayout.createConstraint().widthPercentage(49), firstName).
    add(fullNameLayout.createConstraint().widthPercentage(1), createSeparator()).
    add(fullNameLayout.createConstraint().widthPercentage(50), lastName);
Container fullPhone = TableLayout.encloseIn(3, phonePrefix, createSeparator(), phone);

// The button in the south portion needs the arrow icon to be on the right side so we place the text on the left
Button southButton = new Button("Get started", rightArrow);
southButton.setTextPosition(Component.LEFT);
southButton.setUIID("SouthButton");

// we add the components and the separators the center portion contains all of the elements in a box
// Y container which we allow to scroll. BorderLayout Containers implicitly disable scrolling
Container by = BoxLayout.encloseY(
    fullName,
    createSeparator(),
    email,
    createSeparator(),
    password,
    createSeparator(),
    fullPhone,
    createSeparator()
);
by.setScrollableY(true);
psdTutorial.add(BorderLayout.NORTH, titleContainer).
    add(BorderLayout.SOUTH, southButton).
    add(BorderLayout.CENTER, by);

```

```
    psdTutorial.show();
}
```

4.10.3. Styling The UI

So the code above is most of the work but we still need to put everything together using the theme. This is what we have so far:

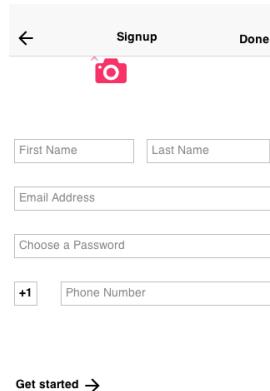


Figure 172. Before applying the changes to the theme this is what we have

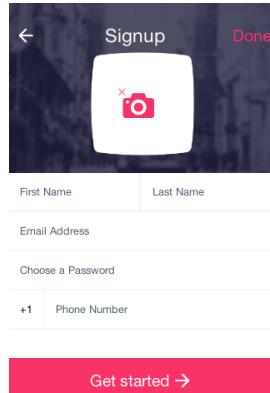


Figure 173. This is what we are aiming at with no additional code changes

This looks like a major set of changes but it requires exactly 10 UIID definitions to get to this look!

Open the designer and select the theme. Press the **Add** button and type in **TitleContainer**. Uncheck **derive** for the background and select **IMAGE_SCALED_FILL** for the **Type** and the **background.jpg** image.

Define the padding as:

- Left - 3 millimeter
- Right - 3 millimeter
- Top - 8 millimeter
- Bottom - 2 millimeter

This will allow enough space for the title. Define margin as 0 on all sides. Then press **OK**.

Add the "Title" UIID. In the **Color** tab define the foreground as **ffffff** define transparency as **0** (fully transparent so we will see the **TitleContainer**). Define padding as 1 millimeter on all sides and

margin as 0 on all sides.

In the **Border** tab press the **...** button and select **[Empty]**.

In the **Font** tab select the **True Type** as **native:MainThin**. Select the **True Type Size** as millimeters and set the value to **3.5**.

Press **OK** to save the changes.

Copy the **Title** UIID and paste it, change the name to "TitleCommand" and press **OK** to save the changes.

Copy the **Title** UIID again and paste it, change the name to "RedCommand". In the **Color** tab set the foreground color to **f73267**. In the **Font** tab set the **True Type** to **native:MainLight** and set the size to **3**. Press **OK** to save the changes.

Add the "TitleArea" UIID. In the **Color** tab define transparency as **0** (fully transparent so we will see the **TitleContainer**). Define padding and margin as 0 on all sides.

In the **Border** tab press the **...** button and select **[Empty]**. Press **OK** to save the changes.

Add the "TextField" UIID. In the **Color** tab define transparency as **255** (fully opaque) and the background as **ffffff** (white). Define padding as 2 millimeter on all sides and margin as 0 on all sides.

In the **Border** tab press the **...** button and select **[Empty]**. In the **Font** tab set the **True Type** to **native:MainLight** and set the size to **2**. Press **OK** to save the changes.

Copy the **TextField** UIID again and paste it, change the name to "TextHint". In the **Color** tab set the foreground color to **4d606f**. Press **OK** to save the changes.

Add the "SouthButton" UIID. In the **Color** tab define transparency as **255** (fully opaque) and the background as **f73267** (red) and the foreground as **ffffff** (white). Define **Alignment** as **Center**.

Define padding as:

- Left - 1 millimeter
- right - 1 millimeter
- top - 2 millimeters
- bottom - 2 millimeters

Define margin as 0 on all sides. In the **Font** tab set the **True Type** to **native:MainThin** and set the size to **3**. Press **OK** to save the changes.

Add the "CameraButton" UIID. In the **Color** tab define transparency as **0** (fully transparent). Define **Alignment** as **Center**.

Define padding as:

- Left - 1 millimeter
- right - 1 millimeter
- top - 3 millimeters

- bottom - 1 millimeter



This helps spacing away from the title

Define margin as 1 millimeter on all sides. Press **OK** to save the changes.

You can now save the theme and the app should look like the final result!

Not Quite There Yet

There is one last piece that you would notice if you actually try to run this code. When pressing the buttons/text fields you would see their look change completely due to the different styles for focus/press behavior.

You can derive the regular styles from the selected/pressed styles but one of the simplest ways is to just copy & paste the styles to the pressed/selected tabs. We can copy `CameraButton`, `RedCommand`, `SouthButton` & `TextField` to the selected state. Then copy `CameraButton`, `RedCommand` & `SouthButton` to the pressed state to get the complete app running!

5. CSS

In this chapter we'll discuss theming with CSS in Codename One.

CSS Changes Don't Require a Recompile



You can change the CSS values while the simulator is running and the changes will reflect in the simulator within a few seconds

5.1. Activating CSS

Codename One applications always use the resource file. The CSS support compiles a file in CSS syntax to a Codename One resource file and adds it to the application. In runtime the CSS no longer exists and the file acts like a regular theme file.

To enable CSS support in Codename One you need to flip a switch in [Codename One Settings](#).

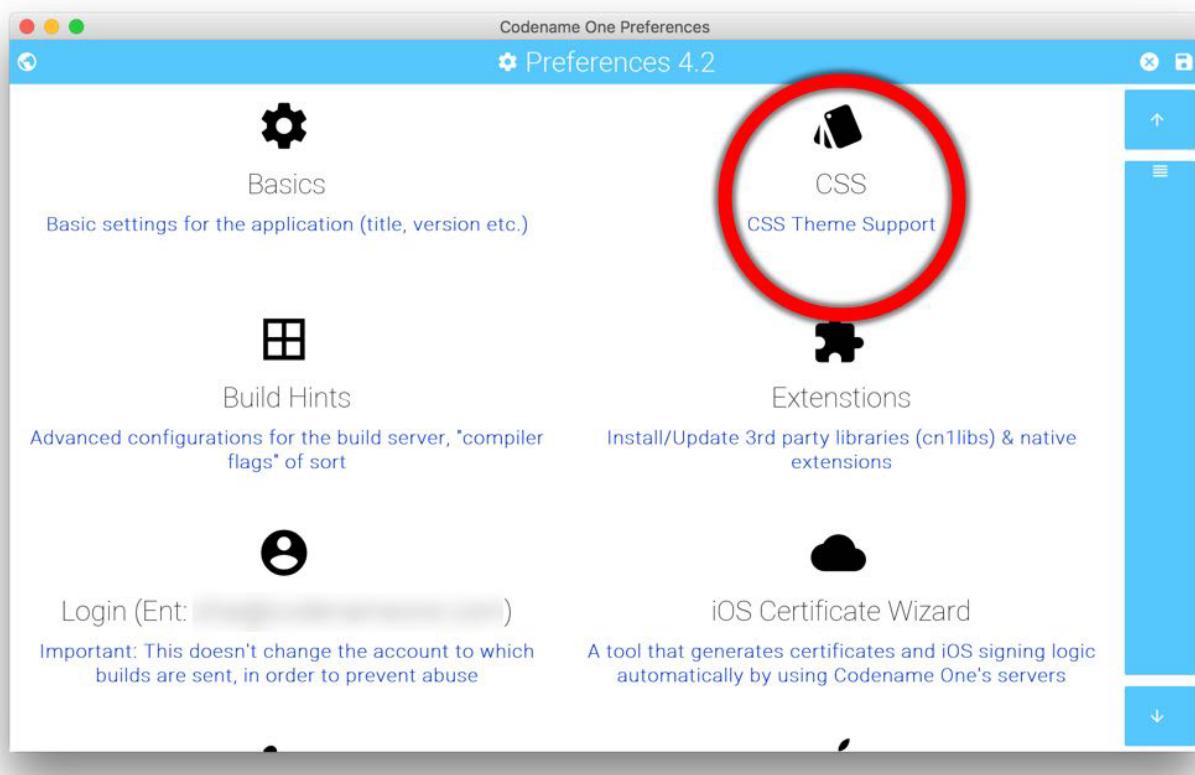


Figure 174. The CSS Option in Codename One Settings Part I

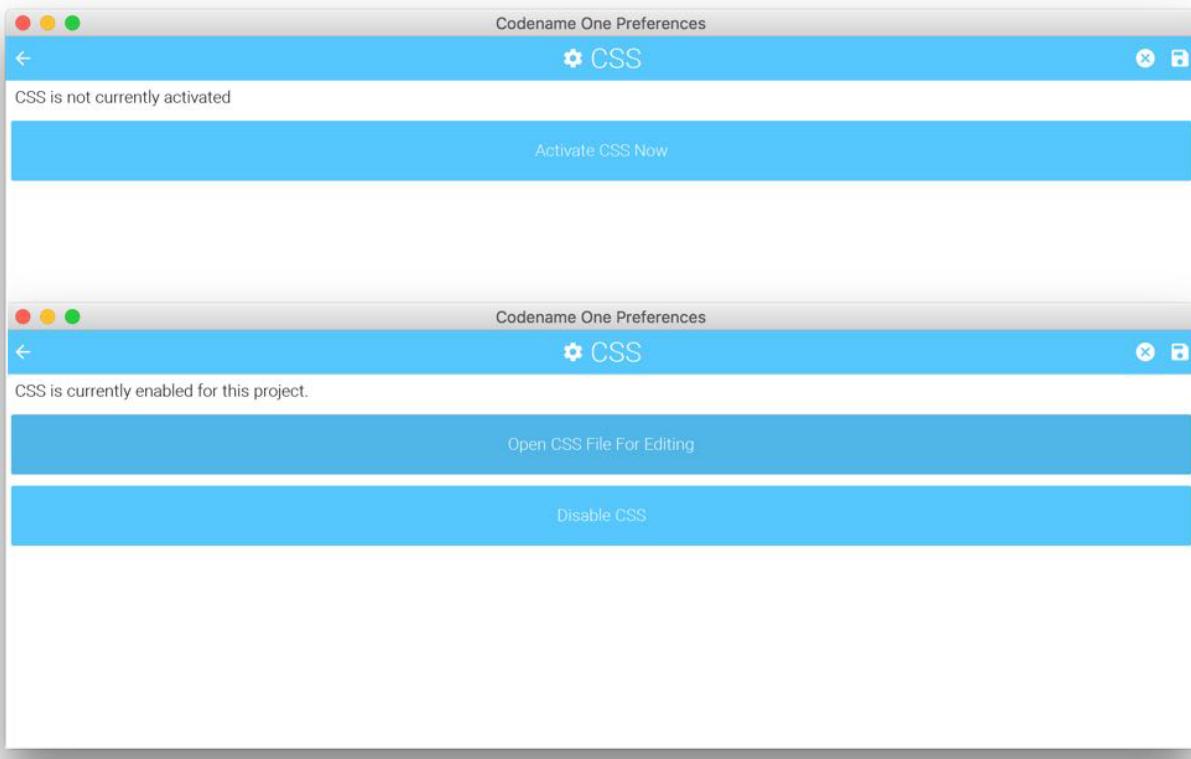


Figure 175. The CSS Option in Codename One Settings Part II

Once enabled your `theme.res` file will regenerate from a CSS file that resides under the `css` directory. Changes you make to the CSS file will instantly update the simulator as you save. However, there are some limits to this live update so in some cases a simulator restart would be necessary.

5.2. Supported CSS Selectors

Since Codename One stylesheets are meant to be used with Codename One component hierarchies instead of XML/HTML documents, selectors work a little differently.

1. All selectors (with some specific exceptions discussed below) are interpreted as UIIDs.
2. Only 4 predefined CSS classes are supported:
 - `.pressed` — Targets the component when in "Pressed" state.
 - `.selected` — Targets the component when in "Selected" state.
 - `.unselected` — Targets the component when in "Unselected" state.
 - `.disabled` — Targets the component when in "Disabled" state.

If no class is specified, then the selector targets "all" states of the given component.

The following are a few possible selectors you can include in your stylesheet.

1. `Button` — Defines styles for the "Button" UIID.

2. `Button.pressed`—Defines styles for the "Button" UIID's "pressed" state.
3. `Button`, `TextField`, `Form`—Defines styles for the "Button", "TextField", and "Form" UIIDs.

The following example creates a simple button with a border, and text aligned center. By default the button will have a transparent background, but when it is pressed, it will have a gray background:

```
Button {  
    text-align: center;  
    border: 1pt solid gray;  
    background-color: transparent;  
}  
  
Button.pressed {  
    background-color: gray;  
}
```

5.2.1. Inheriting properties using `cn1-derive`

The following example defines a custom Button style named "MyButton" that inherits all of the styles of Button but changes the background color to blue.

```
MyButton {  
    cn1-derive: Button;  
    background-color: blue;  
}
```

5.3. Special Selectors

5.3.1. `#Device`

The `#Device` selector allows you to define which device resolutions this CSS file should target. Multi-images generated from this style-sheet will only include variants for device resolutions in the range (`min-resolution`, `max-resolution`) as defined in this section. By default all resolutions are generated.

```
#Device {  
    min-resolution: 120dpi;  
    max-resolution: 480dpi;  
    resolution: 480dpi;  
}
```

5.3.2. `#Constants`

The `#Constants` selector allows you to specify theme constants.

e.g.

```
#Constants {
    PopupDialogArrowBool: false;
    calTitleDayStyleBool: true;
    calTransitionVertBool: false;
    calendarLeftImage: "calendar-arrow-left.png";
    calendarRightImage: "calendar-arrow-right.png";
    centeredPopupBool: false;
    checkBoxCheckDisFocusImage: "Check-Box_Normal.png";
    checkBoxCheckedFocusImage: "Check-Box_Press.png";
    checkBoxCheckedImage: "Check-Box_Press.png";
    checkBoxOppositeSideBool: true;
    checkBoxUncheckedFocusImage: "Check-Box_Normal.png";
    checkBoxUncheckedImage: "Check-Box_Normal.png";
    comboImage: "combo.png";
    commandBehavior: "Side";
    dialogTransitionIn: "fade";
    dialogTransitionOut: "fade";
    dlgButtonCommandUUID: "DialogButton";
    dlgCommandGridBool: true;
    dlgInvisibleButtons: #1a1a1a;
    formTransitionIn: "empty";
    formTransitionOut: "slide";
    includeNativeBool: true;
    menuImage: "of_menu.png";
    noTextModeBool: true;
    onOffIOSModeBool: true;
    otherPopupRendererBool: false;
    pureTouchBool: true;
    radioSelectedFocusImage: "Radio_btn_Press.png";
    radioSelectedImage: "Radio_btn_Press.png";
    radioUnselectedFocusImage: "Radio_btn_Normal.png";
    radioUnselectedImage: "Radio_btn_Normal.png";
    sideMenuImage: "menu.png";
    switchMaskImage: "switch-mask-3.png";
    switchOffImage: "switch-off-3.png";
    switchOnImage: "switch-on-3.png";
    tabPlacementInt: 0;
    backIconImage: "Back-icon.png";
    articleSourceIconImage: "Source-icon.png";
    articleDateIconImage: "Date-icon.png";
    articleArrowRightImage: "Arrow-right.png";
    articleShareIconImage: "Share-icon.png";
    articleBookmarkIconImage: "Bookmark-icon.png";
    articleTextIconImage: "Text-icon.png";
    articleCommentsIconImage: "Comments-icon.png";
    newsIconImage: "News-icon.png";
    channelsIconImage: "Channels-icon.png";
    bookmarksIconImage: "Bookmarks-icon.png";
```

```

    overviewIconImage: "Overview-icon.png";
    calendarIconImage: "Calendar-icon.png";
    timelineIconImage: "Timeline-icon.png";
    profileIconImage: "Profile-icon.png";
    widgetsIconImage: "Widgets-icon.png";
    settingsIconImage: "Settings-icon.png";
    SubmitIconImage: "Submit-icon.png";
    SubmitIconDarkImage: "SubmitButtonLight-icon.png";

}

```

In the above example, the constants referring to an image name as a string requires that the image exists in one of the following locations:

- `res/<cssfilename>/<imageName>`
- `../res/<cssfilename>/<imageName>`
- `../../res/<cssfilename>/<imageName>`

or that it has been defined as a background image in some selector in this CSS file.

5.3.3. Default

The `Default` selector is special in that it will set properties on the theme's "default" element. The default element is a special UIID in Codename One from which all other UIIDs in the same theme are derived. This is a good place to set things like default fonts or background-colors.

5.4. Standard CSS Properties

- `padding` (and variants)
- `margin` (and variants)
- `border` (and variants)
- `border-radius`
- `background` (Usage [below](#))
- `background-color`
- `background-repeat`
- `background-image`
- `border-image`
- `border-image-slice`
- `font` (Usage is covered in the following font section)
- `font-family` (Usage is covered in the following font section)
- `font-style` (Usage is covered in the following font section)
- `font-size` (Usage is covered in the following font section)

- `@font-face` (Usage is covered in the following font section)
- `color`
- `text-align`
- `text-decoration`(Usage [below](#))
- `opacity`
- `box-shadow`
- `width` (only used for generating background-images and borders)
- `height` (only used for generating background-images and borders)

5.5. Custom Properties

- `cn1-source-dpi` — Used to specify source DPI for multi-image generation of background images. Accepted values: `0` (Not multi-image), `120` (Low res), `160` (Medium Res), `320` (Very High Res), `480` (HD), Higher than `480` (2HD)
- `cn1-background-type` — Used to explicitly specify the background-type that should be used for the class.
- `cn1-9patch` — Used to explicitly specify the slices used when generating 9-piece borders.
Deprecated - Use `border-image` and `border-image-slice` for 9-piece borders.
- `cn1-derive` — Used to specify that this UIID should derive from an existing UIID.

5.6. CSS Variables

As of CodenameOne 7.0, you can use variables in your CSS file via the `var()` CSS function. E.g.

```
var(--header-color, blue);
```

The `var()` function can only be used inside property values. I.e. You cannot use it in property names or selectors.

Syntax:

```
var(<custom-property-name>, <declaration-value>?)
```

The `<custom-property-name>` must begin with two dashes (`--`).

The `<declaration-value>` is the fallback value that will be used if the variable hasn't been defined in the CSS file. The fallback value may include commas.

Examples

Listing 13. Example defining and using a CSS variable

```
#Constants {  
    --main-bg-color: red;  
}  
  
MyContainer {  
    background-color: var(--main-bg-color);  
}
```

Listing 14. Example using a fallback value

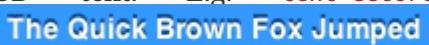
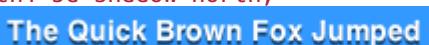
```
#Constants {  
    --main-bg-color: red;  
}  
  
MyContainer {  
    background-color: var(--main-bg-color, blue);  
}
```

See the [MDN docs](https://developer.mozilla.org/en-US/docs/Web/CSS/var) [https://developer.mozilla.org/en-US/docs/Web/CSS/var] for more details about the CSS variable spec.

5.7. CSS Properties

This section isn't as comprehensive as it should be due to the breadth of CSS.

5.7.1. text-decoration

underline	Underlines text. E.g. <code>text-decoration: underline;</code>
overline	Overlines text. E.g. <code>text-decoration: overline;</code>
line-through	Strikes through text. E.g. <code>text-decoration: line-through;</code>
none	No text decoration. E.g. <code>text-decoration: none;</code>
cn1-3d	3D text. E.g. <code>text-decoration: cn1-3d;</code> 
cn1-3d-lowered	3D lowered text. E.g. <code>text-decoration: cn1-3d-lowered;</code> 
cn1-3d-shadow-north	3D text with north shadow. E.g. <code>text-decoration: cn1-3d-shadow-north;</code> 

For other CSS font settings see [the Fonts section](#) [Fonts]

5.7.2. border

This library supports the [border property](#) [<https://developer.mozilla.org/en-US/docs/Web/CSS/border>] and most of its variants (e.g. [border-width](#) [<https://developer.mozilla.org/en-US/docs/Web/CSS/border-width>], [border-style](#) [<https://developer.mozilla.org/en-US/docs/Web/CSS/border-style>], and [border-color](#) [<https://developer.mozilla.org/en-US/docs/Web/CSS/border-color>]). It will try to use native Codename One styles for generating borders if possible. If the border definition is too complex, it will fall-back to generating a 9-piece image border at compile-time. This has the effect of making the resulting resource file larger, but will produce good runtime performance, and a look that is faithful to the provided CSS.

The algorithm used to determine whether to use a native border or to generate a 9-piece image, is complex, but the following guidelines may help you if you wish to design borders that can be rendered natively in CN1:

- Non-pixel units [border-width](#). (Except with the [cn1-round-border](#) and [cn1-pill-border](#) styles)
- Using the [border-radius](#) directive.
- Using [box-shadow](#) (Except when using [cn1-round-border](#) or [cn1-pill-border](#) styles)
- Using a background gradient in combination with a border or any kind
- Using a different [border-width](#), [border-style](#), or [border-color](#) for different sides of the border
- Using a [filter](#)



You can open the resulting theme file in the designer and inspect it to see if an image was generated

Generating the image triggers slower CSS compilation and a larger binary so we generally recommend tuning the CSS so it avoids this fallback.

Round Borders

Rounded borders can be achieved in a few different ways. The easiest methods are:

- **The [cn1-round-border](#) style.** This will render a circular round border in the background **natively**. I.e. this doesn't require generation of an image border
- **The [cn1-pill-border](#) style.** This will render a pill-shaped border in the background **natively**. This also doesn't require generation of an image border
- **The [border-radius](#) property.** This will round the corners of the border. If the style can be achieved using the [RoundRectBorder](#) in CodenameOne, then it will use that border. If not, this will cause the style to be generated as an image border

Examples using [cn1-round-border](#)

```

RoundBorder {
    border: 1px #3399ff cn1-round-border;
    text-align:center;
    margin:2mm;
    padding:3mm;
}

RoundBorderFilled {
    background: cn1-round-border;
    background-color: #ccc;
    text-align:center;
    margin:2mm;
    padding:3mm;
}

```

Examples using `cn1-pill-border`

```

PillBorder {
    border: 1pt #3399ff cn1-pill-border;
    text-align:center;
}

PillBorderFilled {
    background: cn1-pill-border;
    background-color: #3399ff;
    color:white;
    text-align:center;
}

```

Examples using `border-radius`

```

RoundRectLabel {
    background-color: red;
    border-radius: 2mm;
}

```

`cn1-pill-border` and `cn1-round-border` don't support the standard CSS `box-shadow` property. This is because the `box-shadow` property parameters don't map nicely onto the shadow parameters for the Codename One `RoundBorder` class. To get shadows on the `cn1-pill-border`, you should use one or more of the following CSS properties:

- `cn1-box-shadow-spread`—Accepts values in any scalar unit (e.g. px, mm, cm, etc...). This maps directly to the border's `shadowSpread` [[https://www.codenameone.com/javadoc/com/codename1/ui/plaf/RoundBorder.html#shadowSpread-int-boolean-\]](https://www.codenameone.com/javadoc/com/codename1/ui/plaf/RoundBorder.html#shadowSpread-int-boolean-) property.
- `cn1-box-shadow-h`—Accepts values in real values or integers (not a scalar unit). This maps directly to the border's `shadowX` [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/RoundBorder.html#shadowX-real->] property.

`RoundBorder.html#shadowX-float]` property.

- `cn1-box-shadow-v` — Accepts values in real values or integers (not a scalar unit). This maps directly to the border's `shadowY` [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/RoundBorder.html#shadowY-float>] property.
- `cn1-box-shadow-blur` — Scalar value. Maps to the border's `shadowBlur` [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/RoundBorder.html#shadowBlur-float>] property.
- `cn1-box-shadow-color` — The shadow color

Currently using the regular CSS `box-shadow` in conjunction with `border-radius` will cause a 9-piece border to be generated rather than mapping to the `RoundRectBorder`. If, however, you use the `cn1-box-*` properties for the shadow instead, it will use the `RoundRectBorder` — assuming that no other styles are specified that trigger an image border to be generated.

5.7.3. background

The `background` property supports most standard [CSS values](https://developer.mozilla.org/en-US/docs/Web/CSS/background) [<https://developer.mozilla.org/en-US/docs/Web/CSS/background>] for setting the background color, or background image.



9-piece Image borders always take precedence over background settings in Codename One. If your background directive seems to have no effect, it is likely because the theme has specified a 9-piece image border for the UIID. You can disable the image border using a directive like `border: none`

Background Images

See [Images](#) [Images]

Gradients

Both the `linear-gradient` and `radial-gradient` CSS functions are fully supported by this library. If Codename One is capable of rendering the gradient natively then the theme resource file generated will only include encoded parameters for the gradients. If the gradient is not supported by Codename One, then the module will fall back to an image background which it generates at compile-time. It is generally preferable to try to stick to gradients that Codename One supports natively. This will result in a smaller theme resource file since it doesn't need to generate any images for the gradient.

Natively Supported `linear-gradient` Syntax

In order for a linear gradient to be natively supported by Codename One, the following properties must be met:

1. The gradient function has exactly two color stops, and these colors have the same opacity.
2. The gradient is either perfectly horizontal or perfectly vertical. (e.g Direction can be `0deg`, `90deg`, `180deg`, or `270deg`.)

Examples

```
MyContainer {  
    background: linear-gradient(0deg, #ccc, #666);  
}
```

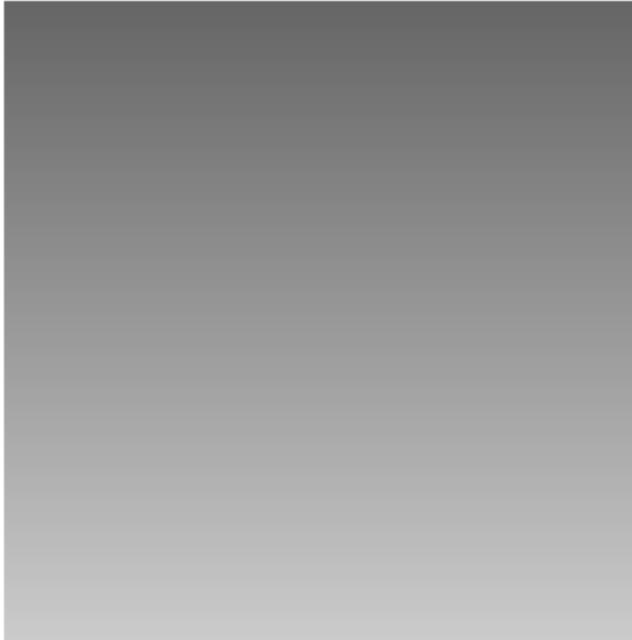


Figure 176. Native linear gradient 0 degrees

```
MyContainer {  
    background: linear-gradient(to top, #ccc, #666);  
}
```

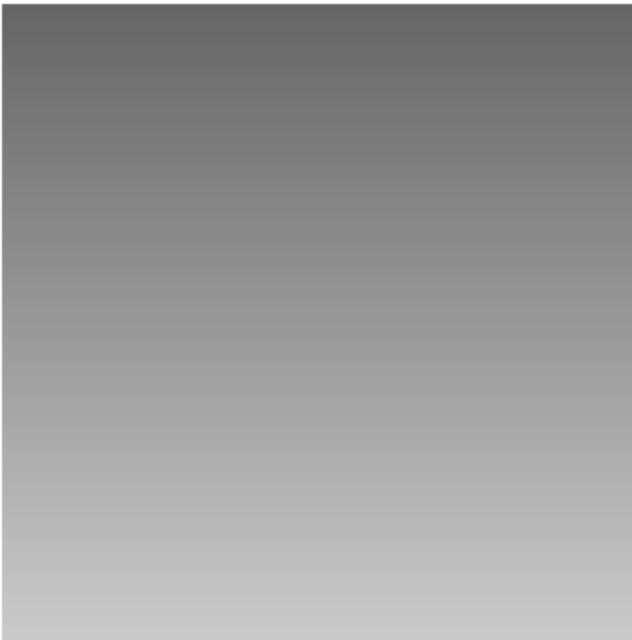


Figure 177. Native linear gradient to top

```
MyContainer {  
    background: linear-gradient(90deg, #ccc, #666);  
}
```

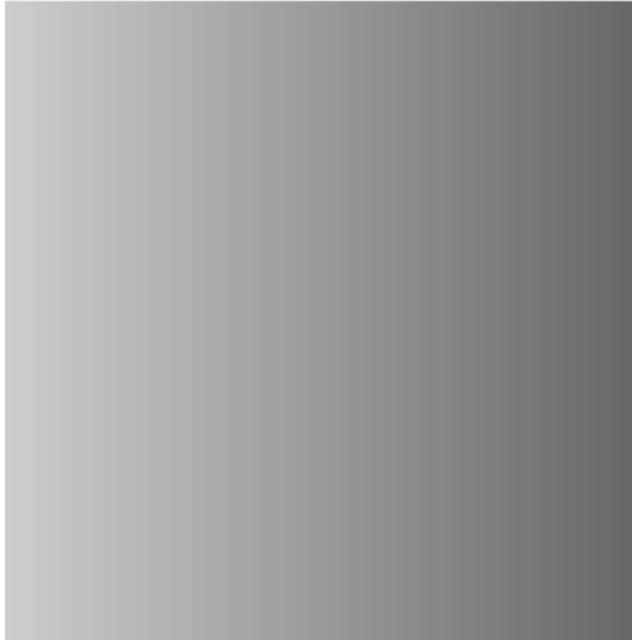


Figure 178. Native linear gradient 90deg

```
MyContainer {  
    background: linear-gradient(to left, #ccc, #666);  
}
```

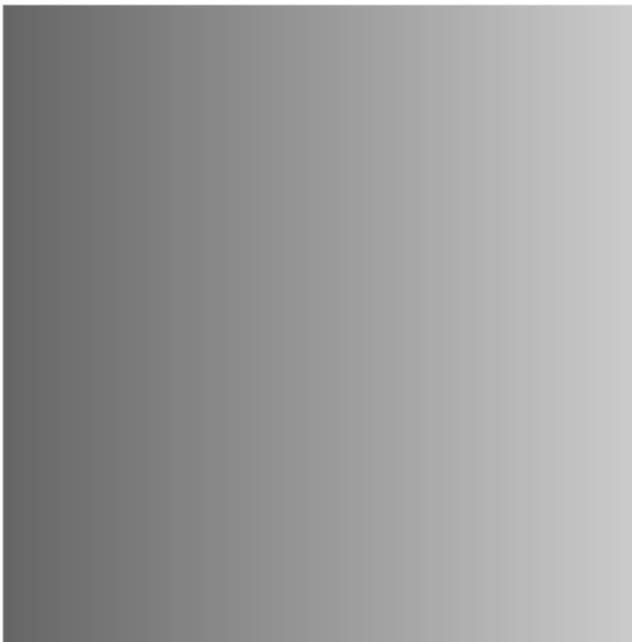


Figure 179. Native linear gradient to left

Unsupported **linear-gradient** syntax

The following are some examples of linear gradients that aren't supported natively by Codename

One, and will result in a background image to be generated at compile-time:

```
MyComponent {  
    background: linear-gradient(45deg, #eaeaea, #666666);  
}
```



Figure 180. 45deg gradient rendered at compile-time — uses background image

The above example is not supported natively because the gradient direction is 45 degrees. Codename One only supports 0, 90, 180, and 270 degrees natively. Therefore this would result in a background image being generated at compile-time with the appropriate gradient.

```
MyComponent {  
    background: linear-gradient(90deg, rgba(255, 0, 0, 0.6), blue);  
}
```

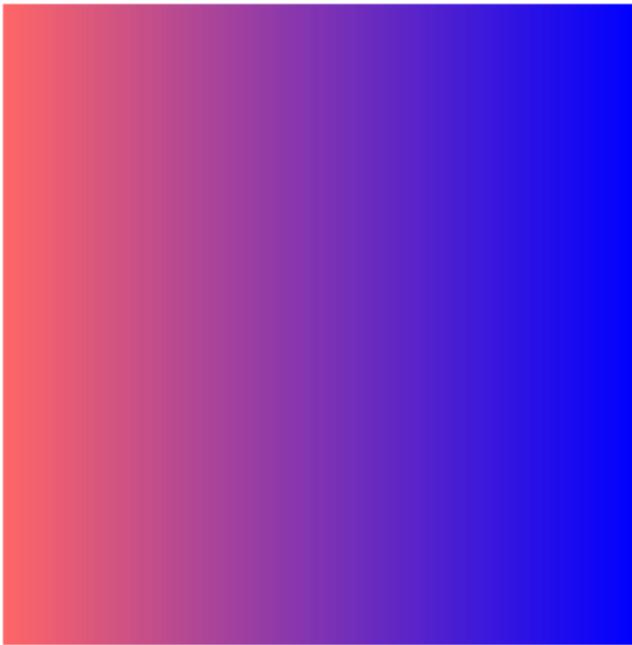


Figure 181. Linear gradient with different alpha

The above linear-gradient is not supported natively because the stop colors have different transparencies. The first color has an opacity of 0.5, and the second as an opacity of 1.0 (implicitly). Therefore, this would result in the gradient being generated as an image at compile-time.

Natively Supported **radial-gradient** Syntax

The following syntax is supported natively for radial gradients. Other syntaxes are also supported by the CSS library, but they will use compile-time image generation for the gradients rather than generating them at runtime.

```
background: radial-gradient(circle [closest-side] [at <position>], <color stop>, <color stop>)
```

- **<position>**—The position using either offset keywords or percentages.
- **<color stop>**—Either a color alone, or a color followed by a percentage. 0% indicates that color begins at center of the circle. 100% indicates that the color begins at the closest edge of the bounding box. Higher/lower values (>0%) will shift the color further or closer to circle's center. If the first color stop is set to a non-zero value, the gradient cannot be rendered natively by Codename One, and an image of the gradient will instead be generated at compile-time.

More complex gradients are supported by this library, but they will be generated at compile-time. For more information about the **radial-gradient** CSS function see its MDN Wiki page [<https://developer.mozilla.org/en-US/docs/Web/CSS/radial-gradient>].

Examples

```
MyContainer {  
    background: radial-gradient(circle, gray, white);  
}
```

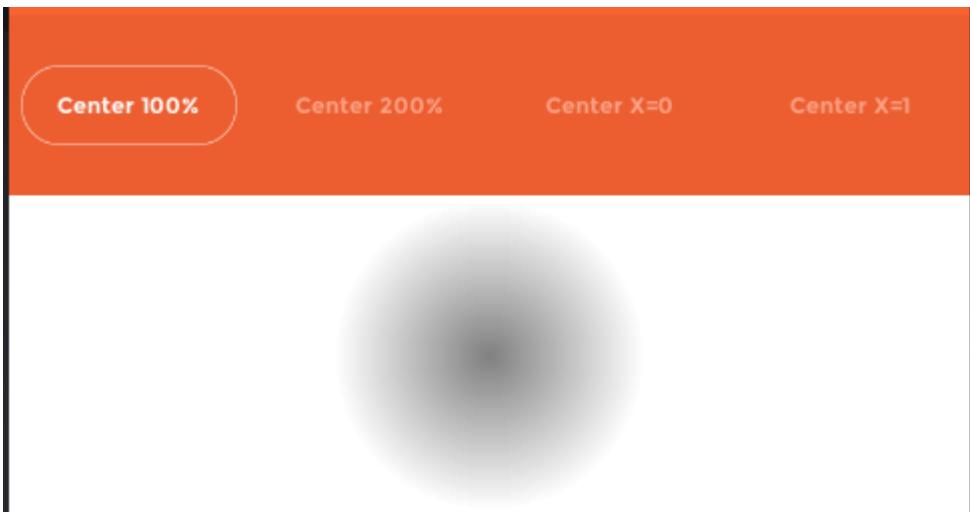


Figure 182. Radial gradient 0 to 100

```
MyContainer {  
    background: radial-gradient(circle, gray, white 200%);  
}
```

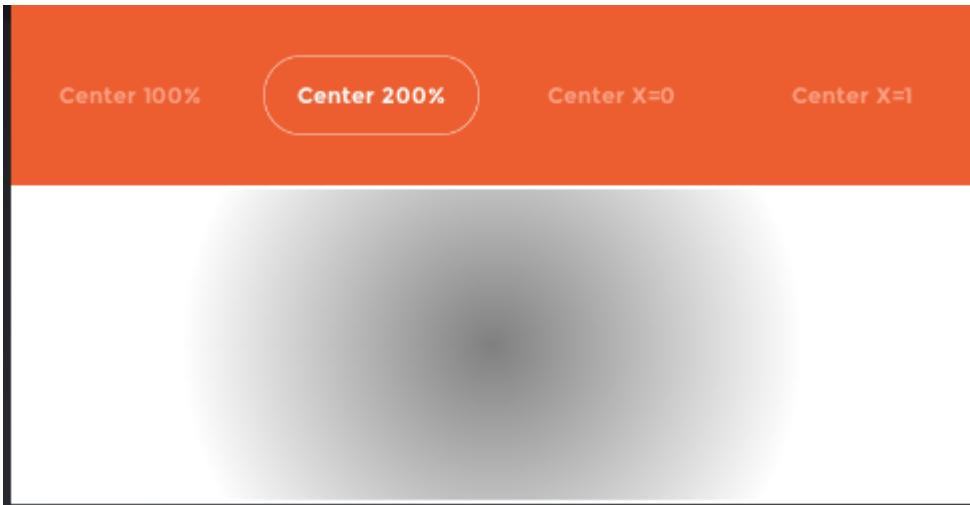


Figure 183. Radial gradient 0 to 200

```
MyContainer {  
    background: radial-gradient(circle at left, gray, white);  
}
```

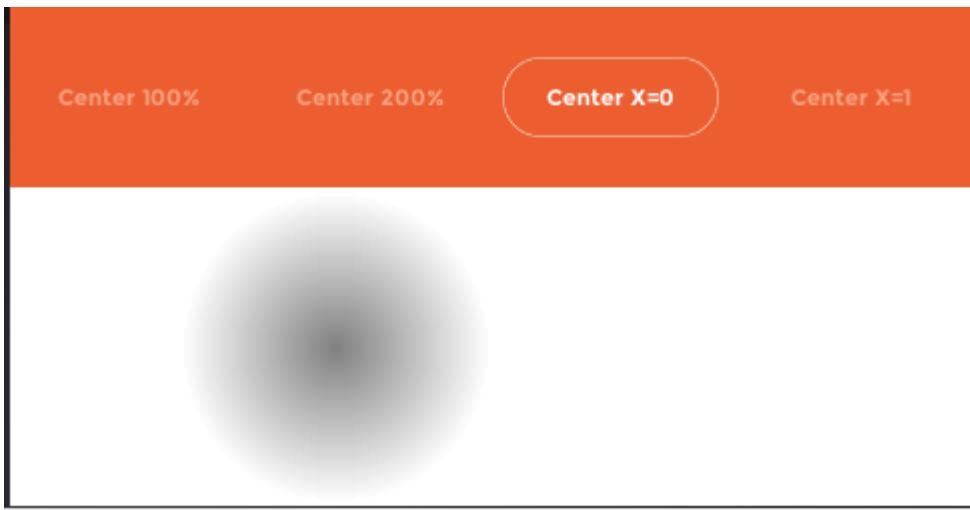


Figure 184. Radial gradient at left

```
MyContainer {  
    background: radial-gradient(circle at right, gray, white);  
}
```

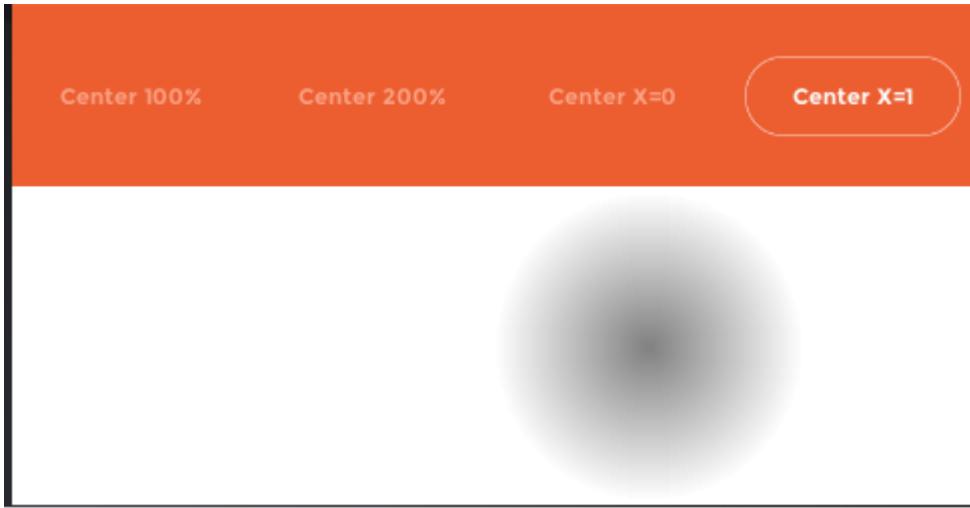


Figure 185. Radial gradient at right

5.7.4. cn1-background-type

It also supports some special Codename One values, which are identifiers with a "cn1-" prefix. The following special values are available. They map to the standard Codename One values we discussed in the theming chapter:

- `cn1-image-scaled`
- `cn1-image-scaled-fill`
- `cn1-image-scaled-fit`
- `cn1-image-tile-both`
- `cn1-image-tile-valign-left`
- `cn1-image-tile-valign-center`

- `cn1-image-tile-valign-right`
- `cn1-image-tile-halign-top`
- `cn1-image-tile-halign-center`
- `cn1-image-tile-halign-bottom`
- `cn1-image-align-bottom`
- `cn1-image-align-left`
- `cn1-image-align-right`
- `cn1-image-align-center`
- `cn1-image-align-top-left`
- `cn1-image-align-top-right`
- `cn1-image-align-bottom-left`
- `cn1-image-align-bottom-right`
- `cn1-image-border`
- `cn1-none`
- `cn1-round-border`
- `cn1-pill-border`

5.8. Images

Images are supported as both "inputs" of the stylesheet, and as outputs to the compiled resource file. "Input" images are usually specified via the `background-image` property in a selector. "Output" images are always saved as multi-images inside the resource file.

5.8.1. Image DPI and Device Densities

In order to appropriately size the image, the CSS compiler needs to know what the source density of the image is. E.g. if an image is 160x160 pixels with a source density of 160dpi (i.e. medium density - or the same as an iPhone 3G), then the resulting multi-image will be sized at 160x160 for medium density devices and 320x320 on very high density devices (e.g. iPhone 4S Retina) - which will result in the same perceived size to the user of 1x1 inch.

However if the image has a source density of 320dpi, then the resulting multi-image would be 80x80 pixels on medium density devices and 160x160 pixels on very high density devices.

Some images have this density information embedded in the image itself so that the CSS processor will know how to resize the image properly. However, it is usually better to explicitly document your intentions by including the `cn1-source-dpi` property as follows:

```
SomeStyle {  
    background-image: url(images/my-image.png);  
    cn1-source-dpi: 160;  
}
```



`cn1-source-dpi` values are meant to fall into threshold ranges. Values less than or equal to 120, are interpreted as low density. 121 - 160 are medium density (iPhone 3GS). 161 - 320, very high density (iPhone 4S). 321 - 480 == HD. 481 and higher == 2HD. In general, you should try to use images that are one of these DPIs exactly: 160, 320, or 480, then images will be scaled up or down to the other densities accordingly.

5.8.2. Multi-Images vs Regular Images

By default all images are imported as multi-images. If you want to import an image as a "regular" image, you can simply set `cn1-source-dpi` to `0`. E.g.

```
SomeStyle {  
    background-image: url(images/my-image.png);  
    cn1-source-dpi: 0;  
}
```

5.8.3. Multi-Images as Inputs

If you have already generated images in all of the appropriate sizes for all densities, you can provide them in the same file structure used by the Codename One XML resource files: The image path is a directory that contains images named after the density that they are intended to be used for. The possible names include:

- `verylow.png`
- `low.png`
- `medium.png`
- `high.png`
- `veryhigh.png`
- `560.png`
- `hd.png`
- `2hd.png`
- `4k.png`

E.g. Given the CSS directives:

```
MyStyle {  
    background-image: url(images/mymultiimage.png);  
}
```

The files would look like:

```
css/  
+--- mycssfile.css  
+--- images/  
    +--- mymultiimage.png/  
        +--- verylow.png  
        +--- low.png  
        +--- medium.png  
        ... etc...
```



Multi-image inputs are only supported for local URLs. You cannot use remote (e.g. <http://>) urls with multi-image inputs

5.8.4. Image Constants

Theme constants can be images. The convention is to suffix the constant name with "Image" so that it will be treated as an image. In addition to the standard `url()` notation for specifying a constant image, you can provide a simple string name of the image, and the CSS processor will try to find an image by that name specified as a background image for one of the styles. If it cannot find one, it will look inside a special directory named "res" (located in the same directory as the CSS stylesheet), inside which it will look for a directory named the same as the stylesheet, inside which it will look for a directory with the specified multi-image. This directory structure is the same as used for Codename One's XML resources directory.

E.g. In the CSS file "mycssfile.css":

```
radioSelectedFocusImage: "Radio_btn_Press.png";
```

Will look for a directory located at `res/mycssfile.css/Radio_btn_Press.png/` with the following images:

- `verylow.png`
- `low.png`
- `medium.png`
- `high.png`
- `veryhigh.png`
- `560.png`
- `hd.png`

- 2hd.png
- 4k.png

It will then create a multi-image from these images and include them in the resource file.

5.9. Image Recipes

5.9.1. Import Multiple Images In Single Selector

It is quite useful to be able to embed images inside the resource file that is generated from the CSS stylesheet so that you can access the images using the `Resources.getImage()` method in your app and set it as an icon on a button or label. In this case, it is easier to simply create a dummy style that you don't intend to use and include multiple images in the background-image property like so:

```
Images {
    background-image: url(images/NowLogo.png),
    url(images/Username-icon.png),
    url(images>Password-icon.png),
    url(images/Name-icon.png),
    url(images/Email-icon.png),
    url(images/SeaIce.png),
    url(images/Back-icon.png),
    url(images/Source-icon.png),
    url(images/Date-icon.png),
    url(images/Arrow-right.png),
    url(images/Share-icon.png),
    url(images/Text-icon.png),
    url(images/Comments-icon.png),
    url(images/RedPlanet.png),
    url(images/News-icon.png),
    url(images/Channels-icon.png),
    url(images/Bookmarks-icon.png),
    url(images/Overview-icon.png),
    url(images/Calendar-icon.png),
    url(images/Timeline-icon.png),
    url(images/Profile-icon.png),
    url(images/Widgets-icon.png),
    url(images/Settings-icon.png),
    url(images/Bookmark-icon.png);
}
```

Then in Java, I might do something like:

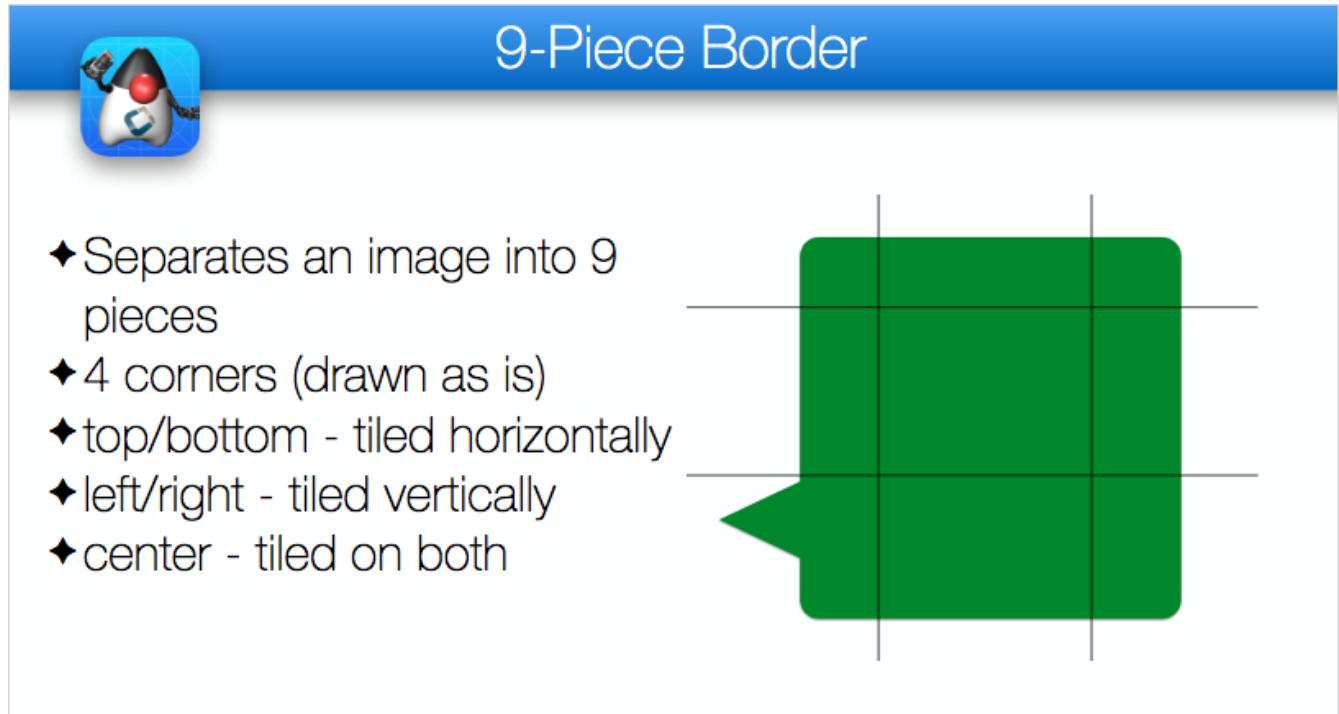
```
Resources theme = Resources.openLayered("/theme.css.res");
Label bookmark = new Label(theme.getImage("Bookmark-icon.png"));
```

5.9.2. Loading Images from URLs

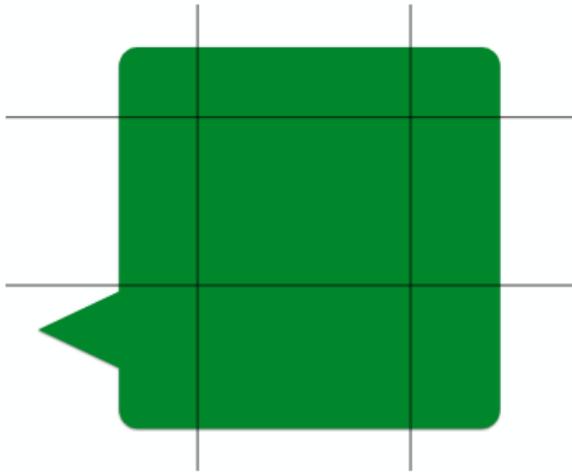
You can also load images from remote URLs. E.g.

```
Images {  
    background-image: url(http://solutions.weblite.ca/logo.png);  
}
```

5.9.3. Generating 9-Piece Image Borders



- ◆ Separates an image into 9 pieces
- ◆ 4 corners (drawn as is)
- ◆ top/bottom - tiled horizontally
- ◆ left/right - tiled vertically
- ◆ center - tiled on both



9-Piece image borders can be created using the `image-border` and `image-border-slice` properties.

E.g.

```
NinePiece {  
    border-image: url('dashbg_landscape.png');  
}
```

In the above example we omitted the `border-image-slice` property, so it defaults to "40%", which means that the image is sliced 40% from the top, 40% from the bottom, 40% from the left, and 40% from the right.

If you want more specific "slice" points, you can add the `border-image-slice` property. E.g.

```

NinePiece {
    border-image:url('dashbg_landscape.png');
    border-image-slice:10% 49%; /*vertical horizontal*/
}

NinePiece2 {
    border-image:url('dashbg_landscape.png');
    border-image-slice:10% 49% 20%; /*top horizontal bottom*/
}

NinePiece3 {
    border-image:url('dashbg_landscape.png');
    border-image-slice:10% 30% 40% 20%; /*top right bottom left*/
}

NinePiece4 {
    border-image:url('dashbg_landscape.png');
    border-image-slice:10%; /*all*/
}

```

5.9.4. Image Backgrounds

Component backgrounds in Codename One are a common source of confusion for newcomers because there are 3 different properties that can be used to define what a component's background looks like, and they have priorities:

1. Background Color - You can specify an RGB color to be used as the background for a component.
2. Background Image - You can specify an image to be used as the background for a component. Codename One includes settings to define how the image is treated, e.g. scale/fill, tile, etc. If a background image is specified, it will override the background color setting - unless the image has transparent regions.
3. Image Border - You can define a 9-piece image border which will effectively cover the entire background of the component. If an image border is specified, it will override the background image of the component.

A common scenario that I run into is trying to set the background color of a component and see no change when I preview my form because the style had an image background defined - which overrides my background color change.

The potential for confusion is mitigated somewhat, but still exists when using CSS. You can make your intentions explicit by adding the `cn1-background-type` property to your style. Possible values include:

- `cn1-image-scaled`
- `cn1-image-scaled-fill`
- `cn1-image-scaled-fit`
- `cn1-image-tile-both`

- `cn1-image-tile-valign-left`
- `cn1-image-tile-valign-center`
- `cn1-image-tile-valign-right`
- `cn1-image-tile-halign-top`
- `cn1-image-tile-halign-center`
- `cn1-image-tile-halign-bottom`
- `cn1-image-align-bottom`
- `cn1-image-align-left`
- `cn1-image-align-right`
- `cn1-image-align-center`
- `cn1-image-align-top-left`
- `cn1-image-align-top-right`
- `cn1-image-align-bottom-left`
- `cn1-image-align-bottom-right`
- `cn1-image-border`
- `cn1-none`
- `none`

Example Setting Background Image to Scale Fill

```
MyContainer {
    background-image: url(myimage.png);
    cn1-background-type: cn1-image-scaled-fill;
}
```

5.10. Image Compression

CN1 resource files support both PNG and JPEG images, but PNG is the default. Multi-images that are generated by the CSS compiler will be PNG if they include alpha transparency, and JPEG otherwise. This is to try to reduce the file size as much as possible while not sacrificing quality.

5.11. Fonts

This library supports the [font](https://developer.mozilla.org/en/docs/Web/CSS/font), [font-size](https://developer.mozilla.org/en/docs/Web/CSS/font-size), [font-family](https://developer.mozilla.org/en/docs/Web/CSS/font-family), [font-style](https://developer.mozilla.org/en/docs/Web/CSS/font-family), [font-weight](https://developer.mozilla.org/en/docs/Web/CSS/font-weight), and [text-decoration](https://developer.mozilla.org/en/docs/Web/CSS/text-decoration) properties, as well at the [@font-face](https://developer.mozilla.org/en/docs/Web/CSS/@font-face) CSS "at" rule for including TTF/OTF fonts.

5.11.1. font-family

By default, [CN1's native fonts](https://www.codenameone.com/blog/good-looking-by-default-native-fonts-simulator-detection-more.html) [https://www.codenameone.com/blog/good-looking-by-default-native-fonts-simulator-detection-more.html] are used. The appropriate native font is selected for the provided font-weight and font-style properties. You can also explicitly specify the native font you wish to use in the `font-family` property. E.g.

```
SideCommand {  
    font-family: "native:MainThin";  
}
```

If you omit the `font-family` directive altogether, it will use `native:MainRegular`. The following native fonts are available:

1. `native:MainThin`
2. `native:MainLight`
3. `native:MainRegular`
4. `native:MainBold`
5. `native:MainBlack`
6. `native:ItalicThin`
7. `native:ItalicLight`
8. `native:ItalicRegular`
9. `native:ItalicBold`
10. `native:ItalicBlack`

Using TTF Fonts

If you want to use a font other than the built-in fonts, you'll need to define the font using the `@font-face` rule. E.g.

```
@font-face {  
    font-family: "Montserrat";  
    src: url(res/Montserrat-Regular.ttf);  
}
```

Then you'll be able to reference the font using the specified `font-family` in any CSS element. E.g.

```
MyLabel {  
    font-family: "Montserrat";  
}
```

The `@font-face` directive's `src` property will accept both local and remote URLs. E.g.

```
@font-face {  
    font-family: "MyFont";  
    src: url(http://example.com/path/to/myfont.ttf);  
}
```

In this case, it will download the `myfont.ttf` file to the same directory as the CSS file. From then on it will use that locally downloaded version of the font so that it doesn't have to make a network request for each build.

Fonts are automatically copied to the project's "src" directory when the CSS file is compiled so that they will be distributed with the app and available at runtime.

Github URLs

Fonts hosted on GitHub are accessible using a special `github:` protocol to make it easier to reference such fonts. E.g. the following directive includes the "FontAwesome" font directly from Github

```
@font-face {  
    font-family: "FontAwesome";  
    src: url(github://FontAwesome/Font-Awesome/blob/master/fonts/fontawesome-webfont.ttf);  
}
```



Apparently FontAwesome has removed its public repositories from Github so this URL no longer works.

5.11.2. `font-size`

It is best practice to size your fonts using millimetres (`mm`) (or another "real-world" measurement unit such as inches (`in`), centimetres (`cm`)). This will allow the font to be sized appropriate for all display densities. If you specify size in pixels (`px`), it will treat it the same as if you sized it in points (`pt`), where $1\text{pt} = 1/72 \text{ inches}$ (one seventy-second of an inch).

If you size your font in percentage units (e.g. `150%`) it will set the font size relative to the medium font size of the platform. This is different than the standard behaviour of a web browser, which would size it relative to the parent element's font size.

You can use system fonts, true type fonts, and native fonts in your CSS stylesheet. True Type fonts need to be defined in a `@font-face` directive before they can be referenced. True-type fonts and native fonts have the advantage that you can specify their sizes in generic terms (e.g. `small`, `medium`, `large`) and in more specific units such as millimeters (`mm`) or pixels (`px`).

5.11.3. `text-decoration`

See [the text-decoration section](#) [Supported-Properties#text-decoration] in the "Supported Properties" page.

5.11.4. Some Sample CSS Directives

```
@font-face {  
    font-family: "Montserrat";  
    src: url(res/Montserrat-Regular.ttf);  
}  
  
@font-face {  
    font-family: "Montserrat-Bold";  
    src: url(res/Montserrat-Bold.ttf);  
}  
  
@font-face {  
    font-family: "FontAwesome";  
    src: url(github://FontAwesome/Font-Awesome/blob/master/fonts/fontawesome-webfont.ttf);  
}  
  
PlainText0p5mm {  
    font-size: 0.5mm;  
}  
  
PlainText1mm {  
    font-size: 1mm;  
}  
  
PlainText2mm {  
    font-size: 2mm;  
}  
  
PlainText5mm {  
    font-size: 5mm;  
}  
  
PlainText10mm {  
    font-size: 10mm;  
}  
  
PlainText50mm {  
    font-size: 50mm;  
}  
  
PlainTextSmall {  
    font-size: small;  
}  
  
PlainTextMedium {  
    font-size: medium;  
}  
  
PlainTextLarge {  
    font-size: large;  
}
```

```
PlainText3pt {
    font-size: 3pt;
}

PlainText6pt {
    font-size: 6pt;
}

PlainText12pt {
    font-size: 12pt;
}

PlainText20pt {
    font-size: 20pt;
}

PlainText36pt {
    font-size: 36pt;
}

BoldText {
    font-weight: bold;
}

BoldText1mm {
    font-weight: bold;
    font-size: 1mm;
}

BoldText2mm {
    font-weight: bold;
    font-size: 2mm;
}

BoldText3mm {
    font-weight: bold;
    font-size: 3mm;
}

BoldText5mm {
    font-weight: bold;
    font-size: 5mm;
}

ItalicText {
    font-style: italic;
}

ItalicText3mm {
    font-style: italic;
    font-size: 3mm;
}
```

```
ItalicBoldText {  
    font-style: italic;  
    font-weight: bold;  
}  
  
PlainTextUnderline {  
    text-decoration: underline;  
}  
  
BoldTextUnderline {  
    text-decoration: underline;  
    font-weight: bold;  
}  
  
ItalicTextUnderline {  
    text-decoration: underline;  
    font-style: italic;  
}  
  
PlainText3d {  
    text-decoration: cn1-3d;  
    color:white;  
    background-color: #3399ff  
}  
  
BoldText3d {  
    text-decoration: cn1-3d;  
    font-weight: bold;  
    color:white;  
    background-color: #3399ff;  
}  
  
ItalicText3d {  
    text-decoration: cn1-3d;  
    font-style: italic;  
    color:white;  
    background-color: #3399ff;  
}  
  
PlainText3dLowered {  
    text-decoration: cn1-3d-lowered;  
    color:black;  
    background-color: #3399ff;  
}  
  
BoldText3dLowered {  
    text-decoration: cn1-3d-lowered;  
    font-weight: bold;  
    color:black;  
    background-color: #3399ff;  
}
```

```

ItalicText3dLowered {
    text-decoration: cn1-3d-lowered;
    font-style: italic;
    color:black;
    background-color: #3399ff;
}

PlainText3dShadow {
    text-decoration: cn1-3d-shadow-north;
    color:white;
    background-color: #3399ff;
}

BoldText3dShadow {
    text-decoration: cn1-3d-shadow-north;
    font-weight: bold;
    color:white;
    background-color: #3399ff;
}

ItalicText3dShadow {
    text-decoration: cn1-3d-shadow-north;
    font-style: italic;
    color:white;
    background-color: #3399ff;
}

MainThin {

    font-size: 200%;
    background: radial-gradient(circle at top left, yellow, blue 100%);
}

MainRegular0001 {
    font-family: "native:MainRegular";
    /*background: cn1-pill-border;
    background-color: red;*/
    color: blue;
    border: 1px cn1-pill-border blue;
    /*box-shadow: 1mm 1mm 0 2mm rgba(0,0,0,1.0);*/
    padding: 2mm;
}

MainRegular0001.pressed {
    font-family: "native:MainRegular";
    background: cn1-pill-border blue;
    /*background-color: red;*/
    color: white;
    border: 1px solid white;
    /*box-shadow: 1mm 1mm 0 2mm rgba(0,0,0,1.0);*/
    padding: 2mm;
}

```

```

Heading {
    font-size: 4mm;
    font-family: "Montserrat-Bold";
    color: black;
    padding: 2mm;
    text-align: center;
}

XMLViewIcon {
    font-family: "FontAwesome";
}

```

5.12. Media Queries

You can use media queries to target styles to specific platforms, devices, and device densities. Currently the following media queries are supported:

1. **platform-xxx** - Target a specific platform. E.g. **platform-and**, **platform-ios**, **platform-mac**, **platform-win**.
2. **density-xxx** - Target a specific device density. E.g. **density-very-low**, **density-low**, **density-medium**, **density-high**, **density-very-high**, **density-hd**, **density-2hd**, and **density-560**.
3. **device-xxx** - Target a specific device type. E.g. **device-desktop**, **device-tablet**, **device-phone**.

Listing 15. Example: Different font colors on Android and iOS. On Android, labels will appear green. On iOS, they will appear red. On all other platforms, they will appear black.

```

Label {
    color: black;
}

@media platform-and {
    Label {
        color: green;
    }
}

@media platform-ios {
    Label {
        color: red;
    }
}

```

Listing 16. Example: Different font colors based on device density. On lower densities, labels will be green. On higher densities, labels will be red.

```
Label {  
    color: black;  
}  
  
@media density-very-low, density-low, density-medium, density-high {  
    Label {  
        color: green;  
    }  
}  
  
@media density-very-high, density-h2, density-2hd, density-560 {  
    Label {  
        color: red;  
    }  
}
```

Listing 17. Example: Different label colors based on device type.

```
Label {  
    color: black;  
}  
  
@media device-desktop {  
    Label {  
        color: green;  
    }  
}  
  
@media device-tablet, device-phone {  
    Label {  
        color: red;  
    }  
}
```



When deploying your app using the Javascript port, it will use a platform name derived from the "UserAgent" string in the browser, rather than the result of `Display.getPlatformName()`, which is used for other ports. When running on Android, then, the platform will be "and". When running on iOS, the platform will be "ios". Etc...

5.12.1. Compound Media Queries

You can combine multiple media queries together, separated by a comma. Queries of the same type are "OR"ed together. Queries of different types are "AND"ed together. For example if you have a media query that specifies two different device densities (e.g. `density-low` and `density-high`) the

query will match **both** devices with low density and high density. However, if the query specifies a device density and a platform (e.g. `density-low` and `platform-and`), then it will only match a device if it matches the platform **and** the density.

Listing 18. Example: Targeting styles to only Android devices with high density

```
@media platform-and, density-high {  
    ...  
}
```

Listing 19. Example: Targeting styles to iOS devices with high or low density

```
@media platform-ios, density-high, density-low {  
    ...  
}
```

Listing 20. Example: Targeting only Mac Desktop.

```
@media device-desktop, platform-mac {  
    ...  
}
```

5.12.2. Order or Precedence

The order of precedence when applying styles differs slightly from the way styles would be applied in standard CSS. The order of precedence is as follows:

1. Styles defined inside `@media` blocks will always take precedence over styles defined outside of `@media` blocks.
2. `@media` blocks with more query matches will take precedence over blocks with fewer query matches. E.g. A media block matching density, platform, and device will take precedence over a block that only matches the density and platform.
3. If the same style is defined in two media blocks which contain the same number of query matches, then the order precedence is `platform`, `device`, `density` in decreasing order. I.e. the block that matches on platform will take precedence over the block that matches on density.
4. If the same style is defined in two media blocks with identical query matches, then the order of precedence is undefined.

5.12.3. Font Scaling Constants

In some cases you may find that fonts are coming out too large or too small across the board on certain types of devices. You can use standard media queries to customize font sizes, but you can also use `font-scaling` constants to **scale** font sizes for the entire stylesheet based on platform, device, and/or density. In some cases you may find this approach easier.

For example, consider the following simple stylesheet that defines a font size of 2mm on labels:

```
Label {  
    font-size: 3mm;  
}
```

During testing, perhaps you find that, on desktop, the fonts are a little bit too small. In this case, you can apply a font-scale constant that only applies to the desktop.

```
#Constants {  
    device-desktop-font-scale: "1.5";  
}  
  
Label {  
    font-size: 3mm;  
}
```

Now, on most devices the Label style will have **3mm** fonts. But on desktop, it will have **4.5mm** fonts.

The above would be roughly equivalent to:

```
Label {  
    font-size: 3mm;  
}  
  
@media device-desktop {  
    Label {  
        font-size: 4.5mm;  
    }  
}
```

Listing 21. Example: Font-scaling based on device, platform, and density

```
#Constants {  
    device-phone-font-scale: "1.5";  
    device-tablet-font-scale: "1.2";  
    device-desktop-font-scale: "1.4";  
    platform-ios-font-scale: "0.9";  
    density-low-font-scale: "1.2";  
    platform-ios-density-low-font-scale: "1.3";  
}
```

All matching **font-scale** constants will be applied to the styles. If you define 3 font-scale constants that all match the current runtime environment, they will all be applied. E.g. If there are 3 matching font-scale constants with "2.0", "3.0", and "4.0", then fonts will be scaled by $2 \times 3 \times 4 = 24$!



6. The Components of Codename One

This chapter covers the components of Codename One. Not all components are covered, but it tries to go deeper than the [JavaDocs](https://www.codenameone.com/javadoc/) [<https://www.codenameone.com/javadoc/>].

6.1. Container

The Codename One container is a base class for many high level components; a container is a component that can contain other components.

Every component has a parent container that can be null if it isn't within a container at the moment or is a top-level container. A container can have many children.

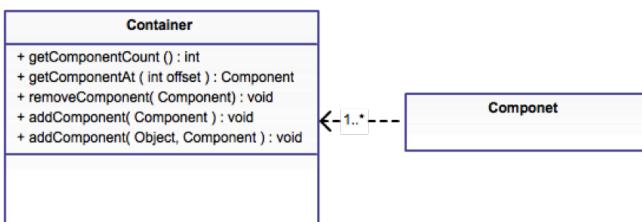


Figure 186. Component-Container relationship expressed as UML

Components are arranged in containers using layout managers which are algorithms that determine the arrangement of components within the container.

You can read more about layout managers in the [basics section](https://www.codenameone.com/manual/basics.html#component-container-hierarchy) [<https://www.codenameone.com/manual/basics.html#component-container-hierarchy>].

6.1.1. Composite Components

Codename One components share a very generic hierarchy of inheritance e.g. [Button](https://www.codenameone.com/javadoc/com/codename1/ui/Button.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/Button.html>] derives from [Label](https://www.codenameone.com/javadoc/com/codename1/ui/Label.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/Label.html>] and thus receives all its abilities.

However, some components are composites and derive from the [Container](https://www.codenameone.com/javadoc/com/codename1/ui/Container.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/Container.html>] class. E.g. the [MultiButton](https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html) [<https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html>] is a composite button that derives from [Container](#) but acts/looks like a [Button](#). Normally this is pretty seamless for the developer, with a few things to keep in mind.

- You should not use the [Container](#) derived methods on such a composite component (e.g. [add](#) /[remove](#) etc.).
- You can't cast it to the type that it relates to e.g. you can't cast [MultiButton](#) to [Button](#).
- Events might be more nuanced. E.g. if you rely on [ActionEvent.getSource\(\)](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getSource-->] or [ActionEvent.getComponent\(\)](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getComponent-->] notice that they might not behave the way you would expect. For a [MultiButton](#) they will return the underlying [Button](#). To workaround this we have [ActionEvent.getActualComponent\(\)](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getActualComponent-->]

ActionEvent.html#getActualComponent--].

Lead Component

Codename One has a rather unique feature for creating composite components: "lead components". This feature effectively allows components like **MultiButton** to act as if they are a single component while really being comprised of multiple components.

Lead components work by setting a single component within as the "leader" it determines the style state for all the components in the hierarchy so if we have a **Container** that is lead by a **Button** the button will determine if the selected/pressed state is returned for the entire container hierarchy.

This creates a case where a single **Component** has multiple nested **UIID**s e.g. 'MultiButton' has **UIID**s such as '**MultiLine1**' that can be customized via API's such as `setUIIDLine1` [<https://www.codenameone.com/javadoc/com/codenname1/components/MultiButton.html#setUIIDLine1-java.lang.String->].

The lead component also handles the events from a single source so clicking in one of the other components within the hierarchy will send the event to the leading **Button** resulting in action events that behave "oddly" (hence the need for `getActualComponent`);

You can learn more about lead components in [here](#) [<https://www.codenameone.com/manual/misc-features.html#lead-component-section>].

6.2. Form

Form [<https://www.codenameone.com/javadoc/com/codenname1/ui/Form.html>] is the top-level container of Codename One, **Form** derives from **Container** and is the element we "show". Only one form can be visible at any given time. We can get the currently visible **Form** using the code:

```
Form currentForm = Display.getInstance().getCurrent();
```

A form is a unique container in the sense that it has a title, a content area and optionally a menu/menu bar area. When invoking methods such as `add/remove` on a form, you are in fact invoking something that maps to this:

```
myForm.getContentPane().add(...);
```

Form is in effect just a **Container** [<https://www.codenameone.com/javadoc/com/codenname1/ui/Container.html>] that has a border layout, its north section is occupied by the title area and its south section by the optional menu bar. The center (which stretches) is the content pane. The content pane is where you place all your components.



Figure 187. Form layout graphic

You can see that every **Form** has space allocated for the title area. If you don't set the title it won't show up (its size will be zero), but it will still be there. The same isn't always true for the case of the menu bar, which can vary significantly. Effectively, the section that matters is the content pane, so the form tries to do the "right thing" by pretending to be the content pane. However, this isn't always seamless and sometimes code needs to just invoke `getContentPane()` in order to work directly with the container.



A good example for such a case is with layout animations. Animating the form might not produce the right results. When in doubt its pretty easy to just use `getContentPane` instead of working with the **Form** directly.

As you can see from [the graphic](#), **Form** has two layers that reside on top of the content pane/title. The first is the layered pane which allows you to place "always on top" components. The layered pane added implicitly when you invoke `getLayeredPane()`.



You still need to place components using layout managers in order to get them to appear in the right place when using the layered pane.

The second layer is the glass pane which allows you to draw arbitrary things on top of everything. The order in the image is indeed accurate:

1. **ContentPane** is lowest
2. **LayeredPane** is second
3. **GlassPane** is painted last



Its important to notice that a layered pane is on top of the **ContentPane** only and doesn't stretch to the title. A **GlassPane** usually stretches all the way but only with a "lightweight" title area e.g. the [Toolbar API](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html>].

The **GlassPane** allows developers to overlay UI on top of existing UI and paint as they see fit. This is useful for things that provide notification but don't want to intrude with application functionality.



In earlier versions of Codename One (pre-3.6), **LayeredPane** & **GlassPane** didn't work with "native" peer components such as media, browser, native maps etc, because peer components were always rendered "in front" of the Codename One UI canvas. However, current versions now allow for proper layering of peer components and light-weight components so that LayeredPane and GlassPane can be used seamlessly with peer components.

6.3. Dialog

A **Dialog** [https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html] is a special kind of **Form** that can occupy only a portion of the screen, it also has the additional functionality of the modal **show** method.

When showing a dialog we have two basic options: modeless and modal:

- Modal dialogs (the default) block the current EDT thread until the dialog is dismissed (to understand how they do it, read about **invokeAndBlock**).
Modal dialogs are an extremely useful way to prompt the user since the code can assume the user responded in the next line of execution. This promotes a linear & intuitive way of writing code.
- Modeless dialogs return immediately so a call to show such a dialog can't assume anything in the next line of execution. This is useful for features such as progress indicators where we aren't waiting for user input.

E.g. a modal dialog can be expressed as such:

```
if(Dialog.show("Click Yes Or No", "Select one", "Yes", "No")) {  
    // user clicked yes  
} else {  
    // user clicked no  
}
```

Notice that during the **show** call above the execution of the next line was "paused" until we got a response from the user and once the response was returned we could proceed directly.



All usage of **Dialog** must be within the Event Dispatch Thread (the default thread of Codename One). This is especially true for modal dialogs. The **Dialog** class knows how to "block the EDT" without blocking it.

To learn more about **invokeAndBlock** which is the workhorse behind the modal dialog functionality check out **the EDT section** [https://www.codenameone.com/manual/edt.html].

The **Dialog** class contains multiple static helper methods to quickly show user notifications, but also allows a developer to create a **Dialog** instance, add information to its content pane and show the dialog.



Dialogs contain a **ContentPane** just like **Form**.

When showing a dialog in this way, you can either ask Codename One to position the dialog in a specific general location (taken from the **BorderLayout** [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html] concept for locations) or position it by spacing it (in pixels) from the 4 edges of the screen.

E.g. you could do something like this to show a simple modal **Dialog**:

```
Dialog d = new Dialog("Title");
d.setLayout(new BorderLayout());
d.add(BorderLayout.CENTER, new SpanLabel("Dialog Body", "DialogBody"));
d.showPacked(BorderLayout.SOUTH, true);
```

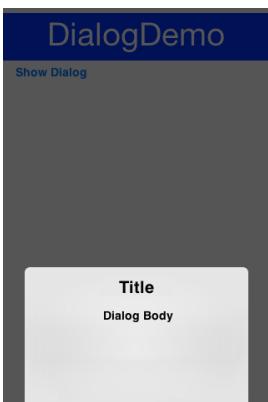


Figure 188. Custom modal Dialog in the south position



You can turn the code above to a modless `Dialog` by flipping the boolean `true` argument to `false`.

We can position a `Dialog` absolutely by determining the space from the edges e.g. with this code we can occupy the bottom portion of the screen:

```
Dialog d = new Dialog("Title");
d.setLayout(new BorderLayout());
d.add(BorderLayout.CENTER, new SpanLabel("Dialog Body", "DialogBody"));
d.show(hi.getHeight() / 2, 0, 0, 0);
```

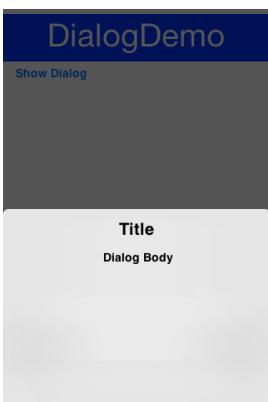


Figure 189. Custom Dialog positioned absolutely



`hi` is the name of the parent `Form` in the sample above.

6.3.1. Styling Dialogs

It's important to style a `Dialog` using `getDialogStyle()` [<https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html#getDialogStyle-->] or `setDialogUID` [<https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html#setDialogUID-->]

codename1/ui/Dialog.html#setDialogUIID-java.lang.String-] methods rather than styling the dialog object directly.

The reason for this is that the **Dialog** is really a **Form** that takes up the whole screen. The **Form** that is visible behind the **Dialog** is rendered as a screenshot. So customizing the actual **UIID** of the **Dialog** won't produce the desired results.

6.3.2. Tint and Blurring

By default a **Dialog** uses a platform specific tint color when it is showing e.g. notice the background in the image below is tinted:

```
Form hi = new Form("Tint Dialog", new BoxLayout(BoxLayout.Y_AXIS));
Button showDialog = new Button("Tint");
showDialog.addActionListener((e) -> Dialog.show("Tint", "Is On....", "OK", null));
hi.add(showDialog);
hi.show();
```

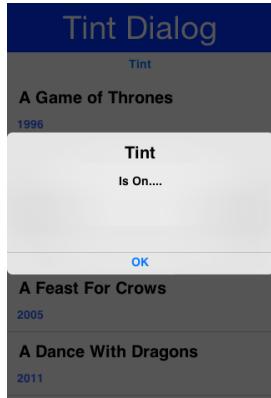


Figure 190. Dialog with tinted background

The tint color can be manipulated on the parent form, you can set it to any AARRGGBB value to set any color using the **setTintColor** method. Notice that this is invoked on the parent form and not on the **Dialog**!



This is an AARRGGBB value and not an RRGGBB value! This means that 0 will be transparent.

You can also manipulate this default value globally using the theme constant **tintColor**. The sample below tints the background in green:

```
Form hi = new Form("Tint Dialog", new BoxLayout(BoxLayout.Y_AXIS));
hi.setTintColor(0x7700ff00);
Button showDialog = new Button("Tint");
showDialog.addActionListener((e) -> Dialog.show("Tint", "Is On....", "OK", null));
hi.add(showDialog);
hi.show();
```

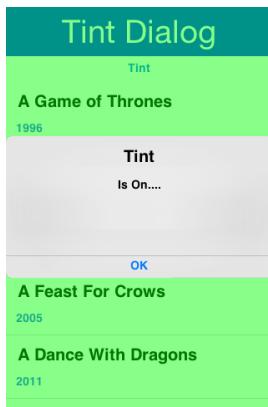


Figure 191. Dialog with green tinted background

We can apply Gaussian blur to the background of a dialog to highlight the foreground further and produce a very attractive effect. We can use the `setDefaultBlurBackgroundRadius` to apply this globally, we can use the theme constant `dialogBlurRadiusInt` to do the same or we can do this on a per `Dialog` basis using `setBlurBackgroundRadius`.



Not all device types support blur you can test if your device supports it using `Display.getInstance().isGaussianBlurSupported()`. If blur isn't supported the blur setting will be ignored.

```
Form hi = new Form("Blur Dialog", new BoxLayout(BoxLayout.Y_AXIS));
Dialog.setDefaultBlurBackgroundRadius(8);
Button showDialog = new Button("Blur");
showDialog.addActionListener((e) -> Dialog.show("Blur", "Is On....", "OK", null));
hi.add(showDialog);
hi.show();
```

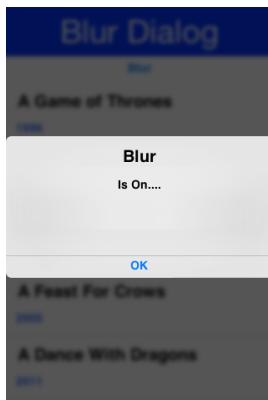


Figure 192. The blur effect coupled with the OS default tint

It might be a bit hard to notice the blur effect with the tinting so here is the same code with tinting disabled:

```
hi.setTintColor(0);
```

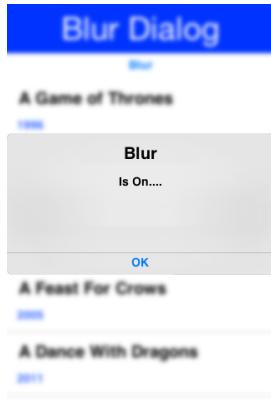


Figure 193. The blur effect is more pronounced when the tint is disabled

6.3.3. Popup Dialog

A popup dialog is a common mobile paradigm showing a **Dialog** that points at a specific component. It's just a standard **Dialog** that is shown in a unique way:

```
Dialog d = new Dialog("Title");
d.setLayout(new BorderLayout());
d.add(BorderLayout.CENTER, new SpanLabel("Dialog Body", "DialogBody"));
d.showPopupDialog(showDialog);
```



Figure 194. Popup Dialog

The popup dialog accepts a **Component** [<https://www.codenameone.com/javadoc/com/codename1/ui/Component.html>] or **Rectangle** [<https://www.codenameone.com/javadoc/com/codename1/ui/geom/Rectangle.html>] to point at and handles the rest.

Styling The Arrow Of The Popup Dialog

When Codename One was young we needed a popup arrow implementation but our low level graphics API was pretty basic. As a workaround we created a version of the 9-piece image border that supported pointing arrows at a component.

Today Codename One supports pointing an arrow from the **RoundRectBorder** class. This is implicit for the **PopupDialog** UI. This allows for better customization of the border (color etc.) and it looks better on newer displays. It also works on all OSs. Right now only the iOS theme has the old image border approach.



This will change with a future update where all OS's will align and iOS will use the lightweight popup too



You can make all OS's act the same way by overriding the `PopupDialog` UIID and defining its style to `RoundRectBorder`

The new `RoundRectBorder` support works by setting the track component property on border. When that's done the border implicitly points to the right location.

If you still need deeper customization of the arrow you can still use the old 9-piece border functionality illustrated below.

Legacy 9-Piece Border Arrow

One of the harder aspects of a popup dialog is the construction of the theme elements required for arrow styling. To get that sort of behavior you will need a custom image border and 4 arrows pointing in each direction that will be overlaid with the border.



The sizes of the arrow images should be similarly proportioned and fit within the image borders whitespace. The block image of the dialog should have empty pixels in the sides to reserve space for the arrow. E.g. if the arrows are all 32x32 pixels then the `PopupDialog` image should have 32 pixels of transparent pixels around it.

You will need to define the following theme constants for the arrow to work:

```
PopupDialogArrowBool=true  
PopupDialogArrowTopImage=arrow up image  
PopupDialogArrowBottomImage=arrow down image  
PopupDialogArrowLeftImage=arrow left image  
PopupDialogArrowRightImage=arrow right image
```

Then style the `PopupDialog` UIID with the image for the `Dialog` itself.

6.4. InteractionDialog

Dialogs in Codename One can be modal or modeless, the former blocks the calling thread and the latter does not. However, there is another definition to those terms: A modal dialog blocks access to the rest of the UI while a modeless dialog "floats" on top of the UI.

In that sense, all dialogs in Codename One are modal; they block the parent form since they are effectively just forms that show the "parent" in their background. `InteractionDialog` [<https://www.codenameone.com/javadoc/com/codename1/components/InteractionDialog.html>] has an API that is very similar to the `Dialog` [<https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html>] API but, unlike dialog, it never blocks anything. Neither the calling thread nor the UI.

i **InteractionDialog** isn't a **Dialog** since it doesn't share the same inheritance hierarchy. However, it acts and "feels" like a **Dialog** despite the fact that it's just a **Container** in the **LayeredPane**.

InteractionDialog is really just a container that is positioned within the layered pane. Notice that because of that design, you can have only one such dialog at the moment and, if you add something else to the layered pane, you might run into trouble.

Using the interaction dialog is pretty trivial and very similar to dialog:

```
InteractionDialog dlg = new InteractionDialog("Hello");
dlg.setLayout(new BorderLayout());
dlg.add(BorderLayout.CENTER, new Label("Hello Dialog"));
Button close = new Button("Close");
close.addActionListener((ee) -> dlg.dispose());
dlg.addComponent(BorderLayout.SOUTH, close);
Dimension pre = dlg.getContentPane().getPreferredSize();
dlg.show(0, 0, Display.getInstance().getDisplayWidth() - (pre.getWidth() + pre.getHeight() / 6), 0);
```

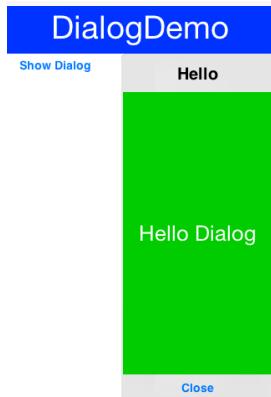


Figure 195. Interaction Dialog

This will show the dialog on the right hand side of the screen, which is pretty useful for a floating in place dialog.

i The **InteractionDialog** can only be shown at absolute or popup locations. This is inherent to its use case which is "non-blocking". When using this component you need to be very aware of its location.

6.5. Label

Label [<https://www.codenameone.com/javadoc/com/codename1/ui/Label.html>] represents a text, icon or both. **Label** is also the base class of **Button** which in turn is the base class for **RadioButton** & **CheckBox**. Thus the functionality of the **Label** class extends to all of these components.

Label text can be positioned in one of 4 locations as such:

```

Label left = new Label("Left", icon);
left.setTextPosition(Component.LEFT);
Label right = new Label("Right", icon);
right.setTextPosition(Component.RIGHT);
Label bottom = new Label("Bottom", icon);
bottom.setTextPosition(Component.BOTTOM);
Label top = new Label("Top", icon);
top.setTextPosition(Component.TOP);
hi.add(left).add(right).add(bottom).add(top);

```



Figure 196. Label positions

Label allows only a single line of text, line breaking is a very expensive operation on mobile devices [2] and so the **Label** class doesn't support it.



SpanLabel supports multiple lines with a single label, notice that it does carry a performance penalty for this functionality.

Labels support tickering and the ability to end with “...” if there isn't enough space to render the label. Developers can determine the placement of the label relatively to its icon in quite a few powerful ways.

6.5.1. Label Gap

The gap between the label text & the icon defaults to 2 pixels due to legacy settings. The **setGap** method of **Label** accepts a gap size in pixels.

Two pixels is low for most cases & it's hard to customize for each **Label**.

You can use the theme constant **labelGap** which is a floating point value you can specify in millimeters that will allow you to determine the default gap for a label. You can also customize this manually using the method **Label.setDefaultGap(int)** which determines the default gap in pixels.

6.5.2. Autosizing Labels

One of the common requests we received over the years is a way to let text "fit" into the allocated space so the font will match almost exactly the width available. In some designs this is very important but it's also very tricky. Measuring the width of a String is a surprisingly expensive

operation on some OS's. Unfortunately, there is no other way other than trial & error to find the "best size".

Still despite the fact that something is "slow" we might still want to use it for some cases, this isn't something you should use in a renderer, infinite scroll etc. and we recommend minimizing the usage of this feature as much as possible.

This feature is only applicable to [Label](#) and its subclasses (e.g. [Button](#)), with components such as [TextArea](#) (e.g. [SpanButton](#)) the choice between shrinking and line break would require some complex logic.

To activate this feature just use `setAutoSizeMode(true)` e.g.:

```
Form hi = new Form("AutoSize", BoxLayout.y());  
  
Label a = new Label("Short Text");  
a.setAutoSizeMode(true);  
Label b = new Label("Much Longer Text than the previous line...");  
b.setAutoSizeMode(true);  
Label c = new Label("MUCH MUCH MUCH Much Longer Text than the previous line by a pretty big margin...");  
c.setAutoSizeMode(true);  
  
Label a1 = new Button("Short Text");  
a1.setAutoSizeMode(true);  
Label b1 = new Button("Much Longer Text than the previous line...");  
b1.setAutoSizeMode(true);  
Label c1 = new Button("MUCH MUCH MUCH Much Longer Text than the previous line by a pretty big margin...");  
c1.setAutoSizeMode(true);  
hi.addAll(a, b, c, a1, b1, c1);  
  
hi.show();
```



Figure 197. Automatically sizes the fonts of the buttons/labels based on text and available space

6.6. TextField and TextArea

The [TextField](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html>] class derives from the [TextArea](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/TextArea.html>] class, and both are used for text input in Codename One.

[TextArea](#) defaults to multi-line input and [TextField](#) defaults to single line input but both can be used

in both cases. The main differences between **TextField** and **TextArea** are:

- Blinking cursor is rendered on **TextField** only
- **DataChangeListener** [https://www.codenameone.com/javadoc/com/codename1/ui/events/DataChangedListener.html] is only available in **TextField**. This is crucial for character by character input event tracking
- **Done listener** [https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html#setDoneListener-com.codename1.ui.events.ActionListener] is only available in the **TextField**
- Different **UIID**



The semantic difference between **TextField** & **TextArea** dates back to the ancestor of Codename One: LWUIT. Feature phones don't have "proper" in-place editing capabilities & thus **TextField** was introduced to allow such input.

Because it lacks the blinking cursor capability **TextArea** is often used as a multi-line label and is used internally in **SpanLabel**, **SpanButton** etc.



A common use case is to have an important text component in edit mode immediately as we enter a **Form**. Codename One forms support this exact use case thru the **Form.setEditableOnShow(TextArea)** [https://www.codenameone.com/javadoc/com/codename1/ui/Form.html#setEditOnShow-com.codename1.ui.TextArea-] method.

TextField & **TextArea** support constraints for various types of input such as **NUMERIC**, **EMAIL**, **URL**, etc. Those usually affect the virtual keyboard used, but might not limit input in some platforms. E.g. on iOS even with **NUMERIC** constraint you would still be able to input characters.



If you need to prevent specific types of input check out the [validation section](#).

The following sample shows off simple text field usage:

```

TableLayout tl;
int spanButton = 2;
if(Display.getInstance().isTablet()) {
    tl = new TableLayout(7, 2);
} else {
    tl = new TableLayout(14, 1);
    spanButton = 1;
}
tl.setGrowHorizontally(true);
hi.setLayout(tl);

TextField firstName = new TextField("", "First Name", 20, TextArea.ANY);
TextField surname = new TextField("", "Surname", 20, TextArea.ANY);
TextField email = new TextField("", "E-Mail", 20, TextArea.EMAILADDR);
TextField url = new TextField("", "URL", 20, TextArea.URL);
TextField phone = new TextField("", "Phone", 20, TextArea.PHONENUMBER);

TextField num1 = new TextField("", "1234", 4, TextArea.NUMERIC);
TextField num2 = new TextField("", "1234", 4, TextArea.NUMERIC);
TextField num3 = new TextField("", "1234", 4, TextArea.NUMERIC);
TextField num4 = new TextField("", "1234", 4, TextArea.NUMERIC);

Button submit = new Button("Submit");
TableLayout.Constraint cn = tl.createConstraint();
cn.setHorizontalSpan(spanButton);
cn.setHorizontalAlignment(Component.RIGHT);
hi.add("First Name").add(firstName).
    add("Surname").add(surname).
    add("E-Mail").add(email).
    add("URL").add(url).
    add("Phone").add(phone).
    add("Credit Card").
        add(GridLayout.encloseIn(4, num1, num2, num3, num4)).
    add(cn, submit);

```



Figure 198. Simple text component sample



The **Toolbar section** contains a very elaborate **TextField** search sample with **DataChangeListener** and rather unique styling.

6.6.1. Masking

A common use case when working with text components is the ability to "mask" input e.g. in the credit card number above we would want 4 digits for each text field and don't want the user to tap Next 3 times.

Masking allows us to accept partial input in one field and implicitly move to the next, this can be used to all types of complex input thanks to the text component API. E.g with the code above we can mask the credit card input so the cursor jumps to the next field implicitly using this code:

```
automoveToNext(num1, num2);
automoveToNext(num2, num3);
automoveToNext(num3, num4);
```

Then implement the method `automoveToNext` as:

```
private void automoveToNext(final TextField current, final TextField next) {
    current.addDataChangedListener((type, index) -> {
        if(current.getText().length() == 5) {
            current.stopEditing();
            current.setText(val.substring(0, 4));
            next.setText(val.substring(4));
            next.startEditingAsync();
        }
    });
}
```

Notice we can invoke `stopEditing(Runnable)` where we receive a callback as editing is stopped.

6.6.2. The Virtual Keyboard

A common misconception for developers is assuming the virtual keyboard represents "keys". E.g. developers often override the "keyEvent" callbacks which are invoked for physical keyboard typing and expect those to occur with a virtual keyboard.

This isn't the case since a virtual keyboard is a very different beast. With a virtual keyboard characters typed might produce a completely different output due to autocorrect. Some keyboards don't even have "keys" in the traditional sense or don't type them in the traditional sense (e.g. swiping).



The constraint property for the `TextField/TextArea` is crucial for a virtual keyboard.



When working with a virtual keyboard it's important that the parent `Container` for the `TextField/TextArea` is scrollable. Otherwise the component won't be reachable or the UI might be distorted when the keyboard appears.

Action Button Client Property

By default, the virtual keyboard on Android has a "Done" button, you can customize it to be a search icon, a send icon, or a go icon using a hint such as this:

```
searchTextField.putClientProperty("searchField", Boolean.TRUE);  
sendTextField.putClientProperty("sendButton", Boolean.TRUE);  
goTextField.putClientProperty("goButton", Boolean.TRUE);
```

This will adapt the icon for the action on the keys.

Next and Done on iOS

We try to hide a lot of the platform differences in Codename One, input is **very** different between OS's. A common reliance is the ability to send the "Done" event when the user presses the **Done** button. Unfortunately this button doesn't always exist e.g. if there is an **Enter** button (due to multiline input) or if there is a **Next** button in that place.

To make the behavior more uniform we slightly customized the iOS keyboard as such:



Figure 199. Next virtual keyboard with toolbar



Figure 200. Done virtual keyboard without toolbar



This works with 3rd party keyboards too...

However, this behavior might not be desired so to block that we can do:

```
tf.putClientProperty("iosHideToolbar", Boolean.TRUE);
```

This will hide the toolbar for that given field.

You can customize the color of the **Done** button in the toolbar by setting the `ios.doneButtonColor` display property. E.g. To change the color to red, you could do `Display.getInstance().setProperty("ios.doneButtonColor", String.valueOf(0xff0000)).@since 5.0`

6.6.3. Clearable Text Field

iOS has a convention where an X can be placed after the text field to clear it. Some Android apps have it but there is no native support for that as of this writing.

You can wrap a `TextField` with a clearable wrapper to get this effect on all platforms. E.g. replace this:

```
cnt.add(myTextField);
```

With this:

```
cnt.add(ClearableTextField.wrap(myTextField));
```

You can also specify the size of the clear icon if you wish. This is technically just a `Container` with the text field style and a button to clear the text at the edge.

6.7. TextComponent

When building input forms we sometimes want to adapt to the native OS behavior and create a UI that's a bit more distinct to the native OS. `TextField` and `TextArea` are very low level, you can create an Android style UI with such components but it might look out of place in iOS.

E.g. this is how most of us would expect the UI to look on iOS and Android respectively:

Title	Vintage Dress
Price	5
Location	Somewhere
Description	Long text that can span multiple lines

Figure 201. Text Input on iOS

Title	
Vintage dress	
Price	Location
5	Somewhere
Description	
Long text	

Figure 202. Text Input on Android

Doing this with text fields is possible but would require code that looks a bit different and jumps through hoops. `TextComponent` allows this exact UI without forcing developers to write OS specific code:

```
TextModeLayout tl = new TextModeLayout(3, 2);
Form f = new Form("Pixel Perfect", tl);
TextComponent title = new TextComponent().label("Title");
TextComponent price = new TextComponent().label("Price");
TextComponent location = new TextComponent().label("Location");
TextComponent description = new TextComponent().label("Description").multiline(true);

f.add(tl.createConstraint().horizontalSpan(2), title);
f.add(tl.createConstraint().widthPercentage(30), price);
f.add(tl.createConstraint().widthPercentage(70), location);
f.add(tl.createConstraint().horizontalSpan(2), description);
f.setEditOnShow(title.getField());
f.show();
```



This code uses the **TextModeLayout** which is discussed in the layouts section

The text component uses a builder approach to set various values e.g.:

```
TextComponent t = new TextComponent().
    text("This appears in the text field").
    hint("This is the hint").
    label("This is the label").
    multiline(true);
```

The code is pretty self explanatory and more convenient than typical setters/getters. It automatically handles the floating hint style of animation when running on Android.

6.7.1. Error Handling

The validator class supports text component and it should "just work". But the cool thing is that it uses the material design convention for error handling!

So if we add to the sample above a **Validator**:

```
Validator val = new Validator();
val.addConstraint(title, new LengthConstraint(2));
val.addConstraint(price, new NumericConstraint(true));
```

You would see something that looks like this on Android:

Title
Input must be at least 2 characters

Price
The value must be a number

Description

Figure 203. Error handling when the text is blank

Title
a
Input must be at least 2 characters

Price
Location
a
The value must be a number

Description

Figure 204. Error handling when there is some input (notice red title label)

Title ✗

Price ✗

Location

Description

Figure 205. On iOS the situation hasn't changed much yet

The underlying system is the `errorMessage` method which you can chain like the other methods on `TextComponent` as such:

```
TextComponent tc = new TextComponent().
    label("Input Required").
    errorMessage("Input is essential in this field");
```

6.7.2. InputComponent and PickerComponent

To keep the code common and generic we use the `InputComponent` abstract base class and derive the other classes from that. `PickerComponent` is currently the only other option.

A picker can work with our existing sample using code like this:

```

TextModeLayout tl = new TextModeLayout(3, 2);
Form f = new Form("Pixel Perfect", tl);
TextComponent title = new TextComponent().label("Title");
TextComponent price = new TextComponent().label("Price");
TextComponent location = new TextComponent().label("Location");
PickerComponent date = PickerComponent.createDate(new Date()).label("Date");
TextComponent description = new TextComponent().label("Description").multiline(true);
Validator val = new Validator();
val.addConstraint(title, new LengthConstraint(2));
val.addConstraint(price, new NumericConstraint(true));
f.add(tl.createConstraint().widthPercentage(60), title);
f.add(tl.createConstraint().widthPercentage(40), date);
f.add(location);
f.add(price);
f.add(tl.createConstraint().horizontalSpan(2), description);
f.setEditOnShow(title.getField());
f.show();

```

This produces the following which looks pretty standard:

Title	
Date	10/31/17
Location	
Price	
Description	

Figure 206. Picker component taking place in iOS

Title	Date
	10/31/17
Location	Price
Description	

Figure 207. And in Android

The one tiny thing you should notice with the `PickerComponent` is that we don't construct the picker component using `new PickerComponent()`. Instead we use create methods such as `PickerComponent.createDate(new Date())`. The reason for that is that we have many types of pickers and it wouldn't make sense to have one constructor.

6.7.3. Underlying Theme Constants and UIID's

These varying looks are implemented via a combination of layouts, theme constants and UIID's. The most important UIID's are: `TextComponent`, `FloatingHint` & `TextHint`.

There are several theme constants related that can manipulate some pieces of this functionality:

- `textComponentErrorColor` a hex RGB color which defaults to null in which case this has no effect. When defined this will change the color of the border and label to the given color to match the material design styling. This implements the red border underline in cases of error and the label text color change
- `textComponentOnTopBool` toggles the on top mode which makes things look like they do on Android. This defaults to true on Android and false on other OS's. This can also be manipulated via the `onTopMode(boolean)` method in `InputComponent` however the layout will only use the theme constant
- `textComponentAnimBool` toggles the animation mode which again can be manipulated by a method in `InputComponent`. If you want to keep the UI static without the floating hint effect set this to false. Notice this defaults to true only on Android
- `textComponentFieldUUID` sets the UUID of the text field to something other than `TextField` this is useful for platforms such as iOS where the look of the text field is different within the text component. This allows us to make the background of the text field transparent when it's within the `TextComponent` and make it different from the regular text field

6.8. Button

`Button` [<https://www.codenameone.com/javadoc/com/codename1/ui/Button.html>] is a subclass of `Label` and as a result it inherits all of its functionality, specifically icon placement, tickering, etc.

Button adds to the mix some additional states such as a pressed `UUID` state and pressed icon.



There are additional icon states in `Button` such as rollover and disabled icon.

`Button` also exposes some functionality for subclasses specifically the `setToggle` method call which has no meaning when invoked on a `Button` but has a lot of implications for `CheckBox` & `RadioButton`.

`Button` event handling can be performed via an `ActionListener` [<https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html>] or via a `Command` [<https://www.codenameone.com/javadoc/com/codename1/ui/Command.html>].



Changes in a `Command` won't be reflected into the `Button` after the command was set to the `Button`.

Here is a trivial hello world style `Button`:

```
Form hi = new Form("Button");
Button b = new Button("My Button");
hi.add(b);
b.addActionListener((e) -> Log.p("Clicked"));
```

Button

My Button

Figure 208. Simple button in the iOS styling, notice iOS doesn't have borders on buttons...

Such a button can be styled to look like a link using code like this or simply by making these settings in the theme and using code such as `btn.setUIID("Hyperlink")`.

```
Form hi = new Form("Button");
Button b = new Button("Link Button");
b.getAllStyles().setBorder(Border.createEmpty());
b.getAllStyles().setTextDecoration(Style.TEXT_DECORATION_UNDERLINE);
hi.add(b);
b.addActionListener((e) -> Log.p("Clicked"));
```

Button

Link Button

Figure 209. Button styled to look like a link

6.8.1. Uppercase Buttons

Buttons on Android's material design UI use upper case styling which isn't the case for iOS. To solve this we have the method `setCapsText(boolean)` in `Button` which has the corresponding `isCapsText`, `isCapsTextDefault` & `setCapsTextDefault(boolean)`. This is pretty core to Codename One so to prevent this from impacting everything unless you explicitly invoke `setCapsText(boolean)` the default value of `true` will only apply when the UIID is `Button`, `RaisedButton` or for the builtin `Dialog` buttons.

We also have a theme constant: `capsButtonTextBool`. This constant controls caps text behavior from the theme and is set to true in the Android native theme.

6.8.2. Raised Button

Raised button is a style of button that's available on Android and used to highlight an important action within a form. To confirm with the material design UI guidelines you might want to leverage a raised button UI element on Android but use a regular button everywhere else.

First we need to know whether a raised button exists in the theme. So on Android this will return true but on other OS's it will return false. A potential future update might make another platform true based on UI guidelines in other OS's.

For this purpose we've got the theme constant `hasRaisedButtonBool` which will return true on Android but will be false elsewhere. You can use it like this:

```
if(UIManager.getInstance().isThemeConstant("hasRaisedButtonBool", false)) {  
    // that means we can use a raised button  
}
```

To enable this we have the `RaisedButton` UIID that derives from `Button` and will act like it except for the places where `hasRaisedButtonBool` is true in which case it will look like this:

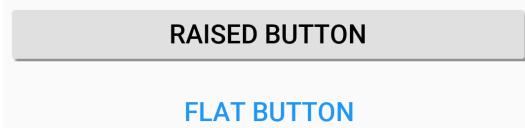


Figure 210. Raised and flat button in simulator

Notice that you can easily customize the colors of these buttons now since the border respects user colors...

In this case I just set the background color to purple and the foreground to white:



RAISED BUTTON

Figure 211. Purple raised button

```
Form f = new Form("Pixel Perfect", BoxLayout.y());  
Button b = new Button("Raised Button", "RaisedButton");  
Button r = new Button("Flat Button");  
f.add(b);  
f.add(r);  
f.show();
```

6.8.3. Ripple Effect

The ripple effect in material design highlights the location of the finger and grows as a circle to occupy the full area of the component as the user presses the button.

We have the ability to perform a ripple effect by darkening the touched area and growing that in a quick animation.

Ripple effect can be applied to any component but we currently only have it turned on for buttons on Android which also applies to things like title commands, side menu elements etc. This might not apply at this moment to lead components like multi-buttons but that might change in the future.

`Component` has a property to enable the ripple effect `setRippleEffect(boolean)` and the corresponding `isRippleEffect()`. You can turn it on or off individually in the component level. However, `Button` has static `setButtonRippleEffectDefault(boolean)` and `isButtonRippleEffectDefault()`. These allow us to define the default behavior for all the buttons and that can be configured via the theme constant `buttonRippleBool` which is currently on by default on the native Android theme.

6.9. CheckBox/RadioButton

[CheckBox](https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html) [https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html] & [RadioButton](https://www.codenameone.com/javadoc/com/codename1/ui/RadioButton.html) [https://www.codenameone.com/javadoc/com/codename1/ui/RadioButton.html] are subclasses of button that allow for either a toggle state or exclusive selection state.

Both [CheckBox](#) & [RadioButton](#) have a selected state that allows us to determine their selection.



[RadioButton](#) doesn't allow us to "deselect" it, the only way to "deselect" a [RadioButton](#) is by selecting another [RadioButton](#).

The [CheckBox](#) can be added to a [Container](#) like any other [Component](#) but the [RadioButton](#) must be associated with a [ButtonGroup](#) otherwise if we have more than one set of [RadioButton](#)s in the form we might have an issue.

Notice in the sample below that we associate all the radio buttons with a group but don't do anything with the group as the radio buttons keep the reference internally. We also show the opposite side functionality and icon behavior:

```
CheckBox cb1 = new CheckBox("CheckBox No Icon");
cb1.setSelected(true);
CheckBox cb2 = new CheckBox("CheckBox With Icon", icon);
CheckBox cb3 = new CheckBox("CheckBox Opposite True", icon);
CheckBox cb4 = new CheckBox("CheckBox Opposite False", icon);
cb3.setOppositeSide(true);
cb4.setOppositeSide(false);
RadioButton rb1 = new RadioButton("Radio 1");
RadioButton rb2 = new RadioButton("Radio 2");
RadioButton rb3 = new RadioButton("Radio 3", icon);
new ButtonGroup(rb1, rb2, rb3);
rb2.setSelected(true);
hi.add(cb1).add(cb2).add(cb3).add(cb4).add(rb1).add(rb2).add(rb3);
```

Checks & Radios

CheckBox No Icon	<input checked="" type="checkbox"/>
▲ CheckBox With Icon	<input type="checkbox"/>
▲ CheckBox Opposite True	<input type="checkbox"/>
□ ▲ CheckBox Opposite False	<input type="checkbox"/>
Radio 1	
Radio 2	✓
▲ Radio 3	

Figure 212. [RadioButton](#) & [CheckBox](#) usage

Both of these components can be displayed as toggle buttons (see the toggle button section below), or just use the default check mark/filled circle appearance based on the type/OS.

6.9.1. Toggle Button

A toggle button is a button that is pressed and stays pressed. When a toggle button is pressed again it's released from the pressed state. Hence the button has a selected state to indicate if it's pressed or not exactly like the `CheckBox/RadioButton` components in Codename One.

To turn any `CheckBox` or `RadioButton` to a toggle button just use the `setToggle(true)` method. Alternatively you can use the static `createToggle` method on both `CheckBox` and `RadioButton` to create a toggle button directly.



Invoking `setToggle(true)` implicitly converts the `UIID` to `ToggleButton` unless it was changed by the user from its original default value.

We can easily convert the sample above to use toggle buttons as such:

```
CheckBox cb1 = CheckBox.createToggle("CheckBox No Icon");
cb1.setSelected(true);
CheckBox cb2 = CheckBox.createToggle("CheckBox With Icon", icon);
CheckBox cb3 = CheckBox.createToggle("CheckBox Opposite True", icon);
CheckBox cb4 = CheckBox.createToggle("CheckBox Opposite False", icon);
cb3.setOppositeSide(true);
cb4.setOppositeSide(false);
ButtonGroup bg = new ButtonGroup();
RadioButton rb1 = RadioButton.createToggle("Radio 1", bg);
RadioButton rb2 = RadioButton.createToggle("Radio 2", bg);
RadioButton rb3 = RadioButton.createToggle("Radio 3", icon, bg);
rb2.setSelected(true);
hi.add(cb1).add(cb2).add(cb3).add(cb4).add(rb1).add(rb2).add(rb3);
```

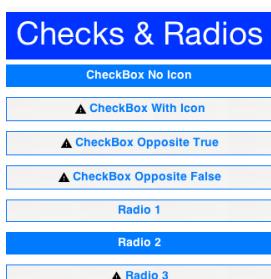


Figure 213. Toggle button converted sample

That's half the story though, to get the full effect of some cool toggle button UI's we can use a `ComponentGroup` [<https://www.codenameone.com/javadoc/com/codename1/ui/ComponentGroup.html>]. This allows us to create a button bar effect with the toggle buttons.

E.g. lets enclose the `CheckBox` components in a vertical `ComponentGroup` and the `RadioButton`s in a horizontal group. We can do this by changing the last line of the code above as such:

```
hi.add(ComponentGroup.enclose(cb1, cb2, cb3, cb4));
    add(ComponentGroup.encloseHorizontal(rb1, rb2, rb3));
```

Checks & Radios



Figure 214. Toggle button converted sample wrapped in ComponentGroup

6.10. ComponentGroup

[ComponentGroup](https://www.codenameone.com/javadoc/com/codename1/ui/ComponentGroup.html) [https://www.codenameone.com/javadoc/com/codename1/ui/ComponentGroup.html] is a special container that can be either horizontal or vertical ([BoxLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/Layout.html) [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/Layout.html] X_AXIS or Y_AXIS respectively).

ComponentGroup "restyles" the elements within the group to have a [UUID](#) that allows us to create a "round border" effect that groups elements together.

The following code adds 4 component groups to a [Container](#) to demonstrate the various [UUID](#) changes:

```
hi.add("Three Labels").
    add(ComponentGroup.enclose(new Label("GroupElementFirst UUID"), new Label("GroupElement UUID"), new
Label("GroupElementLast UUID"))).
    add("One Label").
    add(ComponentGroup.enclose(new Label("GroupElementOnly UUID"))).
    add("Three Buttons").
    add(ComponentGroup.enclose(new Button("ButtonGroupFirst UUID"), new Button("ButtonGroup UUID"), new
Button("ButtonGroupLast UUID"))).
    add("One Button").
    add(ComponentGroup.enclose(new Button("ButtonGroupOnly UUID")));
```

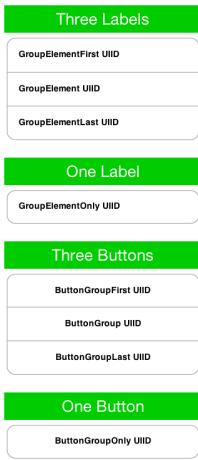


Figure 215. `ComponentGroup` adapts the `UIID`'s of the components added so we can style them

Notice the following about the code above and the resulting image:

- Buttons have a different `UIID` than other element types. Their styling is slightly different in such UI's so you need to pay attention to that.
- When an element is placed alone within a `ComponentGroup` its a special case `UIID`.



By default, `ComponentGroup` does **nothing**. You need to explicitly activate it in the theme by setting a theme property to true. Specifically you need to set `ComponentGroupBool` to `true` for `ComponentGroup` to do something otherwise its just a box layout container! The `ComponentGroupBool` flag is true by default in the iOS native theme.

When `ComponentGroupBool` is set to true, the component group will modify the styles of all components placed within it to match the element `UIID` given to it (`GroupElement` by default) with special caveats to the first/last/only elements. E.g.

1. If I have one element within a component group it will have the `UIID`: `GroupElementOnly`
2. If I have two elements within a component group they will have the `UIID`'s `GroupElementFirst`, `GroupElementLast`
3. If I have three elements within a component group they will have the `UIID`'s `GroupElementFirst`, `GroupElement`, `GroupElementLast`
4. If I have four elements within a component group they will have the `UIID`'s `GroupElementFirst`, `GroupElement`, `GroupElement`, `GroupElementLast`

This allows you to define special styles for the edges.

You can customize the `UIID` set by the component group by calling `setElementUUID` in the component group e.g. `setElementUUID("ToggleButton")` for three elements result in the following `UIID`s:

`ToggleButtonFirst`, `ToggleButton`, `ToggleButtonLast`

6.11. MultiButton

`MultiButton` [https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html] is a

composite component (lead component) that acts like a versatile **Button** [<https://www.codenameone.com/javadoc/com/codename1/ui/Button.html>]. It supports up to 4 lines of text (it doesn't automatically wrap the text), an emblem (usually navigational arrow, or check box) and an icon.

MultiButton can be used as a button, a **CheckBox** or a **RadioButton** for creating rich UI's.



The **MultiButton** was inspired by the aesthetics of the **UITableView** iOS component.

A common source of confusion in the **MultiButton** is the difference between the icon and the emblem, since both may have an icon image associated with them. The icon is an image representing the entry while the emblem is an optional visual representation of the action that will be undertaken when the element is pressed. Both may be used simultaneously or individually of one another.

```
MultiButton twoLinesNoIcon = new MultiButton("MultiButton");
twoLinesNoIcon.setTextLine2("Line 2");
MultiButton oneLineIconEmblem = new MultiButton("Icon + Emblem");
oneLineIconEmblem.setIcon(icon);
oneLineIconEmblem.setEmblem(emblem);
MultiButton twoLinesIconEmblem = new MultiButton("Icon + Emblem");
twoLinesIconEmblem.setIcon(icon);
twoLinesIconEmblem.setEmblem(emblem);
twoLinesIconEmblem.setTextLine2("Line 2");

MultiButton twoLinesIconEmblemHorizontal = new MultiButton("Icon + Emblem");
twoLinesIconEmblemHorizontal.setIcon(icon);
twoLinesIconEmblemHorizontal.setEmblem(emblem);
twoLinesIconEmblemHorizontal.setTextLine2("Line 2 Horizontal");
twoLinesIconEmblemHorizontal.setHorizontalLayout(true);

MultiButton twoLinesIconCheckBox = new MultiButton("CheckBox");
twoLinesIconCheckBox.setIcon(icon);
twoLinesIconCheckBox.setCheckBox(true);
twoLinesIconCheckBox.setTextLine2("Line 2");

MultiButton fourLinesIcon = new MultiButton("With Icon");
fourLinesIcon.setIcon(icon);
fourLinesIcon.setTextLine2("Line 2");
fourLinesIcon.setTextLine3("Line 3");
fourLinesIcon.setTextLine4("Line 4");

hi.add(oneLineIconEmblem).
    add(twoLinesNoIcon).
    add(twoLinesIconEmblem).
    add(twoLinesIconEmblemHorizontal).
    add(twoLinesIconCheckBox).
    add(fourLinesIcon);
```

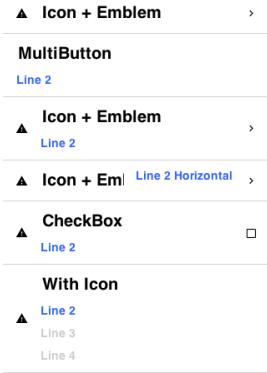


Figure 216. Multiple usage scenarios for the MultiButton

6.11.1. Styling The MultiButton

Since the **MultiButton** is a composite component setting its **UIID** will only impact the top level UI.

To customize everything you need to customize the UIID's for **MultiLine1**, **MultiLine2**, **MultiLine3**, **MultiLine4** & **Emblem**.

You can customize the individual **UIID**'s thru the API directly using the **setIconUIID**, **setUIIDL1ne1**, **setUIIDL2**, **setUIIDL3**, **setUIIDL4** & **setEmblemUIID**.

6.12. SpanButton

SpanButton [https://www.codenameone.com/javadoc/com/codename1/components/SpanButton.html] is a composite component (lead component) that looks/acts like a **Button** but can break lines rather than crop them when the text is very long.

Unlike the **MultiButton** it uses the **TextArea** internally to break lines seamlessly. The **SpanButton** is far simpler than the **MultiButton** and as a result isn't as configurable.

```
SpanButton sb = new SpanButton("SpanButton is a composite component (lead component) that looks/acts like a Button but can break lines rather than crop them when the text is very long.");
sb.setIcon(icon);
hi.add(sb);
```

SpanButton is a composite component (lead component) that looks/acts like a Button but can break lines rather than crop them when the text is very long.

Figure 217. The SpanButton Component



SpanButton is slower than both **Button** and **MultiButton**. We recommend using it only when there is a genuine need for its functionality.

6.13. SpanLabel

SpanLabel [https://www.codenameone.com/javadoc/com/codename1/components/SpanLabel.html] is a composite component (lead component) that looks/acts like a **Label** [https://www.codenameone.com/

javadoc/com/codename1/ui/Label.html] but can break lines rather than crop them when the text is very long.

SpanLabel uses the **TextArea** internally to break lines seamlessly and so doesn't provide all the elaborate configuration options of **Label**.

One of the features of label that moved into **SpanLabel** to some extent is the ability to position the icon. However, unlike a **Label** the icon position is determined by the layout manager of the composite so **setIconPosition** accepts a **BorderLayout** constraint.

```
SpanLabel d = new SpanLabel("Default SpanLabel that can seamlessly line break when the text is really long.");
d.setIcon(icon);
SpanLabel l = new SpanLabel("NORTH Positioned Icon SpanLabel that can seamlessly line break when the text is really long.");
l.setIcon(icon);
l.setIconPosition(BorderLayout.NORTH);
SpanLabel r = new SpanLabel("SOUTH Positioned Icon SpanLabel that can seamlessly line break when the text is really long.");
r.setIcon(icon);
r.setIconPosition(BorderLayout.SOUTH);
SpanLabel c = new SpanLabel("EAST Positioned Icon SpanLabel that can seamlessly line break when the text is really long.");
c.setIcon(icon);
c.setIconPosition(BorderLayout.EAST);
hi.add(d).add(l).add(r).add(c);
```

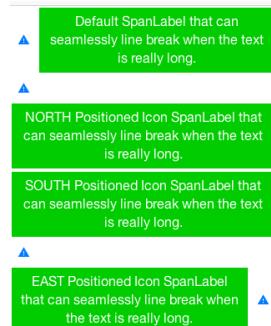


Figure 218. The **SpanLabel** Component



SpanLabel is significantly slower than **Label**. We recommend using it only when there is a genuine need for its functionality.

6.14. OnOffSwitch

The **OnOffSwitch** [https://www.codenameone.com/javadoc/com/codename1/components/OnOffSwitch.html] allows you to write an application where the user can swipe a switch between two states (on/off). This is a common UI paradigm in Android and iOS, although it's implemented in a radically different way in both platforms.

This is a rather elaborate component because of its very unique design on iOS, but we're able to accommodate most of the small behaviors of the component into our version, and it seamlessly adapts between the Android style and the iOS style.

The image below was generated based on the default use of the `OnOffSwitch`:

```
OnOffSwitch onOff = new OnOffSwitch();
hi.add(onOff);
```



Figure 219. The `OnOffSwitch` component as it appears on/off on iOS (top) and on Android (bottom)

As you can understand the difference between the way iOS and Android render this component has triggered two very different implementations within a single component. The Android implementation just uses standard buttons and is the default for non-iOS platforms.



You can force the Android or iOS mode by using the theme constant `onOffIOSModeBool`.

6.14.1. Validation

Validation is an inherent part of text input, and the `Validator` [<https://www.codenameone.com/javadoc/com/codename1/ui/validation/Validator.html>] class allows just that. You can enable validation thru the `Validator` class to add constraints for a specific component. It's also possible to define components that would be enabled/disabled based on validation state and the way in which validation errors are rendered (change the components `UIID`, paint an emblem on top, etc.). A `Constraint` [<https://www.codenameone.com/javadoc/com/codename1/ui/validation/Constraint.html>] is an interface that represents validation requirements. You can define a constraint in Java or use some of the builtin constraints such as `LengthConstraint` [<https://www.codenameone.com/javadoc/com/codename1/ui/validation/LengthConstraint.html>], `RegexConstraint` [<https://www.codenameone.com/javadoc/com/codename1/ui/validation/RegexConstraint.html>], etc.

This sample below continues from the place where the `TextField` sample above stopped by adding validation to that code.

```

Validator v = new Validator();
v.addConstraint(firstName, new LengthConstraint(2)).
    addConstraint(surname, new LengthConstraint(2)).
    addConstraint(url, RegexConstraint.validURL()).
    addConstraint(email, RegexConstraint.validEmail()).
    addConstraint(phone, new RegexConstraint(phoneRegex, "Must be valid phone number")).
    addConstraint(num1, new LengthConstraint(4)).
    addConstraint(num2, new LengthConstraint(4)).
    addConstraint(num3, new LengthConstraint(4)).
    addConstraint(num4, new LengthConstraint(4));

v.addSubmitButtons(submit);

```

Figure 220. Validation & Regular Expressions

6.15. InfiniteProgress

The **InfiniteProgress** [<https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html>] indicator spins an image infinitely to indicate that a background process is still working.



This style of animation is often nicknamed "washing machine" as it spins endlessly.

InfiniteProgress can be used in one of two ways either by embedding the component into the UI thru something like this:

```
myContainer.add(new InfiniteProgress());
```

InfiniteProgress can also appear over the entire screen, thus blocking all input. This tints the background while the infinite progress rotates:

```

Dialog ip = new InfiniteProgress().showInifiniteBlocking();

// do some long operation here using invokeAndBlock or do something in a separate thread and callback later
// when you are done just call

ip.dispose();

```



Figure 221. Infinite progress

The image used in the **InfiniteProgress** animation is defined by the native theme. You can override that definition either by defining the theme constant `infiniteImage` or by invoking the `setAnimation` [<https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html#setAnimation-com.codename1.ui.Image>] method.



Despite the name of the method `setAnimation` expects a static image that will be rotated internally. Don't use an animated image.

6.16. InfiniteScrollAdapter and InfiniteContainer

[InfiniteScrollAdapter](https://www.codenameone.com/javadoc/com/codename1/components/InfiniteScrollAdapter.html) [<https://www.codenameone.com/javadoc/com/codename1/components/InfiniteScrollAdapter.html>] & [InfiniteContainer](https://www.codenameone.com/javadoc/com/codename1/ui/InfiniteContainer.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/InfiniteContainer.html>] allow us to create a scrolling effect that "never" ends with the typical **Container / Component** paradigm.

The motivation behind these classes is simple, say we have a lot of data to fetch from storage or from the internet. We can fetch the data in batches and show progress indication while we do this.

Infinite scroll fetches the next batch of data dynamically as we reach the end of the **Container**. **InfiniteScrollAdapter** & **InfiniteContainer** represent two similar ways to accomplish that task relatively easily.

Let start by exploring how we can achieve this UI that fetches data from a webservice:

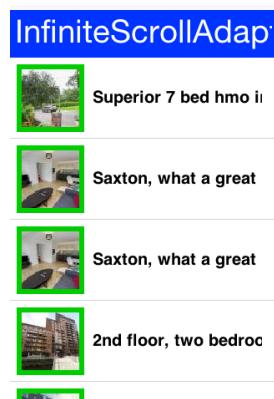


Figure 222. InfiniteScrollAdapter demo code fetching property cross data

The first step is creating the webservice call, we won't go into too much detail here as webservices & IO are discussed later in the guide:

```

int pageNumber = 1;
java.util.List<Map<String, Object>> fetchPropertyData(String text) {
    try {
        ConnectionRequest r = new ConnectionRequest();
        r.setPost(false);
        r.setUrl("http://api.nestoria.co.uk/api");
        r.addArgument("pretty", "0");
        r.addArgument("action", "search_listings");
        r.addArgument("encoding", "json");
        r.addArgument("listing_type", "buy");
        r.addArgument("page", "" + pageNumber);
        pageNumber++;
        r.addArgument("country", "uk");
        r.addArgument("place_name", text);
        NetworkManager.getInstance().addToQueueAndWait(r);
        Map<String, Object> result = new JSONParser().parseJSON(new InputStreamReader(new
ByteArrayInputStream(r.getResponseData()), "UTF-8"));
        Map<String, Object> response = (Map<String, Object>)result.get("response");
        return (java.util.List<Map<String, Object>>)response.get("listings");
    } catch(Exception err) {
        Log.e(err);
        return null;
    }
}

```

 The demo code here doesn't do any error handling! This is a very bad practice and it is taken here to keep the code short and readable. Proper error handling is used in the Property Cross demo.

The `fetchPropertyData` is a very simplistic tool that just fetches the next page of listings for the nestoria webservice. Notice that this method is synchronous and will block the calling thread (legally) until the network operation completes.

Now that we have a webservice lets proceed to create the UI. Check out the code annotations below:

```

Form hi = new Form("InfiniteScrollAdapter", new BoxLayout(BoxLayout.Y_AXIS));

Style s = UIManager.getInstance().getComponentStyle("MultiLine1");
FontImage p = FontImage.createMaterial(FontImage.MATERIAL_PORTRAIT, s);
EncodedImage placeholder = EncodedImage.createFromImage(p.scaled(p.getWidth() * 3, p.getHeight() * 3), false); ①

InfiniteScrollAdapter.createInfiniteScroll(hi.getContentPane(), () -> { ②
    java.util.List<Map<String, Object>> data = fetchPropertyData("Leeds"); ③
    MultiButton[] cmps = new MultiButton[data.size()];
    for(int iter = 0 ; iter < cmps.length ; iter++) {
        Map<String, Object> currentListing = data.get(iter);
        if(currentListing == null) { ④
            InfiniteScrollAdapter.addMoreComponents(hi.getContentPane(), new Component[0], false);
            return;
        }
        String thumb_url = (String)currentListing.get("thumb_url");
        String guid = (String)currentListing.get("guid");
        String summary = (String)currentListing.get("summary");
        cmps[iter] = new MultiButton(summary);
        cmps[iter].setIcon(URLImage.createToStorage(placeholder, guid, thumb_url));
    }
    InfiniteScrollAdapter.addMoreComponents(hi.getContentPane(), cmps, true); ⑤
}, true); ⑥

```

- ① Placeholder is essential for the [URLImage](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html>] class which we will discuss at a different place.
- ② The [InfiniteScrollAdapter](#) accepts a runnable which is invoked every time we reach the edge of the scrolling. We used a closure instead of the typical run() method override.
- ③ This is a blocking call, after the method completes we'll have all the data we need. Notice that this method doesn't block the EDT illegally.
- ④ If there is no more data we call the [addMoreComponents](#) method with a false argument. This indicates that there is no additional data to fetch.
- ⑤ Here we add the actual components to the end of the form. Notice that we **must not** invoke the [add/remove](#) method of [Container](#). Those might conflict with the work of the [InfiniteScrollAdapter](#).
- ⑥ We pass true to indicate that the data isn't "prefilled" so the method should be invoked immediately when the [Form](#) is first shown



Do not violate the EDT in the callback. It is invoked on the event dispatch thread and it is crucial

6.16.1. The InfiniteContainer

[InfiniteContainer](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/InfiniteContainer.html>] was introduced to simplify and remove some of the boilerplate of the [InfiniteScrollAdapter](#). It takes a more traditional approach of inheriting the Container class to provide its functionality.

Unlike the [InfiniteScrollAdapter](#) the [InfiniteContainer](#) accepts an index and amount to fetch. This is useful for tracking your position but also important since the [InfiniteContainer](#) also implements [Pull To Refresh](#) as part of its functionality.

Converting the code above to an [InfiniteContainer](#) is pretty simple we just moved all the code into

the callback `fetchComponents` method and returned the array of `Component`s as a response.

Unlike the `InfiniteScrollAdapter` we can't use the `ContentPane` directly so we have to use a `BorderLayout` and place the `InfiniteContainer` there:

```
Form hi = new Form("InfiniteContainer", new BorderLayout());  
  
Style s = UIManager.getInstance().getComponentStyle("MultiLine1");  
FontImage p = FontImage.createMaterial(FontImage.MATERIAL_PORTRAIT, s);  
EncodedImage placeholder = EncodedImage.createFromImage(p.scaled(p.getWidth() * 3, p.getHeight() * 3), false);  
  
InfiniteContainer ic = new InfiniteContainer() {  
    @Override  
    public Component[] fetchComponents(int index, int amount) {  
        java.util.List<Map<String, Object>> data = fetchPropertyData("Leeds");  
        MultiButton[] cmps = new MultiButton[data.size()];  
        for(int iter = 0 ; iter < cmps.length ; iter++) {  
            Map<String, Object> currentListing = data.get(iter);  
            if(currentListing == null) {  
                return null;  
            }  
            String thumb_url = (String)currentListing.get("thumb_url");  
            String guid = (String)currentListing.get("guid");  
            String summary = (String)currentListing.get("summary");  
            cmps[iter] = new MultiButton(summary);  
            cmps[iter].setIcon(URLImage.createToStorage(placeholder, guid, thumb_url));  
        }  
        return cmps;  
    }  
};  
hi.add(BorderLayout.CENTER, ic);
```

6.17. List, MultiList, Renderers & Models

6.17.1. InfiniteContainer/InfiniteScrollAdapter vs. List/ContainerList

Our recommendation is to always go with `Container`, `InfiniteContainer` or `InfiniteScrollAdapter`.

We recommend avoiding `List` or its subclasses/related classes specifically `ContainerList` & `MultiList`.



We recommend replacing `ComboBox` with `Picker` but that's a completely different discussion.

A `Container` with ~5000 nested containers within it can perform on par with a `List` and probably exceed its performance when used correctly.

Larger sets of data are rarely manageable on phones or tablets so the benefits for lists are dubious.

In terms of API we found that even experienced developers experienced a great deal of pain when wrangling the Swing styled lists and their stateless approach.

Since animation, swiping and other capabilities that are so common in mobile are so hard to

accomplish with lists we see no actual reason to use them.

6.17.2. Why Isn't List Deprecated?

We deprecated `ContainerList` which performs really badly and has some inherent complexity issues. `List` has some unique use cases and is still used all over Codename One.

`MultiList` is a reasonable version of `List` that is far easier to use without most of the pains related to renderer configuration.

There are cases where using `List` or `MultiList` is justified, they are just rarer than usual hence our recommendation.

6.17.3. MVC In Lists

A Codename One `List` [<https://www.codenameone.com/javadoc/com/codename1/ui/List.html>] doesn't contain components, but rather arbitrary data; this seems odd at first but makes sense. If you want a list to contain components, just use a Container.

The advantage of using a `List` in this way is that we can display it in many ways (e.g. fixed focus positions, horizontally, etc.), and that we can have more than a million entries without performance overhead. We can also do some pretty nifty things, like filtering the list on the fly or fetching it dynamically from the Internet as the user scrolls down the list. To achieve these things the list uses two interfaces: `ListModel` [<https://www.codenameone.com/javadoc/com/codename1/ui/listListModel.html>] and `ListCellRenderer`. `List` [<https://www.codenameone.com/javadoc/com/codename1/ui/List.html>] model represents the data; its responsibility is to return the arbitrary object within the list at a given offset. Its second responsibility is to notify the list when the data changes, so the list can refresh.



Think of the model as an array of objects that can notify you when it changes.

The list renderer is like a rubber stamp that knows how to draw an object from the model, it's called many times per entry in an animated list and must be very fast. Unlike standard Codename One components, it is only used to draw the entry in the model and is immediately discarded, hence it has no memory overhead, but if it takes too long to process a model value it can be a big bottleneck!



Think of the render as a translation layer that takes the "data" from the model and translates it to a visual representation.

This is all very generic, but a bit too much for most, doing a list "properly" requires some understanding. The main source of confusion for developers is the stateless nature of the list and the transfer of state to the model (e.g. a checkbox list needs to listen to action events on the list and update the model, in order for the renderer to display that state). Once you understand that it's easy.

6.17.4. Understanding MVC

Let's recap, what is MVC:

- **Model** - Represents the data for the component (list), the model can tell us exactly how many items are in it and which item resides at a given offset within the model. This differs from a simple **Vector** (or array), since all access to the model is controlled (the interface is simpler), and unlike a **Vector**/Array, the model can notify us of changes that occur within it.
- **View** - The view draws the content of the model. It is a "dumb" layer that has no notion of what is displayed and only knows how to draw. It tracks changes in the model (the model sends events) and redraws itself when it changes.
- **Controller** - The controller accepts user input and performs changes to the model, which in turn cause the view to refresh.

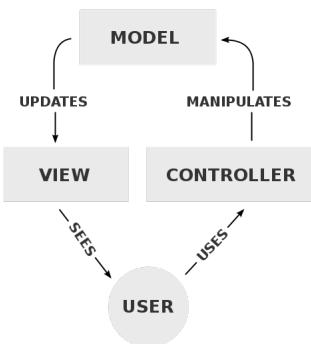


Figure 223. Typical MVC Diagram ^[3]

Codename One's [List](https://www.codenameone.com/javadoc/com/codenname1/ui/List.html) [https://www.codenameone.com/javadoc/com/codenname1/ui/List.html] component uses the MVC paradigm in its implementation. [List](#) itself is the **Controller** (with a bit of the **View** mixed in). The [ListCellRenderer](https://www.codenameone.com/javadoc/com/codenname1/ui/list/ListCellRenderer.html) [https://www.codenameone.com/javadoc/com/codenname1/ui/list/ListCellRenderer.html] interface is the rest of the **View** and the [ListModel](https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html) [https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html] is (you guessed it by now) the **Model**.

When the list is painted, it iterates over the visible elements in the model and asks the model for the data, it then draws them using the renderer. Notice that because of this both the model and the renderer must be REALLY fast and that's hard.

Why is this useful?

Since the model is a lightweight interface, it can be implemented by you and replaced in runtime if so desired, this allows several use cases:

1. A list can contain thousands of entries but only load the portion visible to the user. Since the model will only be queried for the elements that are visible to the user, it won't need to load the large data set into memory until the user starts scrolling down (at which point other elements may be offloaded from memory).
2. A list can cache efficiently. E.g. a list can mirror data from the server into local RAM without actually downloading all the data. Data can also be mirrored from storage for better performance and discarded for better memory utilization.
3. There is no need for state copying. Since renderers allow us to display any object type, the list model interface can be implemented by the application's data structures (e.g. persistence/network engine), which would return internal application data structures saving you the need of copying application state into a list specific data structure. Note that this

advantage only applies with a custom renderer which is pretty difficult to get right.

4. Using the proxy pattern we can layer logic such as filtering, sorting, caching, etc. on top of existing models without changing the model source code.
5. We can reuse generic models for several views, e.g. a model that fetches data from the server can be initialized with different arguments, to fetch different data for different views. View objects in different Forms can display the same model instance in different view instances, thus they would update automatically when we change one global model.

Most of these use cases work best for lists that grow to a larger size, or represent complex data, which is what the list object is designed to do.

6.17.5. Important - Lists & Layout Managers

Usually when working with lists, you want the list to handle the scrolling (otherwise it will perform badly). This means you should place the list in a non-scrollable container (no parent can be scrollable), notice that the content pane is scrollable by default, so you should disable that.

It is also recommended to place the list in the **CENTER** location of a **BorderLayout** [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html>] to produce the most effective results. e.g.:

```
form.setScrollable(false);
form.setLayout(new BorderLayout());
form.add(BorderLayout.CENTER, myList);
```

6.17.6. MultiList & DefaultListModel

So after this long start lets show the first sample of creating a list using the **MultiList** [<https://www.codenameone.com/javadoc/com/codename1/ui/list/MultiList.html>].

The **MultiList** is a preconfigured list that contains a ready made renderer with defaults that make sense for the most common use cases. It still retains most of the power available to the **List** component but reduces the complexity of one of the hardest things to grasp for most developers: rendering.

The full power of the **ListModel** is still available and allows you to create a million entry list with just a few lines of code. However the objects that the model returns should always be in the form of **Map** objects and not an arbitrary object like the standard **List** allows.

Here is a simple example of a **MultiList** containing a highly popular subject matter:

```

Form hi = new Form("MultiList", new BorderLayout());
ArrayList<Map<String, Object>> data = new ArrayList<>();
data.add(createListEntry("A Game of Thrones", "1996"));
data.add(createListEntry("A Clash Of Kings", "1998"));
data.add(createListEntry("A Storm Of Swords", "2000"));
data.add(createListEntry("A Feast For Crows", "2005"));
data.add(createListEntry("A Dance With Dragons", "2011"));
data.add(createListEntry("The Winds of Winter", "2016 (please, please, please)"));
data.add(createListEntry("A Dream of Spring", "Ugh"));

DefaultListModel<Map<String, Object>> model = new DefaultListModel<>(data);
MultiList ml = new MultiList(model);
hi.add(BorderLayout.CENTER, ml);

```

MultiList

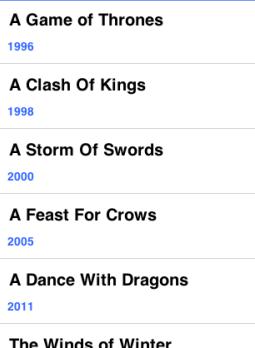


Figure 224. Basic usage of the MultiList & DefaultListModel]

`createListEntry` is relatively trivial:

```

private Map<String, Object> createListEntry(String name, String date) {
    Map<String, Object> entry = new HashMap<>();
    entry.put("Line1", name);
    entry.put("Line2", date);
    return entry;
}

```

There is one major piece missing here and that is the cover images for the books. A simple approach would be to just place the image objects into the entries using the "icon" property as such:

```

private Map<String, Object> createListEntry(String name, String date, Image cover) {
    Map<String, Object> entry = new HashMap<>();
    entry.put("Line1", name);
    entry.put("Line2", date);
    entry.put("icon", cover);
    return entry;
}

```

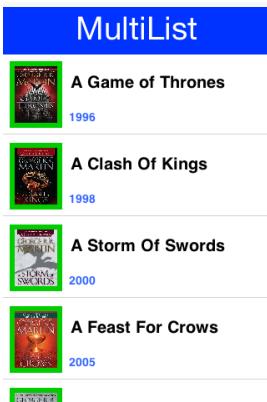


Figure 225. With cover images in place



Since the `MultiList` uses the `GenericListCellRenderer` internally we can use `URLImage` [<https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html>] to dynamically fetch the data. This is discussed in the graphics section of this guide.

Going Further With the ListModel

Lets assume that [GRRM](http://www.georgerrmartin.com/) [<http://www.georgerrmartin.com/>] was really prolific and wrote 1 million books. The default list model won't make much sense in that case but we would still be able to render everything in a list model.

We'll fake it a bit but notice that 1M components won't be created even if we somehow scroll all the way down...

The `ListModel` [<https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html>] interface can be implemented by anyone in this case we just did a really stupid simple implementation:

```
class GRMMModel implements ListModel<Map<String, Object>> {
    @Override
    public Map<String, Object> getItemAt(int index) {
        int idx = index % 7;
        switch(idx) {
            case 0:
                return createListEntry("A Game of Thrones " + index, "1996");
            case 1:
                return createListEntry("A Clash Of Kings " + index, "1998");
            case 2:
                return createListEntry("A Storm Of Swords " + index, "2000");
            case 3:
                return createListEntry("A Feast For Crows " + index, "2005");
            case 4:
                return createListEntry("A Dance With Dragons " + index, "2011");
            case 5:
                return createListEntry("The Winds of Winter " + index, "2016 (please, please, please)");
            default:
                return createListEntry("A Dream of Spring " + index, "Ugh");
        }
    }

    @Override
    public int getSize() {
        return 1000000;
```

```

}

@Override
public int getSelectedIndex() {
    return 0;
}

@Override
public void setSelectedIndex(int index) {
}

@Override
public void addDataChangedListener(DataChangedListener l) {
}

@Override
public void removeDataChangedListener(DataChangedListener l) {
}

@Override
public void addSelectionListener(SelectionListener l) {
}

@Override
public void removeSelectionListener(SelectionListener l) {
}

@Override
public void addItem(Map<String, Object> item) {
}

@Override
public void removeItem(int index) {
}
}

```

We can now replace the existing model by removing all the model related logic and changing the constructor call as such:

```
MultiList ml = new MultiList(new GRMMModel());
```

MultiList

2011

The Winds of Winter 7600

2016 (please, please, please)

A Dream of Spring 7601

Ugh

A Game of Thrones 7602

1996

A Clash Of Kings 7603

1998

A Storm Of Swords 7604

2000

Figure 226. It took ages to scroll this far... This goes to a million...

6.17.7. List Cell Renderer

The Renderer is a simple interface with 2 methods:

```
public interface ListCellRenderer {  
    //This method is called by the List for each item, when the List paints itself.  
    public Component getListCellRendererComponent(List list, Object value, int index, boolean isSelected);  
  
    //This method returns the List animated focus which is animated when list selection changes  
    public Component getListFocusComponent(List list);  
}
```

The most simple/naive implementation may choose to implement the renderer as follows:

```
public Component getListCellRendererComponent(List list, Object value, int index, boolean isSelected){  
    return new Label(value.toString());  
}  
  
public Component getListFocusComponent(List list){  
    return null;  
}
```

This will compile and work, but won't give you much, notice that you won't see the **List** selection move on the List, this is just because the renderer returns a [Label](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Label.html>] with the same style regardless if it's selected or not.

Now Let's try to make it a bit more useful.

```
public Component getListCellRendererComponent(List list, Object value, int index, boolean isSelected){  
    Label l = new Label(value.toString());  
    if (isSelected) {  
        l.setFocus(true);  
        l.getAllStyles().setBgTransparency(100);  
    } else {  
        l.setFocus(false);  
        l.getAllStyles().setBgTransparency(0);  
    }  
    return l;  
}  
public Component getListFocusComponent(List list){  
    return null;  
}
```

In this renderer we set the [Label.setFocus\(true\)](#) if it's selected, calling to this method doesn't really give the focus to the Label, it simply renders the label as selected.

Then we invoke [Label.getAllStyles\(\).setBgTransparency\(100\)](#) to give the selection semi transparency, and [0](#) for full transparency if not selected.

That is still not very efficient because we create a new [Label](#) each time the method is invoked.

To make the code tighter, keep a reference to the [Component](#) or extend it as [DefaultListCellRenderer](#)

[<https://www.codenameone.com/javadoc/com/codename1/ui/list/DefaultListCellRenderer.html>] does.

```
class MyRenderer extends Label implements ListCellRenderer {  
    public Component getListCellRendererComponent(List list, Object value, int index, boolean isSelected){  
        setText(value.toString());  
        if (isSelected) {  
            setFocus(true);  
            getAllStyles().setBgTransparency(100);  
        } else {  
            setFocus(false);  
            getAllStyles().setBgTransparency(0);  
        }  
        return this;  
    }  
}
```

Now Let's have a look at a more advanced Renderer.

```
class ContactsRenderer extends Container implements ListCellRenderer {  
  
    private Label name = new Label("");  
    private Label email = new Label("");  
    private Label pic = new Label("");  
  
    private Label focus = new Label("");  
  
    public ContactsRenderer() {  
        setLayout(new BorderLayout());  
        addComponent(BorderLayout.WEST, pic);  
        Container cnt = new Container(new BoxLayout(BoxLayout.Y_AXIS));  
        name.getAllStyles().setBgTransparency(0);  
        name.getAllStyles().setFont(Font.createSystemFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM));  
        email.getAllStyles().setBgTransparency(0);  
        cnt.addComponent(name);  
        cnt.addComponent(email);  
        addComponent(BorderLayout.CENTER, cnt);  
  
        focus.getStyle().setBgTransparency(100);  
    }  
  
    public Component getListCellRendererComponent(List list, Object value, int index, boolean isSelected) {  
  
        Contact person = (Contact) value;  
        name.setText(person.getName());  
        email.setText(person.getEmail());  
        pic.setIcon(person.getPic());  
        return this;  
    }  
  
    public Component getListFocusComponent(List list) {  
        return focus;  
    }  
}
```

In this renderer we want to render a **Contact** object to the Screen, we build the **Component** in the

constructor and in the getListCellRendererComponent we simply update the Labels' texts according to the **Contact** object.

Notice that in this renderer we return a focus **Label** with semi transparency, as mentioned before, the focus component can be modified within this method.

For example, I can modify the focus **Component** to have an icon.

```
focus.getAllStyles().setBgTransparency(100);
try {
    focus.setIcon(Image.createImage("/duke.png"));
    focus.setAlignment(Component.RIGHT);
} catch (IOException ex) {
    ex.printStackTrace();
}
```

6.17.8. Generic List Cell Renderer

As part of the GUI builder work, we needed a way to customize rendering for a List, but the renderer/model approach seemed impossible to adapt to a GUI builder (it seems the Swing GUI builders had a similar issue). Our solution was to introduce the **GenericListCellRenderer**, which while introducing limitations and implementation requirements still manages to make life easier, both in the GUI builder and outside of it.

GenericListCellRenderer [https://www.codenameone.com/javadoc/com/codenname1/ui/list/GenericListCellRenderer.html] is a renderer designed to be as simple to use as a **Component-Container** hierarchy, we effectively crammed most of the common renderer use cases into one class. To enable that, we need to know the content of the objects within the model, so the **GenericListCellRenderer** assumes the model contains only **Map** objects. Since **Maps** can contain arbitrary data the list model is still quite generic and allows storing application specific data. Furthermore a **Map** can still be derived and extended to provide domain specific business logic.

The **GenericListCellRenderer** accepts two container instances (more later on why at least two, and not one), which it maps to individual **Map** entries within the model, by finding the appropriate components within the given container hierarchy. Components are mapped to the **Map** entries based on the name property of the component (**getName/setName**) and the key/value within the **Map**, e.g.:

For a model that contains a **Map** entry like this:

```
"Foo": "Bar"
"X": "Y"
"Not": "Applicable"
"Number": Integer(1)
```

A renderer will loop over the component hierarchy in the container, searching for components whose name matches Foo, X, Not, and Number, and assigning the appropriate value to them.



You can also use image objects as values, and they will be assigned to labels as expected. However, you can't assign both an image and a text to a single label, since the key will be taken. That isn't a big problem, since two labels can be used quite easily in such a renderer.

To make matters even more attractive the renderer seamlessly supports list tickering when appropriate, and if a [CheckBox](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html>] appears within the renderer, it will toggle a boolean flag within the [Map](#) seamlessly.

One issue that crops up with this approach is that, if a value is missing from the [Map](#), it is treated as empty and the component is reset.

This can pose an issue if we hardcode an image or text within the renderer and we don't want them replaced (e.g. an arrow graphic on a [Label](#) within the renderer). The solution for this is to name the component with Fixed in the end of the name, e.g. [HardcodedIconFixed](#).

Naming a component within the renderer with \$number will automatically set it as a counter component for the offset of the component within the list.

Styling the [GenericListCellRenderer](#) is slightly different, the renderer uses the [UUID](#) of the [Container](#) passed to the generic list cell renderer, and the background focus uses that same [UUID](#) with the word "Focus" appended to it.

It is important to notice that the generic list cell renderer will grant focus to the child components of the selected entry if they are focusable, thus changing the style of said entries. E.g. a [Container](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Container.html>] might have a child [Label](#) that has one style when the parent container is unselected and another when it's selected (focused), this can be easily achieved by defining the label as focusable. Notice that the component will never receive direct focus, since it is still part of a renderer.

Last but not least, the generic list cell renderer accepts two or four instances of a Container, rather than the obvious choice of accepting only one instance. This allows the renderer to treat the selected entry differently, which is especially important to tickering, although it's also useful for the fisheye effect ^[4]. Since it might not be practical to seamlessly clone the [Container](#) for the renderer's needs, Codename One expects the developer to provide two separate instances, they can be identical in all respects, but they must be separate instances for tickering to work. The renderer also allows for a fisheye effect, where the selected entry is actually different from the unselected entry in its structure, it also allows for a pinstripe effect, where odd/even rows have different styles (this is accomplished by providing 4 instances of the containers selected/unselected for odd/even).

The best way to learn about the generic list cell renderer and the [Map](#) model is by playing with them in the old GUI builder. Notice they can be used in code without any dependency on the GUI builder and can be quite useful at that.

Here is a simple example of a list with checkboxes that gets updated automatically:

```

com.codename1.ui.List list = new com.codename1.ui.List(createGenericListCellRendererModelData());
list.setRenderer(new GenericListCellRenderer(createGenericRendererContainer(), createGenericRendererContainer()));

private Container createGenericRendererContainer() {
    Label name = new Label();
    name.setFocusable(true);
    name.setName("Name");
    Label surname = new Label();
    surname.setFocusable(true);
    surname.setName("Surname");
    CheckBox selected = new CheckBox();
    selected.setName("Selected");
    selected.setFocusable(true);
    Container c = BorderLayout.center(name).
        add(BorderLayout.SOUTH, surname).
        add(BorderLayout.WEST, selected);
    c.setUIID("ListRenderer");
    return c;
}

private Object[] createGenericListCellRendererModelData() {
    Map<String, Object>[] data = new HashMap[5];
    data[0] = new HashMap<>();
    data[0].put("Name", "Shai");
    data[0].put("Surname", "Almog");
    data[0].put("Selected", Boolean.TRUE);
    data[1] = new HashMap<>();
    data[1].put("Name", "Chen");
    data[1].put("Surname", "Fishbein");
    data[1].put("Selected", Boolean.TRUE);
    data[2] = new HashMap<>();
    data[2].put("Name", "Ofir");
    data[2].put("Surname", "Leitner");
    data[3] = new HashMap<>();
    data[3].put("Name", "Yaniv");
    data[3].put("Surname", "Vakarat");
    data[4] = new HashMap<>();
    data[4].put("Name", "Meirav");
    data[4].put("Surname", "Nachmanovitch");
    return data;
}

```

GenericListCelRe



Figure 227. GenericListCellRenderer demo code

Custom UIID Of Entry in GenenricListCellRenderer/MultiList

With [MultiList](#) [https://www.codenameone.com/javadoc/com/codenname1/ui/list/MultiList.html] one of the common issues is making a UI where a specific component within the list renderer has a different UIID style based on data. E.g. this can be helpful to mark a label within the list as red, for instance, in the case of a list of monetary transactions.

This can be achieved with a custom renderer, but that is a pretty difficult task.

[GenericListCellRenderer](#) ([MultiList](#) uses [GenericListCellRenderer](#) internally) has another option.

Normally, to build the model for a renderer of this type, we use something like:

```
map.put("componentName", "Component Value");
```

What if we want componentName to be red? Just use:

```
map.put("componentName_uuid", "red");
```

This will apply the UIID "red" to the component, which you can then style in the theme. Notice that once you start doing this, you need to define this entry for all entries, e.g.:

```
map.put("componentName_uuid", "blue");
```

Otherwise the component will stay red for the next entry (since the renderer acts like a rubber stamp).

Rendering Prototype

Because of the rendering architecture of a [List](#) its pretty hard to calculate the right preferred size for such a component. The default behavior includes querying a few entries from the model then constructing their renderers to get a "sample" of the preferred size value.

As you might guess this triggers a performance penalty that is paid with every reflow of the UI. The solution is to use [setRenderingPrototype](#).

[setRenderingPrototype](#) accepts a "fake" value that represents a reasonably large amount of data and it will be used to calculate the preferred size. E.g. for a multiList that should render 2 lines of text with 20 characters and a 5mm square icon I can do something like this:

```
Map<String, Object> proto = new HashMap<>();
map.put("Line1", "WWWWWWWWWWWWWWWWWW");
map.put("Line2", "WWWWWWWWWWWWWWWWWW");
int mm5 = Display.getInstance().convertToPixels(5, true);
map.put("icon", Image.create(mm5, mm5));
myMultiList.setRenderingPrototype(map);
```

6.17.9. ComboBox

The [ComboBox](https://www.codenameone.com/javadoc/com/codename1/ui/ComboBox.html) [https://www.codenameone.com/javadoc/com/codename1/ui/ComboBox.html] is a specialization of [List](#) that displays a single selected entry. When clicking that entry a popup is presented allowing the user to pick an entry from the full list of entries.



The [ComboBox](#) UI paradigm isn't as common on OS's such as iOS where there is no native equivalent to it. We recommend using either the [Picker](https://www.codenameone.com/javadoc/com/codename1/ui/spinner/Picker.html) [https://www.codenameone.com/javadoc/com/codename1/ui/spinner/Picker.html] class or the [AutoCompleteTextField](https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html) [https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html].

[ComboBox](#) is notoriously hard to style properly as it relies on a complex dynamic of popup renderer and instantly visible renderer. The [UUID](#) for the [ComboBox](#) is [ComboBox](#) however if you set it to something else all the other [UUIDs](#) will also change their prefix. E.g. the [ComboBoxPopup](#) [UUID](#) will become [MyNewUUIDPopup](#).

The combo box defines the following [UUID](#)'s by default:

- [ComboBox](#)
- [ComboBoxItem](#)
- [ComboBoxFocus](#)
- [PopupContentPane](#)
- [PopupItem](#)
- [PopupFocus](#)

The [ComboBox](#) also defines theme constants that allow some native themes to manipulate its behavior e.g.:

- [popupTitleBool](#) - shows the "label for" value as the title of the popup dialog
- [popupCancelBodyBool](#) - Adds a cancel button into the popup dialog
- [centeredPopupBool](#) - shows the popup dialog in the center of the screen instead of under the popup
- [otherPopupRendererBool](#) - Uses a different list cell render for the popup than the one used for the [ComboBox](#) itself. When this is [false](#) [PopupItem](#) & [PopupFocus](#) become irrelevant. Notice that the Android native theme defines this to [true](#).

Since a [ComboBox](#) is really a [List](#) you can use everything we learned about a [List](#) to build a [ComboBox](#) including models, [GenericListCellRenderer](#) etc.

E.g. the demo below uses the GRRM demo data from above to build a [ComboBox](#):

```

Form hi = new Form("ComboBox", new BoxLayout(BoxLayout.Y_AXIS));
ComboBox<Map<String, Object>> combo = new ComboBox<>(
    createListEntry("A Game of Thrones", "1996"),
    createListEntry("A Clash Of Kings", "1998"),
    createListEntry("A Storm Of Swords", "2000"),
    createListEntry("A Feast For Crows", "2005"),
    createListEntry("A Dance With Dragons", "2011"),
    createListEntry("The Winds of Winter", "2016 (please, please, please)"),
    createListEntry("A Dream of Spring", "Ugh"));

combo.setRenderer(new GenericListCellRenderer<>(new MultiButton(), new MultiButton()));

```



Figure 228. GRRM ComboBox

6.18. Slider

A [Slider](#) [https://www.codenameone.com/javadoc/com/codename1/ui/Slider.html] is an empty component that can be filled horizontally or vertically to allow indicating progress, setting volume etc. It can be editable to allow the user to determine its value or none editable to just relay that information to the user. It can have a thumb on top to show its current position.



Figure 229. Slider

The interesting part about the slider is that it has two separate style [UIID's](#), [Slider](#) & [SliderFull](#). The [Slider UIID](#) is always painted and [SliderFull](#) is rendered on top based on the amount the [Slider](#) should be filled.

[Slider](#) is highly customizable e.g. a slider can be used to replicate a 5 star rating widget as such. Notice that this slider will only work when its given its preferred size otherwise additional stars will appear. That's why we place it within a [FlowLayout](#):

```

Form hi = new Form("Star Slider", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(FlowLayout.encloseCenter(createStarRankSlider()));
hi.show();

```

The slider itself is initialized in the code below. Notice that you can achieve almost the same result using a theme by setting the [Slider](#) & [SliderFull](#) UIID's (both in selected & unselected states).

In fact doing this in the theme might be superior as you could use one image that contains 5 stars already and that way you won't need the preferred size hack below:

```

private void initStarRankStyle(Style s, Image star) {
    s.setBackgroundType(Style.BACKGROUND_IMAGE_TILE_BOTH);
    s.setBorder(Border.createEmpty());
    s.setBgImage(star);
    s.setBgTransparency(0);
}

private Slider createStarRankSlider() {
    Slider starRank = new Slider();
    starRank.setEditable(true);
    starRank.setMinValue(0);
    starRank.setMaxValue(10);
    Font fnt = Font.createTrueTypeFont("native:MainLight", "native:MainLight").
        derive(Display.getInstance().convertToPixels(5, true), Font.STYLE_PLAIN);
    Style s = new Style(0xfffff33, 0, fnt, (byte)0);
    Image fullStar = FontImage.createMaterial(FontImage.MATERIAL_STAR, s).toImage();
    s.setOpacity(100);
    s.setFgColor(0);
    Image emptyStar = FontImage.createMaterial(FontImage.MATERIAL_STAR, s).toImage();
    initStarRankStyle(starRank.getSliderEmptySelectedStyle(), emptyStar);
    initStarRankStyle(starRank.getSliderEmptyUnselectedStyle(), emptyStar);
    initStarRankStyle(starRank.getSliderFullSelectedStyle(), fullStar);
    initStarRankStyle(starRank.getSliderFullUnselectedStyle(), fullStar);
    starRank.setPreferredSize(new Dimension(fullStar.getWidth() * 5, fullStar.getHeight()));
    return starRank;
}

private void showStarPickingForm() {
    Form hi = new Form("Star Slider", new BoxLayout(BoxLayout.Y_AXIS));
    hi.add(FlowLayout.encloseCenter(createStarRankSlider()));
    hi.show();
}

```

Star Slider



Figure 230. Star Slider set to 5 (its between 0 - 10)



This slider goes all the way to 0 stars which is less common. You can use a [Label](#) to represent the first star and have the slider work between 0 - 8 values to provide 4 additional stars.

6.19. Table

[Table](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html>] is a composite component (but it isn't a [lead component](#)), this means it is a subclass of [Container](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Container.html>]. It's effectively built from multiple components.



Table is heavily based on the **TableLayout** [<https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.html>] class. It's important to be familiar with that layout manager when working with **Table**.

Here is a trivial sample of using the standard table component:

```
Form hi = new Form("Table", new BorderLayout());
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col 2", "Col 3"}, new Object[][][] {
    {"Row 1", "Row A", "Row X"},  

    {"Row 2", "Row B", "Row Y"},  

    {"Row 3", "Row C", "Row Z"},  

    {"Row 4", "Row D", "Row K"},  

}) {
    public boolean isCellEditable(int row, int col) {
        return col != 0;
    }
};
Table table = new Table(model);
hi.add(BorderLayout.CENTER, table);
hi.show();
```

Col 1	Col 2	Col 3
Row 1	Row A	Row X
Row 2	Row B	Row Y
Row 3	Row C	Row Z
Row 4	Row D	Row K

Figure 231. Simple Table usage



In the sample above the title area and first column aren't editable. The other two columns are editable.

The more "interesting" capabilities of the **Table** class can be utilized via the **TableLayout**. You can use the layout constraints (also exposed in the table class) to create spanning and elaborate UI's.

E.g.:

```

Form hi = new Form("Table", new BorderLayout());
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col 2", "Col 3"}, new Object[][] {
    {"Row 1", "Row A", "Row X"},  

    {"Row 2", "Row B can now stretch", null},  

    {"Row 3", "Row C", "Row Z"},  

    {"Row 4", "Row D", "Row K"},  

}) {  

    public boolean isCellEditable(int row, int col) {  

        return col != 0;  

    }  

};  

Table table = new Table(model) {  

    @Override  

    protected TableLayout.Constraint createCellConstraint(Object value, int row, int column) {  

        TableLayout.Constraint con = super.createCellConstraint(value, row, column);  

        if(row == 1 && column == 1) {  

            con.setHorizontalSpan(2);  

        }  

        con.setWidthPercentage(33);  

        return con;  

    }  

};  

hi.add(BorderLayout.CENTER, table);

```

Table		
Col 1	Col 2	Col 3
Row 1	Row A	Row X
Row 2	Row B can now stretch	
Row 3	Row C	Row Z
Row 4	Row D	Row K

Figure 232. Table with spanning & fixed widths to 33%

In order to customize the table cell behavior you can derive the **Table** to create a "renderer like" widget, however unlike the list this component is "kept" and used as is. This means you can bind listeners to this component and work with it as you would with any other component in Codename One.

So lets fix the example above to include far more capabilities:

```

Table table = new Table(model) {
    @Override
    protected Component createCell(Object value, int row, int column, boolean editable) { ①
        Component cell;
        if(row == 1 && column == 1) { ②
            Picker p = new Picker();
            p.setType(Display.PICKER_TYPE_STRINGS);
            p.setStrings("Row B can now stretch", "This is a good value", "So Is This", "Better than text field");
            p.setSelectedString((String)value); ③
            p.setUUID("TableCell");
            p.addActionListener((e) -> getModel().setValueAt(row, column, p.getSelectedString())); ④
            cell = p;
        } else {
            cell = super.createCell(value, row, column, editable);
        }
        if(row > -1 && row % 2 == 0) { ⑤
            // pinstripe effect
            cell.getAllStyles().setBgColor(0xeeeeee);
            cell.getAllStyles().setBgTransparency(255);
        }
        return cell;
    }

    @Override
    protected TableLayout.Constraint createCellConstraint(Object value, int row, int column) {
        TableLayout.Constraint con = super.createCellConstraint(value, row, column);
        if(row == 1 && column == 1) {
            con.setHorizontalSpan(2);
        }
        con.setWidthPercentage(33);
        return con;
    }
};

```

- ① The `createCell` method is invoked once per component but is similar conceptually to the `List` renderer. Notice that it doesn't return a "rubber stamp" though, it returns a full component.
- ② We only apply the picker to one cell for simplicities sake.
- ③ We need to set the value of the component manually, this is crucial since the `Table` doesn't "see" this.
- ④ We need to track the event and update the model in this case as the `Table` isn't aware of the data change.
- ⑤ We set the "pinstripe" effect by coloring even rows. Notice that unlike renderers we only need to apply the coloring once as the `Components` are stateful.

Table		
Col 1	Col 2	Col 3
Row 1	Row A	Row X
Row 2	Row B can now stretch	
Row 3	Row C	Row Z
Row 4	Row D	Row K

Figure 233. Table with customize cells using the pinstripe effect

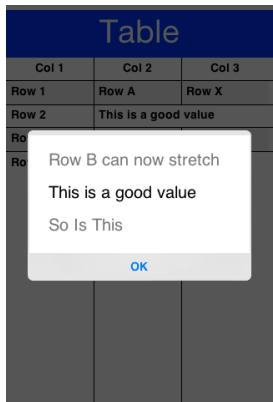


Figure 234. Picker table cell during edit

To line wrap table cells we can just override the `createCell` method and return a `TextArea` [<https://www.codenameone.com/javadoc/com/codename1/ui/TextArea.html>] instead of a `TextField` [<https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html>] since the `TextArea` defaults to the multi-line behavior this should work seamlessly. E.g.:

```

Form hi = new Form("Table", new BorderLayout());
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col 2", "Col 3"}, new Object[][] {
    {"Row 1", "Row A", "Row X"},  

    {"Row 2", "Row B can now stretch very long line that should span multiple rows as much as possible", "Row Y"},  

    {"Row 3", "Row C", "Row Z"},  

    {"Row 4", "Row D", "Row K"},  

}) {  

    public boolean isCellEditable(int row, int col) {  

        return col != 0;  

    }  

};  

Table table = new Table(model) {  

    @Override  

    protected Component createCell(Object value, int row, int column, boolean editable) {  

        TextArea ta = new TextArea((String)value);  

        ta.setUIID("TableCell");  

        return ta;  

}  

    @Override  

    protected TableLayout.Constraint createCellConstraint(Object value, int row, int column) {  

        TableLayout.Constraint con = super.createCellConstraint(value, row, column);  

        con.setWidthPercentage(33);  

        return con;  

}  

};  

hi.add(BorderLayout.CENTER, table);  

hi.show();

```



Notice that we don't really need to do anything else as binding to the `TextArea` is builtin to the `Table`.



We must set the column width constraint when we want multi-line to work. Otherwise the preferred size of the column might be too wide and the remaining columns might not have space left.

Table		
Col 1	Col 2	Col 3
Row 1	Row A	Row X
Row 2	Row B can now stretch very long line that should span multiple rows as much as possible	Row Y
Row 3	Row C	Row Z
Row 4	Row D	Row K

Figure 235. Multiline table cell in portrait mode

Table		
Col 1	Col 2	Col 3
Row 1	Row A	Row X
Row 2	Row B can now stretch very long line that should span multiple rows as much as possible	Row Y
Row 3	Row C	Row Z
Row 4	Row D	Row K

Figure 236. Multiline table cell in landscape mode. Notice the cell row count adapts seamlessly

6.19.1. Sorting Tables

Sorting tables by clicking the titles is something that should generally work out of the box by using an API like `setSortSupported(true)`.

```
Form hi = new Form("Table", new BorderLayout());
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col 2", "Col 3"}, new Object[][] {
    {"Row 1", "Row A", 1},
    {"Row 2", "Row B", 4},
    {"Row 3", "Row C", 7.5},
    {"Row 4", "Row D", 2.24},
});
Table table = new Table(model);
table.setSortSupported(true);
hi.add(BorderLayout.CENTER, table);
hi.add(NORTH, new Button("Button"));
hi.show();
```

Notice this works with numbers, Strings and might work with dates but you can generally support any object type by overriding the method `protected Comparator createColumnSortComparator(int column)` which should return a comparator for your custom object type in the column.

6.20. Tree

`Tree` [https://www.codenameone.com/javadoc/com/codename1/ui/tree/Tree.html] allows displaying hierarchical data such as folders and files in a collapsible/expandable UI. Like the `Table` it is a composite component (but it isn't a `lead component`). Like the `Table` it works in consort with a model to construct its user interface on the fly but doesn't use a stateless renderer (as `List` does).

The data of the `Tree` arrives from a model model e.g. this:

```

class StringArrayTreeModel implements TreeModel {
    String[][] arr = new String[][] {
        {"Colors", "Letters", "Numbers"},
        {"Red", "Green", "Blue"},
        {"A", "B", "C"},
        {"1", "2", "3"}
    };

    public Vector getChildren(Object parent) {
        if(parent == null) {
            Vector v = new Vector();
            for(int iter = 0 ; iter < arr[0].length ; iter++) {
                v.addElement(arr[0][iter]);
            }
            return v;
        }
        Vector v = new Vector();
        for(int iter = 0 ; iter < arr[0].length ; iter++) {
            if(parent == arr[0][iter]) {
                if(arr.length > iter + 1 && arr[iter + 1] != null) {
                    for(int i = 0 ; i < arr[iter + 1].length ; i++) {
                        v.addElement(arr[iter + 1][i]);
                    }
                }
            }
        }
        return v;
    }

    public boolean isLeaf(Object node) {
        Vector v = getChildren(node);
        return v == null || v.size() == 0;
    }
}

Tree dt = new Tree(new StringArrayTreeModel());

```

Will result in this:



Figure 237. Tree



Since **Tree** is hierarchy based we can't have a simple model like we have for the **Table** as deep hierarchy is harder to represent with arrays.

A more practical "real world" example would be working with XML data. We can use something like this to show an XML **Tree**:

```
Form hi = new Form("XML Tree", new BorderLayout());
InputStream is = Display.getInstance().getResourceAsStream(getClass(), "/build.xml");
try(Reader r = new InputStreamReader(is, "UTF-8")) {
    Element e = new XMLParser().parse(r);
    Tree xmlTree = new Tree(new XMLTreeModel(e)) {
        @Override
        protected String childToDisplayLabel(Object child) {
            if(child instanceof Element) {
                return ((Element)child).getTagName();
            }
            return child.toString();
        }
    };
    hi.add(BorderLayout.CENTER, xmlTree);
} catch(IOException err) {
    Log.e(err);
}
```



The **try(Stream)** syntax is a try with resources clogic that implicitly closes the stream.

XML Tree

```
project
description
import
property
taskdef
taskdef
taskdef
taskdef
taskdef
target
mkdir
javac
preparetests
```

Figure 238. XML Tree

The model for the XML hierarchy is implemented as such:

```

class XMLTreeModel implements TreeModel {
    private Element root;
    public XMLTreeModel(Element e) {
        root = e;
    }

    public Vector getChildren(Object parent) {
        if(parent == null) {
            Vector c = new Vector();
            c.addElement(root);
            return c;
        }
        Vector result = new Vector();
        Element e = (Element)parent;
        for(int iter = 0 ; iter < e.getNumChildren() ; iter++) {
            result.addElement(e.getChildAt(iter));
        }
        return result;
    }

    public boolean isLeaf(Object node) {
        Element e = (Element)node;
        return e.getNumChildren() == 0;
    }
}

```

6.21. ShareButton

[ShareButton](https://www.codenameone.com/javadoc/com/codename1/components/ShareButton.html) [https://www.codenameone.com/javadoc/com/codename1/components/ShareButton.html] is a button you can add into the UI to let a user share an image or block of text.

The [ShareButton](#) uses a set of predefined share options on the simulator. On Android & iOS the [ShareButton](#) is mapped to the OS native sharing functionality and can share the image/text with the services configured on the device (e.g. Twitter, Facebook etc.).



Sharing text is trivial but to share an image we need to save it to the [FileSystemStorage](#) [https://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html]. Notice that saving to Storage **won't work!**

In the sample code below we take a screenshot which is saved to [FileSystemStorage](#) [https://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html] for sharing:

```

Form hi = new Form("ShareButton");
ShareButton sb = new ShareButton();
sb.setText("Share Screenshot");
hi.add(sb);

Image screenshot = Image.createImage(hi.getWidth(), hi.getHeight());
hi.revalidate();
hi.setVisible(true);
hi.paintComponent(screenshot.getGraphics(), true);

String imageFile = FileSystemStorage.getInstance().getAppHomePath() + "screenshot.png";
try(OutputStream os = FileSystemStorage.getInstance().openOutputStream(imageFile)) {
    ImageIO.getImageIO().save(screenshot, os, ImageIO.FORMAT_PNG, 1);
} catch(IOException err) {
    Log.e(err);
}
sb.setImageToShare(imageFile, "image/png");

```

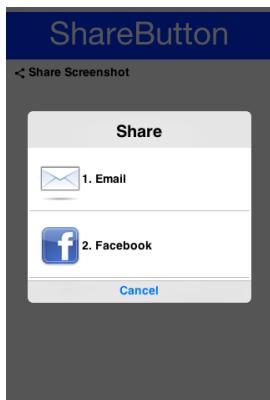


Figure 239. The share button running on the simulator



ShareButton behaves very differently on the device...

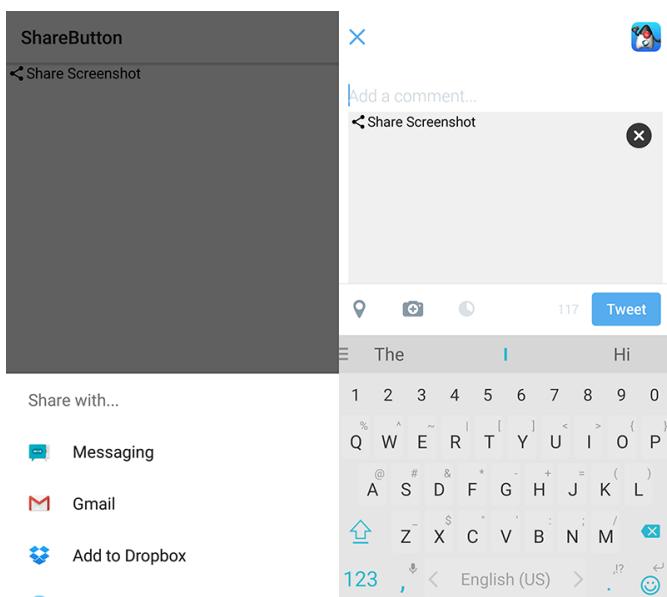


Figure 240. The share button running on the Android device and screenshot sent into twitter



The **ShareButton** features some share service classes to allow plugging in additional share services. However, this functionality is only relevant to devices where native sharing isn't supported. So this code isn't used on iOS/Android...

6.22. Tabs

The **Tabs** [https://www.codenameone.com/javadoc/com/codename1/ui/Tabs.html] **Container** arranges components into groups within "tabbed" containers. **Tabs** is a container type that allows leafing through its children using labeled toggle buttons. The tabs can be placed in multiple different ways (top, bottom, left or right) with the default being determined by the platform. This class also allows swiping between components to leaf between said tabs (for this purpose the tabs themselves can also be hidden).

Since **Tabs** are a **Container** its a common mistake to try and add a **Tab** using the **add** method. That method won't work since a **Tab** can have both an **Image** and text String associated with it.

```
Form hi = new Form("Tabs", new BorderLayout());  
  
Tabs t = new Tabs();  
Style s = UIManager.getInstance().getComponentStyle("Tab");  
FontImage icon1 = FontImage.createMaterial(FontImage.MATERIAL_QUESTION_ANSWER, s);  
  
Container container1 = BoxLayout.encloseY(new Label("Label1"), new Label("Label2"));  
t.addTab("Tab1", icon1, container1);  
t.addTab("Tab2", new SpanLabel("Some text directly in the tab"));  
  
hi.add(BorderLayout.CENTER, t);
```

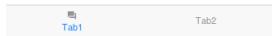


Figure 241. Simple usage of Tabs

A common usage for **Tabs** is the the swipe to proceed effect which is very common in iOS applications. In the code below we use **RadioButton** [https://www.codenameone.com/javadoc/com/codename1/ui/RadioButton.html] and **LayeredLayout** [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.html] with hidden tabs to produce that effect:

```

Form hi = new Form("Swipe Tabs", new LayeredLayout());
Tabs t = new Tabs();
t.hideTabs();

Style s = UIManager.getInstance().getComponentStyle("Button");
FontImage radioEmptyImage = FontImage.createMaterial(FontImage.MATERIAL_RADIO_BUTTON_UNCHECKED, s);
FontImage radioFullImage = FontImage.createMaterial(FontImage.MATERIAL_RADIO_BUTTON_CHECKED, s);
((DefaultLookAndFeel)UIManager.getInstance().getLookAndFeel()).setRadioButtonImages(radioFullImage, radioEmptyImage,
radioFullImage, radioEmptyImage);

Container container1 = BoxLayout.encloseY(new Label("Swipe the tab to see more"),
new Label("You can put anything here"));
t.addTab("Tab1", container1);
t.addTab("Tab2", new SpanLabel("Some text directly in the tab"));

RadioButton firstTab = new RadioButton("");
RadioButton secondTab = new RadioButton("");
firstTab.setUUID("Container");
secondTab.setUUID("Container");
new ButtonGroup(firstTab, secondTab);
firstTab.setSelected(true);
Container tabsFlow = FlowLayout.encloseCenter(firstTab, secondTab);

hi.add(t);
hi.add(BorderLayout.south(tabsFlow));

t.addSelectionListener((i1, i2) -> {
    switch(i2) {
        case 0:
            if(!firstTab.isSelected()) {
                firstTab.setSelected(true);
            }
            break;
        case 1:
            if(!secondTab.isSelected()) {
                secondTab.setSelected(true);
            }
            break;
    }
});

```



Figure 242. Swipeable Tabs with an iOS carousel effect page 1

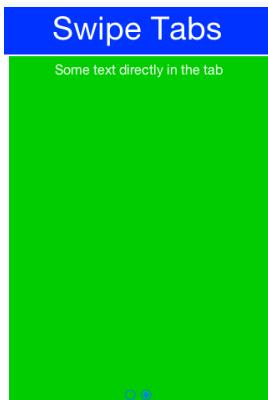


Figure 243. Swipeable Tabs with an iOS carousel effect page 2



Notice that we used `setRadioButtonImages` to explicitly set the radio button images to the look we want for the carousel.

6.23. MediaManager & MediaPlayer



MediaPlayer is a **peer component**, understanding this is crucial if your application depends on such a component. You can learn about peer components and their issues [here](#) [<https://www.codenameone.com/manual/advanced-topics.html#native-peer-components>].

The **MediaPlayer** [<https://www.codenameone.com/javadoc/com/codename1/components/MediaPlayer.html>] allows you to control video playback. To use the **MediaPlayer** we need to first load the **Media** object from the **MediaManager** [<https://www.codenameone.com/javadoc/com/codename1/media/MediaManager.html>].

The **MediaManager** is the core class responsible for media interaction in Codename One.



You should also check out the **Capture** [<https://www.codenameone.com/javadoc/com/codename1/capture/Capture.html>] class for things that aren't covered by the **MediaManager**.

In the demo code below we use the gallery functionality to pick a video from the device's video gallery.

```

final Form hi = new Form("MediaPlayer", new BorderLayout());
hi.setToolbar(new Toolbar());
Style s = UIManager.getInstance().getComponentStyle("Title");
FontImage icon = FontImage.createMaterial(FontImage.MATERIAL_VIDEO_LIBRARY, s);
hi.getToolbar().addCommandToRightBar("", icon, evt) -> {
    Display.getInstance().openGallery((e) -> {
        if(e != null && e.getSource() != null) {
            String file = (String)e.getSource();
            try {
                Media video = MediaManager.createMedia(file, true);
                hi.removeAll();
                hi.add(BorderLayout.CENTER, new MediaPlayer(video));
                hi.revalidate();
            } catch(IOException err) {
                Log.e(err);
            }
        }
    }, Display.GALLERY_VIDEO);
});
hi.show();

```



Figure 244. Video playback running on the simulator

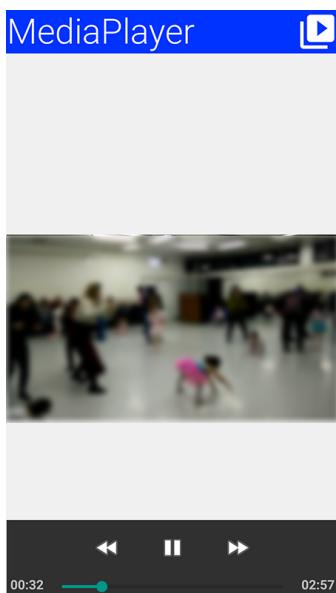


Figure 245. Video playback running on an Android device. Notice the native playback controls that appear when the video is tapped



Video playback in the simulator will only work with JavaFX enabled. This is the default for Java 8 or newer so we recommend using that.

6.24. ImageViewer

The [ImageViewer](https://www.codenameone.com/javadoc/com/codename1/components/ImageViewer.html) [https://www.codenameone.com/javadoc/com/codename1/components/ImageViewer.html] allows us to inspect, zoom and pan into an image. It also allows swiping between images if you have a set of images (using an image list model).



The [ImageViewer](#) is a complex rich component designed for user interaction. If you just want to display an image use [Label](https://www.codenameone.com/javadoc/com/codename1/ui/Label.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Label.html] if you want the image to scale seamlessly use [ScaleImageLabel](https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageLabel.html) [https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageLabel.html].

You can use the [ImageViewer](#) as a tool to view a single image which allows you to zoom in/out to that image as such:

```
Form hi = new Form("ImageViewer", new BorderLayout());
ImageViewer iv = new ImageViewer(duke);
hi.add(BorderLayout.CENTER, iv);
```



You can simulate pinch to zoom on the simulator by dragging the right button away from the top left corner to zoom in and towards the top left corner to zoom out. On Mac touchpads you can drag two fingers to achieve that.

ImageViewer



Figure 246. ImageViewer as the demo loads with the image from the default icon



Figure 247. ImageViewer zoomed in

We can work with a list of images to produce a swiping effect for the image viewer where you can swipe from one image to the next and also zoom in/out on a specific image:

```
Form hi = new Form("ImageViewer", new BorderLayout());  
  
Image red = Image.createImage(100, 100, 0xffff0000);  
Image green = Image.createImage(100, 100, 0xff00ff00);  
Image blue = Image.createImage(100, 100, 0xff0000ff);  
Image gray = Image.createImage(100, 100, 0xffcccccc);  
  
ImageViewer iv = new ImageViewer(red);  
iv.setImageList(new DefaultListModel<>(red, green, blue, gray));  
hi.add(BorderLayout.CENTER, iv);
```

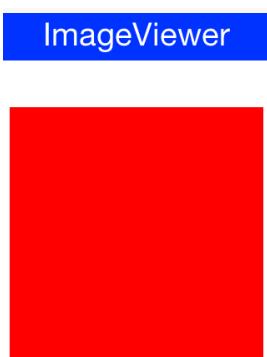


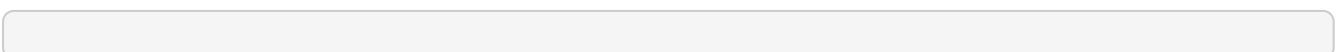
Figure 248. An ImageViewer with multiple elements is indistinguishable from a single ImageViewer with the exception of swipe

Notice that we use a [ListModel](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html>] to allow swiping between images.



EncodedImages aren't always fully loaded and so when you swipe if the images are really large you might see delays!

You can dynamically download images directly into the **ImageViewer** with a custom list model like this:



```

Form hi = new Form("ImageViewer", new BorderLayout());
final EncodedImage placeholder = EncodedImage.createFromImage(
    FontImage.createMaterial(FontImage.MATERIAL_SYNC, s).
        scaled(300, 300), false);

class ImageList implements ListModel<Image> {
    private int selection;
    private String[] imageURLs = {
        "http://awoiaf.westeros.org/images/thumb/9/93/AGameOfThrones.jpg/300px-AGameOfThrones.jpg",
        "http://awoiaf.westeros.org/images/thumb/3/39/AClashOfKings.jpg/300px-AClashOfKings.jpg",
        "http://awoiaf.westeros.org/images/thumb/2/24/ASTormOfSwords.jpg/300px-ASTormOfSwords.jpg",
        "http://awoiaf.westeros.org/images/thumb/a/a3/AFeastForCrows.jpg/300px-AFeastForCrows.jpg",
        "http://awoiaf.westeros.org/images/7/79/ADanceWithDragons.jpg"
    };
    private Image[] images;
    private EventDispatcher listeners = new EventDispatcher();

    public ImageList() {
        this.images = new EncodedImage[imageURLs.length];
    }

    public Image getItemAt(final int index) {
        if(images[index] == null) {
            images[index] = placeholder;
            Util.downloadUrlToStorageInBackground(imageURLs[index], "list" + index, (e) -> {
                try {
                    images[index] = EncodedImage.create(Storage.getInstance().createInputStream("list" + index));
                    listeners.fireDataChangeEvent(index, DataChangedListener.CHANGED);
                } catch(IOException err) {
                    err.printStackTrace();
                }
            });
        }
        return images[index];
    }

    public int getSize() {
        return imageURLs.length;
    }

    public int getSelectedIndex() {
        return selection;
    }

    public void setSelectedIndex(int index) {
        selection = index;
    }

    public void addDataChangedListener(DataChangedListener l) {
        listeners.addListener(l);
    }

    public void removeDataChangedListener(DataChangedListener l) {
        listeners.removeListener(l);
    }

    public void addSelectionListener(SelectionListener l) {
    }

    public void removeSelectionListener(SelectionListener l) {
    }

    public void addItem(Image item) {
    }
}

```

```

    public void removeItem(int index) {
    }
};

ImageList imodel = new ImageList();

ImageViewer iv = new ImageViewer(imodel.getItemAt(0));
iv.setImageList(imodel);
hi.add(BorderLayout.CENTER, iv);

```

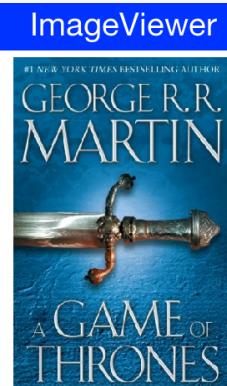


Figure 249. Dynamically fetching an image URL from the internet ^[5]

This fetches the images in the URL asynchronously and fires a data change event when the data arrives to automatically refresh the **ImageViewer** when that happens.

6.25. ScaleImageLabel & ScaleImageButton

ScaleImageLabel [<https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageLabel.html>] & **ScaleImageButton** [<https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageButton.html>] allow us to position an image that will grow/shrink to fit available space. In that sense they differ from **Label** & **Button** which keeps the image at the same size.



The default UIID of **ScaleImageLabel** is “Label”, however the default UIID of **ScaleImageButton** is “ScaleImageButton”. The reasoning for the difference is that the “Button” UIID includes a border and a lot of legacy.

You can use **ScaleImageLabel/ScaleImageButton** interchangeably. The only major difference between these components is the buttons ability to handle click events/focus.

Here is a simple example that also shows the difference between the **scale to fill** and **scale to fit** modes:

```

TableLayout tl = new TableLayout(2, 2);
Form hi = new Form("ScaleImageButton/Label", tl);
Style s = UIManager.getInstance().getComponentStyle("Button");
Image icon = FontImage.createMaterial(FontImage.MATERIAL_WARNING, s);
ScaleImageLabel fillLabel = new ScaleImageLabel(icon);
fillLabel.setBackgroundType(Style.BACKGROUND_IMAGE_SCALED_FILL);
ScaleImageButton fillButton = new ScaleImageButton(icon);
fillButton.setBackgroundType(Style.BACKGROUND_IMAGE_SCALED_FILL);
hi.add(tl.createConstraint().widthPercentage(20), new ScaleImageButton(icon));
    add(tl.createConstraint().widthPercentage(80), new ScaleImageLabel(icon));
    add(fillLabel);
    add(fillButton);
hi.show();

```



Figure 250. *ScaleImageLabel/Button*, the top row includes scale to fit versions (the default) whereas the bottom row includes the scale to fill versions



When styling these components keep in mind that changing attributes such as background behavior might cause an issue with their functionality or might not work.

6.26. Toolbar

The [Toolbar](https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html] API provides deep customization of the title bar area with more flexibility e.g. placing a [TextField](https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html) [https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html] for search or buttons in arbitrary title area positions. The [Toolbar](#) API replicates some of the native functionality available on Android/iOS and integrates with features such as the side menu to provide very fine grained control over the title area behavior.

The [Toolbar](#) needs to be installed into the [Form](#) in order for it to work. You can setup the Toolbar in one of these three ways:

1. `form.setToolbar(new Toolbar());` - allows you to activate the [Toolbar](#) to a specific [Form](#) and not for the entire application
2. `Toolbar.setGlobalToolbar(true);` - enables the [Toolbar](#) for all the forms in the app
3. Theme constant `globalToobarBool` - this is equivalent to `Toolbar.setGlobalToolbar(true);`

The basic functionality of the [Toolbar](#) includes the ability to add a command to the following 4 places:

- Left side of the title - `addCommandToLeftBar`
- Right side of the title - `addCommandToRightBar`

- Side menu bar (the drawer that opens when you click the icon on the top left or swipe the screen from left to right) - [addCommandToSideMenu](#)
- Overflow menu (the menu that opens when you tap the 3 vertical dots in the top right corner) - [addCommandToOverflowMenu](#)

The code below provides a brief overview of these options:

```
Toolbar.setGlobalToolbar(true);

Form hi = new Form("Toolbar", new BoxLayout(BoxLayout.Y_AXIS));
hi.getToolbar().addCommandToLeftBar("Left", icon, (e) -> Log.p("Clicked"));
hi.getToolbar().addCommandToRightBar("Right", icon, (e) -> Log.p("Clicked"));
hi.getToolbar().addCommandToOverflowMenu("Overflow", icon, (e) -> Log.p("Clicked"));
hi.getToolbar().addCommandToSideMenu("Sidemenu", icon, (e) -> Log.p("Clicked"));
hi.show();
```



Figure 251. The Toolbar

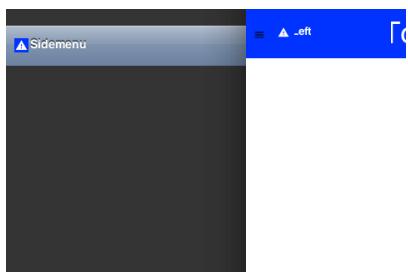


Figure 252. The default sidemenu of the Toolbar



Figure 253. The overflow menu of the Toolbar

Normally you can just set a title with a [String](#) but if you would want the component to be a text field or a multi line label you can use [setTitleComponent\(Component\)](#) which allows you to install any component into the title area.



The code below demonstrates searching using custom code however the Toolbar also has builtin support for search covered in the next section

The customization of the title area allows for some pretty powerful UI effects e.g. the code below allows searching dynamically within a set of entries and uses some very neat tricks:

```

Toolbar.setGlobalToolbar(true);
Style s = UIManager.getInstance().getComponentStyle("Title");

Form hi = new Form("Toolbar", new BoxLayout(BoxLayout.Y_AXIS));
TextField searchField = new TextField("", "Toolbar Search"); ①
searchField.getHintLabel().setUIID("Title");
searchField.setUIID("Title");
searchField.getAllStyles().setAlignment(Component.LEFT);
hi.getToolbar().setTitleComponent(searchField);
FontImage searchIcon = FontImage.createMaterial(FontImage.MATERIAL_SEARCH, s);
searchField.addDataChangeListener((i1, i2) -> { ②
    String t = searchField.getText();
    if(t.length() < 1) {
        for(Component cmp : hi.getContentPane()) {
            cmp.setHidden(false);
            cmp.setVisible(true);
        }
    } else {
        t = t.toLowerCase();
        for(Component cmp : hi.getContentPane()) {
            String val = null;
            if(cmp instanceof Label) {
                val = ((Label)cmp).getText();
            } else {
                if(cmp instanceof TextArea) {
                    val = ((TextArea)cmp).getText();
                } else {
                    val = (String)cmp.getPropertyValue("text");
                }
            }
            boolean show = val != null && val.toLowerCase().indexOf(t) > -1;
            cmp.setHidden(!show); ③
            cmp.setVisible(show);
        }
    }
    hi.getContentPane().animateLayout(250);
});
hi.getToolbar().addCommandToRightBar("", searchIcon, (e) -> {
    searchField.startEditingAsync(); ④
});

hi.add("A Game of Thrones").
    add("A Clash Of Kings").
    add("A Storm Of Swords").
    add("A Feast For Crows").
    add("A Dance With Dragons").
    add("The Winds of Winter").
    add("A Dream of Spring");
hi.show();

```

① We use a **TextField** the whole time and just style it to make it (and its hint) look like a regular title. An alternative way is to replace the title component dynamically.

② We use the **DataChangeListener** to update the search results as we type them.

③ Hidden & Visible use the opposite flag values to say similar things (e.g. when hidden is set to false you would want to set visible to true).

Visible indicates whether a component can be seen. It will still occupy the physical space on the screen even when it isn't visible. Hidden will remove the space occupied by the component from the screen, but some code might still try to paint it. Normally, visible is redundant but we use it with hidden for good measure.

④ The search button is totally unnecessary here. We can just click the **TextField**!

However, that isn't intuitive to most users so we added the button to start editing.

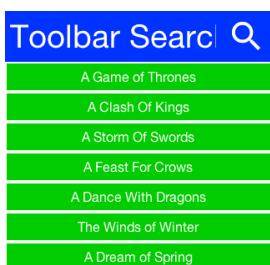


Figure 254. Search field within the toolbar



Figure 255. Search field after typing a couple of letters

6.26.1. Search Mode

While you can implement search manually using the builtin search offers a simpler and more uniform UI.

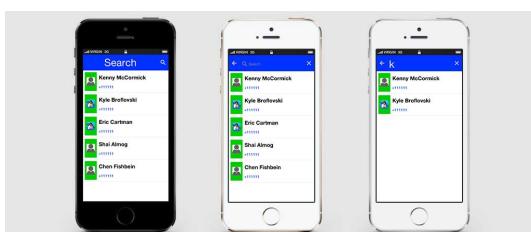


Figure 256. Builtin toolbar search functionality

You can customize the appearance of the search bar by using the UIID's: [ToolbarSearch](#), [TextFieldSearch](#) & [TextHintSearch](#).

In the sample below we fetch all the contacts from the device and enable search thru them, notice it expects and image called `duke.png` which is really just the default Codename One icon renamed and placed in the src folder:

```
Image duke = null;
try {
    duke = Image.createImage("/duke.png");
} catch(IOException err) {
    Log.e(err);
}
int fiveMM = Display.getInstance().convertToPixels(5);
final Image finalDuke = duke.scaledWidth(fiveMM);
Toolbar.setGlobalToolbar(true);
Form hi = new Form("Search", BoxLayout.y());
hi.add(new InfiniteProgress());
Display.getInstance().scheduleBackgroundTask(()-> {
    // this will take a while...
    Contact[] cnts = Display.getInstance().getAllContacts(true, true, true, true, false, false);
    Display.getInstance().callSerially(() -> {
        hi.removeAll();
        for(Contact c : cnts) {
            MultiButton m = new MultiButton();
            m.setTextLine1(c.getDisplayName());
            m.setTextLine2(c.getPrimaryPhoneNumber());
            Image pic = c.getPhoto();
            if(pic != null) {
                m.setIcon(fill(pic, finalDuke.getWidth(), finalDuke.getHeight()));
            } else {
                m.setIcon(finalDuke);
            }
            hi.add(m);
        }
        hi.revalidate();
    });
});
hi.getToolbar().addSearchCommand(e -> {
    String text = (String)e.getSource();
    if(text == null || text.length() == 0) {
        // clear search
        for(Component cmp : hi.getContentPane()) {
            cmp.setHidden(false);
            cmp.setVisible(true);
        }
        hi.getContentPane().animateLayout(150);
    } else {
        text = text.toLowerCase();
        for(Component cmp : hi.getContentPane()) {
            MultiButton mb = (MultiButton)cmp;
            String line1 = mb.getTextLine1();
            String line2 = mb.getTextLine2();
```

```

        boolean show = line1 != null && line1.toLowerCase().indexOf(text) > -1 ||  

            line2 != null && line2.toLowerCase().indexOf(text) > -1;  

        mb.setVisible(!show);  

        mb.setHidden(show);  

    }  

    hi.getContentPane().animateLayout(150);  

}  

}, 4);  
  

hi.show();

```

6.26.2. South Component

A common feature in side menu bar is the ability to add a component to the "south" part of the side menu.

Notice that this feature only works with the on-top and permanent versions of the side menu and not with the legacy versions:

```
toolbar.setComponentToSideMenuSouth(myComponent);
```

This places the component below the side menu bar. Notice that this component controls its entire UIID & is separate from the [SideNavigationViewPanel](#) UIID so if you set that component you might want to place it within a container that has the [SideNavigationViewPanel](#) UIID so it will blend with the rest of the UI.

6.26.3. Title Animations

Modern UI's often animate the title upon scrolling to balance the highly functional smaller title advantage with the gorgeous large image based title. This is pretty easy to do with the Toolbar API thru the Title animation API.

The code below shows off an attractive title based on a book by GRRM on top of text [\[6\]](#) that is scrollable. As the text is scrolled the title fades out.

```

Toolbar.setGlobalToolbar(true);

Form hi = new Form("Toolbar", new BoxLayout(BoxLayout.Y_AXIS));
EncodedImage placeholder = EncodedImage.createFromImage(Image.createImage(hi.getWidth(), hi.getWidth() / 5, 0xffff0000),
true);
URLImage background = URLImage.createToStorage(placeholder, "400px-AGameOfThrones.jpg",
"http://awoiaf.westeros.org/images/thumb/9/93/AGameOfThrones.jpg/400px-AGameOfThrones.jpg");
background.fetch();
Style stitle = hi.getToolbar().getTitleComponent().getUnselectedStyle();
stitle.setBgImage(background);
stitle.setBackgroundType(Style.BACKGROUND_IMAGE_SCALED_FILL);
stitle.setPaddingUnit(Style.UNIT_TYPE_DIPS, Style.UNIT_TYPE_DIPS, Style.UNIT_TYPE_DIPS, Style.UNIT_TYPE_DIPS);
stitle.setPaddingTop(15);
SpanButton credit = new SpanButton("This excerpt is from A Wiki Of Ice And Fire. Please check it out by clicking here!");
credit.addActionListener((e) -> Display.getInstance().execute("http://awoiaf.westeros.org/index.php/A_Game_of_Thrones"));
hi.add(new SpanLabel("A Game of Thrones is the first of seven planned novels in A Song of Ice and Fire, an epic fantasy
series by American author George R. R. Martin. It was first published on 6 August 1996. The novel was nominated for the
1998 Nebula Award and the 1997 World Fantasy Award,[1] and won the 1997 Locus Award.[2] The novella Blood of the Dragon,
comprising the Daenerys Targaryen chapters from the novel, won the 1997 Hugo Award for Best Novella. "));
add(new Label("Plot introduction", "Heading"));
add(new SpanLabel("A Game of Thrones is set in the Seven Kingdoms of Westeros, a land reminiscent of Medieval
Europe. In Westeros the seasons last for years, sometimes decades, at a time.\n\n" +
"\"Fifteen years prior to the novel, the Seven Kingdoms were torn apart by a civil war, known alternately as
\"Robert's Rebellion\" and the \"War of the Usurper.\" Prince Rhaegar Targaryen kidnapped Lyanna Stark, arousing the ire
of her family and of her betrothed, Lord Robert Baratheon (the war's titular rebel). The Mad King, Aerys II Targaryen,
had Lyanna's father and eldest brother executed when they demanded her safe return. Her second brother, Eddard, joined
his boyhood friend Robert Baratheon and Jon Arryn, with whom they had been fostered as children, in declaring war against
the ruling Targaryen dynasty, securing the allegiances of House Tully and House Arryn through a network of dynastic
marriages (Lord Eddard to Catelyn Tully and Lord Arryn to Lysa Tully). The powerful House Tyrell continued to support the
King, but House Lannister and House Martell both stalled due to insults against their houses by the Targaryens. The civil
war climaxed with the Battle of the Trident, when Prince Rhaegar was killed in battle by Robert Baratheon. The Lannisters
finally agreed to support King Aerys, but then brutally... "));
add(credit);
ComponentAnimation title = hi.getToolbar().getTitleComponent().createStyleAnimation("Title", 200);
hi.getAnimationManager().onTitleScrollAnimation(title);
hi.show();

```



Figure 257. The Toolbar starts with the large URLImage fetched from the web



Figure 258. As we scroll down the image fades and the title shrinks in size returning to the default UIID look



Figure 259. As scrolling continues the title reaches standard size

Almost all of the code above just creates the "look" of the application. The key piece of code above is this:

```
ComponentAnimation title = hi.getToolbar().getTitleComponent().createStyleAnimation("Title", 200);  
hi.getAnimationManager().onTitleScrollAnimation(title);
```

In the first line we create a style animation that will translate the style from the current settings to the destination UIID (the first argument) within 200 pixels of scrolling. We then bind this animation to the title scrolling animation event.

6.27. BrowserComponent & WebBrowser



BrowserComponent is a **peer component**, understanding this is crucial if your application depends on such a component. You can learn about peer components and their issues [here](https://www.codenameone.com/manual/advanced-topics.html#native-peer-components) [<https://www.codenameone.com/manual/advanced-topics.html#native-peer-components>].

The **WebBrowser** [<https://www.codenameone.com/javadoc/com/codename1/components/WebBrowser.html>] component shows the native device web browser when supported by the device and the **HTMLComponent** [<https://www.codenameone.com/javadoc/com/codename1/ui/html/HTMLComponent.html>] when the web browser isn't supported on the given device. If you only intend to target smartphones you should use the **BrowserComponent** [<https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html>] directly instead of the **WebBrowser**.

The **BrowserComponent** can point at an arbitrary URL to load it:

```
Form hi = new Form("Browser", new BorderLayout());
BrowserComponent browser = new BrowserComponent();
browser.setURL("https://www.codenameone.com/");
hi.add(BorderLayout.CENTER, browser);
```



Figure 260. Browser Component showing the Codename One website on the simulator



The scrollbars only appear in the simulator, device versions of the browser component act differently and support touch scrolling.



A **BrowserComponent** should be in the center of a BorderLayout. Otherwise its preferred size might be zero before the HTML finishes loading/layout in the native layer and layout might be incorrect as a result.

You can use **WebBrowser** and **BrowserComponent** interchangeably for most basic usage. However, if you need access to JavaScript or native browser functionality then there is really no use in going thru the **WebBrowser** abstraction.

The **BrowserComponent** has full support for executing local web pages from within the jar. The basic support uses the `jar:///` URL as such:

```
BrowserComponent wb = new BrowserComponent();
wb.setURL("jar:///Page.html");
```



On Android a native indicator might show up when the web page is loading. This can be disabled using the `Display.getInstance().setProperty("WebLoadingHidden", "true");` call. You only need to invoke this once.

6.27.1. BrowserComponent Hierarchy

When Codename One packages applications into native apps it hides a lot of details to make the process simpler. One of the things hidden is the fact that we aren't dealing with a JAR anymore, so `getResource/getResourceAsStream` are problematic... Both of these API's support hierarchies and a concept of package relativity both of which might not be supported on all OS's.

Codename One has its own `getResourceAsStream` in the [Display](#) [<https://www.codenameone.com/javadoc/com/codenname1/ui/Display.html>] class and that works just fine, but it requires that all files be in the `src` root directory.



That's why we recommend that you place files inside `res` files. A resource file allows you to add arbitrary data files and you can have as many resource files as you need.

For web developers this isn't enough since hierarchies are used often to represent the various dependencies, this means that many links & references are relative. To work with such hierarchies just place all of your resources in a hierarchy under the `html` package in the project source directory (`src/html`). The build server will `tar` the entire content of that package and add an `html.tar` file into the native package. This `tar` is seamlessly extracted on the device when you actually need the resources and only with new application versions (not on every launch). So assuming the resources are under the `html` root package they can be displayed with code like this:

```
try {
    browserComponent.setURLHierarchy("/htmlFile.html");
} catch(IOException err) {
    ...
}
```

Notice that the path is relative to the `html` directory and starts with `/` but inside the HTML files you should use relative (not absolute) paths.

Also notice that an `IOException` can be thrown due to the process of untarring. Its unlikely to happen but is entirely possible.

6.27.2. NavigationCallback

At the core of the `BrowserComponent` we have the [BrowserNavigationCallback](#) [<https://www.codenameone.com/javadoc/com/codenname1/ui/events/BrowserNavigationCallback.html>]. It might not seem like the most important interface within the browser but it is the "glue" that allows the JavaScript code to communicate back into the Java layer.

You can bind a `BrowserNavigationCallback` by invoking `setBrowserNavigationCallback` on the `BrowserComponent`. At that point with every navigation within the browser the callback will get invoked.

The `shouldNavigate` method from the `BrowserNavigationCallback` is invoked in a native thread and **NOT ON THE EDT!**

It is crucial that this method returns immediately and that it won't do any changes on the UI.

The `shouldNavigate` indicates to the native code whether navigation should proceed or not. E.g. if a user clicks a specific link we might choose to do something in the Java code so we can just return false and block the navigation. We can invoke `callSerially` [<https://www.codenameone.com/javadoc/com/codename1/ui/Display.html#callSerially-java.lang.Runnable->] to do the actual task in the Java side.

```
Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
    "   <head>\n" +
    "     <meta charset=\"utf-8\">\n" +
    "     <script>\n" +
    "       function fnc(message) {\n" +
    "         document.write(message);\n" +
    "       }\n" +
    "     </script>\n" +
    "   </head>\n" +
    "   <body >\n" +
    "     <p><a href=\"http://click\">Demo</a></p>\n" +
    "   </body>\n" +
    "</html>", null);
hi.add(BorderLayout.CENTER, bc);
bc.setBrowserNavigationCallback((url) -> {
    if(url.startsWith("http://click")) {
        Display.getInstance().callSerially(() -> bc.execute("fnc('<p>You clicked!</p>')"));
        return false;
    }
    return true;
});
```

BrowserCompone

[Demo](#)

Figure 261. Before the link is clicked for the "shouldNavigate" call

BrowserCompone

You clicked!

Figure 262. After the link is clicked for the "shouldNavigate" call



The JavaScript Bridge is implemented on top of the `BrowserNavigationCallback`.

6.27.3. JavaScript



The JavaScript bridge is sometimes confused with the JavaScript Port. The JavaScript bridge allows us to communicate with JavaScript from Java (and visa versa). The JavaScript port allows you to compile the Codename One application into a JavaScript application that runs in a standard web browser without code changes (think GWT without source changes and with thread support).+ We discuss the JavaScript port further later in the guide.

Codename One 4.0 introduced a new API for interacting with Javascript in Codename One. This API is part of the [BrowserComponent](#) class, and effectively replaces the [com.codename1.javascript package](#) [<https://www.codenameone.com/javadoc/com/codename1/javascript/package-summary.html>], which is now deprecated.

So what was wrong with the old API?

The old API provided a synchronous wrapper around an inherently asynchronous process, and made extensive use of [invokeAndBlock\(\)](#) underneath the covers. This resulted in a very nice API with high-level abstractions that played nicely with a synchronous programming model, but it came with a price-tag in terms of performance, complexity, and predictability. Let's take a simple example, getting a reference to the "window" object:

```
JSObject window = ctx.get("window");
```

This code looks harmless enough, but this is actually quite expensive. It issues a command to the [BrowserComponent](#), and uses [invokeAndBlock\(\)](#) to wait for the command to go through and send back a response. [invokeAndBlock\(\)](#) is a magical tool that allows you to "block" without blocking the EDT, but it has its costs, and shouldn't be overused. Most of the Codename One APIs that use [invokeAndBlock\(\)](#) indicate this in their name. E.g. [Component.animateLayoutAndWait\(\)](#). This gives you the expectation that this call could take some time, and helps to alert you to the underlying cost.

The problem with the `ctx.get("window")` call is that it looks the same as a call to [Map.get\(key\)](#). There's no indication that this call is expensive and could take time. One call like this probably isn't a big deal, but it doesn't take long before you have dozens or even hundreds of calls like this littered throughout your codebase, and they can be hard to pick out.

The New API

The new API fully embraces the asynchronous nature of Javascript. It uses callbacks instead of return values, and provides convenience wrappers with the appropriate "AndWait()" naming convention to allow for synchronous usage. Let's look at a simple example:



In all of the sample code below, you can assume that variables named `bc` represent an instance of [BrowserComponent](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html>].

```
bc.execute(  
    "callback.onSuccess(3+4)",  
    res -> Log.p("The result was "+res.getInt())  
)
```

This code should output “The result was 7” to the console. It is fully asynchronous, so you can include this code anywhere without worrying about it “bogging down” your code. The full signature of this form of the `execute()` [[https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html#execute-java.lang.String-com.codename1.util.SuccessCallback-\]](https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html#execute-java.lang.String-com.codename1.util.SuccessCallback-)] method is:

```
public void execute(String js, SuccessCallback<JSRef> callback)
```

The first parameter is just a javascript expression. This javascript **MUST** call either `callback.onSuccess(result)` or `callback.onError(message, errCode)` at some point in order for your callback to be called.

The second parameter is your callback that is executed from the javascript side, when `callback.onSuccess(res)` is called. The callback takes a single parameter of type `JSRef` [<https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.JSRef.html>] which is a generic wrapper around a javascript variable. JSRef has accessors to retrieve the value as some of the primitive types. E.g. `getBoolean()`, `getDouble()`, `getInt()`, `toString()`, and it provides some introspection via the `getType()` method.



It is worth noting that the callback method can only take a single parameter. If you need to pass multiple parameters, you may consider including them in a single string which you parse in your callback.

Synchronous Wrappers

As mentioned before, the new API also provides an `executeAndWait()` wrapper for `execute()` that will work synchronously. It, as its name suggests, uses `invokeAndBlock` under the hood so as not to block the EDT while it is waiting.

E.g.

```
JSRef res = bc.executeAndWait("callback.onSuccess(3+4)");  
Log.p("The result was "+res.Int());
```

Prints “The result was 7”.



When using the `andWait()` variant, it is **extremely** important that your Javascript calls your callback method at some point - otherwise it will block **indefinitely**. We provide variants of `executeAndWait()` that include a timeout in case you want to hedge against this possibility.

Multi-use Callbacks

The callbacks you pass to `execute()` and `executeAndWait()` are single-use callbacks. You can't, for example, store the `callback` variable on the javascript side for later use (e.g. to respond to a button click event). If you need a “multi-use” callback, you should use the `addJSCallback()` method instead. Its usage looks identical to `execute()`, the only difference is that the callback will live on after its first use. E.g. Consider the following code:

```
bc.execute(  
    "$('#somebutton').click(function(){callback.onSuccess('Button was clicked')})",  
    res -> Log.p(res.toString())  
)
```

The above example, assumes that jQuery is loaded in the webpage that we are interacting with, and we are adding a click handler to a button with ID “somebutton”. The click handler calls our callback.

If you run this example, the first time the button is clicked, you'll see “Button was clicked” printed to the console as expected. However, the 2nd time, you'll just get an exception. This is because the callback passed to `execute()` is only single-use.

We need to modify this code to use the `addJSCallback()` method as follows:

```
bc.addJSCallback(  
    "$('#somebutton').click(function(){callback.onSuccess('Button was clicked')})",  
    res -> Log.p(res.toString())  
)
```

Now it will work no matter how many times the button is clicked.

Passing Parameters to Javascript

In many cases, the javascript expressions that you execute will include parameters from your java code. Properly escaping these parameters is tricky at worst, and annoying at best. E.g. If you're passing a string, you need to make sure that it escapes quotes and new lines properly or it will cause the javascript to have a syntax error. Luckily we provide variants of `execute()` and `addJSCallback()` [https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html#addJSCallback-java.lang.String-com.codename1.util.SuccessCallback-] that allow you to pass your parameters and have them automatically escaped.

For example, suppose we want to pass a string with text to set in a textarea within the webpage. We can do something like:

```

bc.execute(
    "jQuery('#bio').text(${0}); jQuery('#age').text(${1})",
    new Object[]{ 
        "A multi-line\n string with \"quotes\"",
        27
    }
);

```

The gist is that you embed placeholders in the javascript expression that are replaced by the corresponding entry in an array of parameters. The `${0}` placeholder is replaced by the first item in the parameters array, the `${1}` placeholder is replaced by the 2nd, and so on.

Proxy Objects

The new API also includes a `JSProxy` [https://www.codenameone.com/javadoc/com/codename1/ui/ BrowserComponent.JSProxy.html] class that encapsulates a Javascript object simplify the getting and setting of properties on Javascript objects - and the calling of their methods. It provides essentially three core methods, along with several variants of each to allow for async or synchronous usages, parameters, and timeouts.

E.g. We might want to create a proxy for the `window.location` [https://developer.mozilla.org/en-US/docs/Web/API/Window/location] object so that we can access its properties more easily from Java.

```
JSProxy location = bc.createJSProxy("window.location");
```

Then we can retrieve its properties using the `get()` method:

```
location.get("href", res -> Log.p("location.href="+res));
```

Or synchronously:

```
JSRef href = location.getAndWait("href");
Log.p("location.href="+href);
```

We can also set its properties:

```
location.set("href", "http://www.google.com");
```

And call its methods:

```
location.call("replace", new Object[]{"http://www.google.com"}, 
    res -> Log.p("Return value was "+res)
);
```

Legacy JSObject Support

This section describes the now deprecated **JSObject** approach. It's here for reference by developers working with older code. We suggest using the new API when starting a new project.

BrowserComponent can communicate with the HTML code using JavaScript calls. E.g. we can create HTML like this:

```
Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
    "    <head>\n" +
    "        <meta charset=\"utf-8\">\n" +
    "        <script>\n" +
    "            function fnc(message) {\n" +
    "                document.write(message);\n" +
    "            }\n" +
    "        </script>\n" +
    "    </head>\n" +
    "    <body >\n" +
    "        <p>Demo</p>\n" +
    "    </body>\n" +
    "</html>", null);
TextField tf = new TextField();
hi.add(BorderLayout.CENTER, bc);
    add(BorderLayout.SOUTH, tf);
bc.addWebEventListener("onLoad", (e) -> bc.execute("fnc('<p>Hello World</p>')"));
tf.addActionListener((e) -> bc.execute("fnc('<p>" + tf.getText() + "</p>')"));
hi.show();
```

BrowserCompone

Hello World
Text From The Text Field

Text From The Text Field

Figure 263. JavaScript code was invoked to append text into the browser image above



Notice that opening an alert in an embedded native browser might not work

We use the **execute** method above to execute custom JavaScript code. We also have an **executeAndReturnString** method that allows us to receive a response value from the JavaScript side.

Coupled with **shouldNavigate** we can effectively do everything which is exactly what the JavaScript Bridge tries to do.

The JavaScript Bridge

While it's possible to just build everything on top of `execute` and `shouldNavigate`, both of these methods have their limits. That is why we introduced the `javascript` package, it allows you to communicate with JavaScript using intuitive code/syntax.

The [JavascriptContext](https://www.codenameone.com/javadoc/com/codename1/javascript/JavascriptContext.html) [https://www.codenameone.com/javadoc/com/codename1/javascript/JavascriptContext.html] class lays the foundation by enabling you to call JavaScript code directly from Java. It provides automatic type conversion between Java and JavaScript types as follows:

Table 5. Java to JavaScript

Java Type	Javascript Type
<code>String</code>	<code>String</code>
<code>Double/Integer/Float/Long</code>	<code>Number</code>
<code>Boolean</code>	<code>Boolean</code>
<code>JSONObject</code>	<code>Object</code>
<code>null</code>	<code>null</code>
<code>Other</code>	Not Allowed

Table 6. JavaScript to Java

Javascript Type	Java Type
<code>String</code>	<code>String</code>
<code>Number</code>	<code>Double</code>
<code>Boolean</code>	<code>Boolean</code>
<code>Object</code>	<code>JSONObject</code>
<code>Function</code>	<code>JSONObject</code>
<code>Array</code>	<code>JSONObject</code>
<code>null</code>	<code>null</code>
<code>undefined</code>	<code>null</code>

This conversion table is more verbose than necessary, since JavaScript functions and arrays are, in fact Objects themselves, so those rows are redundant. All JavaScript objects are converted to `JSONObject` [https://www.codenameone.com/javadoc/com/codename1/javascript/JSONObject.html].



We can access JavaScript variables easily from the context by using code like this:

```

Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
            "    <head>\n" +
            "        <meta charset=\"utf-8\">\n" +
            "    </head>\n" +
            "    <body >\n" +
            "        <p>This will appear twice...</p>\n" +
            "    </body>\n" +
            "</html>", null);
hi.add(BorderLayout.CENTER, bc);
bc.addWebEventListerner("onLoad", (e) -> {
    // Create a Javascript context for this BrowserComponent
    JavascriptContext ctx = new JavascriptContext(bc);

    String pageContent = (String)ctx.get("document.body.innerHTML");
    hi.add(BorderLayout.SOUTH, pageContent);
    hi.revalidate();
});
hi.show();

```

BrowserCompone

This will appear twice...

<p>This will appear twice...</p>

Figure 264. The contents was copied from the DOM and placed in the south position of the form

Notice that when you work with numeric values or anything related to the types mentioned above your code must be aware of the typing. E.g. in this case the type is **Double** and not **String**:

```
Double outerWidth = (Double)ctx.get("window.outerWidth");
```

You can also query the context for objects and modify their value e.g.

```

Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\"\>\n" +
            "    <head>\n" +
            "        <meta charset=\"utf-8\"\>\n" +
            "    </head>\n" +
            "    <body >\n" +
            "        <p>Please Wait...</p>\n" +
            "    </body>\n" +
            "</html>", null);
hi.add(BorderLayout.CENTER, bc);
bc.addWebEventListerner("onLoad", (e) -> {
    // Create a Javascript context for this BrowserComponent
    JavascriptContext ctx = new JavascriptContext(bc);

    JSObject jo = (JSObject)ctx.get("window");
    jo.set("location", "https://www.codenameone.com/");
});

```

This code effectively navigates to the Codename One home page by fetching the DOM's window object and setting its `location` property to <https://www.codenameone.com/> [<https://www.codenameone.com/>].

6.27.4. Cordova/PhoneGap Integration

PhoneGap was one of the first web app packager tools in the market. It's a tool that is effectively a browser component within a native wrapper coupled with native access API's. Cordova is the open source extension of this popular project.

Codename One supports embedding PhoneGap/Cordova applications directly into Codename One applications. This is relatively easy to do with the `BrowserComponent` and JavaScript integration. The main aspect that this integration requires is support for Cordova plugins & its JavaScript API's.

The effort to integrate Cordova/PhoneGap support into Codename One is handled within an open source github project [here](#) [<https://github.com/codenameone/CN1Cordova>]. The chief benefits of picking Codename One rather than using Cordova directly are:

- Build Cloud
- Better Native Code Support
- Better Protection Of IP
- IDE Integration Java - JavaScript - HTML
- Easy, Doesn't Require A Mac, Automates Certificates/Signing
- Migration To Java

This is discussed further in the [original announcement](#) [<https://www.codenameone.com/blog/phonegap-cordova-compatibility-for-codename-one.html>].

6.28. AutoCompleteTextField

The [AutoCompleteTextField](#) [https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html] allows us to write text into a text field and select a completion entry from the list in a similar way to a search engine.

This is really easy to incorporate into your code, just replace your usage of [TextField](#) [https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html] with [AutoCompleteTextField](#) and define the data that the autocomplete should work from. There is a default implementation that accepts a [String](#) array or a [ListModel](#) [https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html] for completion strings, this can work well for a "small" set of thousands (or tens of thousands) of entries.

E.g. This is a trivial use case that can work well for smaller sample sizes:

```
Form hi = new Form("Auto Complete", new BoxLayout(BoxLayout.Y_AXIS));
AutoCompleteTextField ac = new AutoCompleteTextField("Short", "Shock", "Sholder", "Shrek");
ac.setMinimumElementsShownInPopup(5);
hi.add(ac);
```



Figure 265. Autocomplete Text Field

However, if you wish to query a database or a web service you will need to derive the class and perform more advanced filtering by overriding the [filter](#) method:

```

public void showForm() {
    final DefaultListModel<String> options = new DefaultListModel<>();
    AutoCompleteTextField ac = new AutoCompleteTextField(options) {
        @Override
        protected boolean filter(String text) {
            if(text.length() == 0) {
                return false;
            }
            String[] l = searchLocations(text);
            if(l == null || l.length == 0) {
                return false;
            }

            options.removeAll();
            for(String s : l) {
                options.addItem(s);
            }
            return true;
        }
    };

    ac.setMinimumElementsShownInPopup(5);
    hi.add(ac);
    hi.add(new SpanLabel("This demo requires a valid google API key to be set below "
        + "you can get this key for the webservice (not the native key) by following the instructions here: "
        + "https://developers.google.com/places/web-service/get-api-key"));
    hi.add(apiKey);
    hi.getToolbar().addCommandToRightBar("Get Key", null, e ->
Display.getInstance().execute("https://developers.google.com/places/web-service/get-api-key"));
    hi.show();
}

TextField apiKey = new TextField();

String[] searchLocations(String text) {
    try {
        if(text.length() > 0) {
            ConnectionRequest r = new ConnectionRequest();
            r.setPost(false);
            r.setUrl("https://maps.googleapis.com/maps/api/place/autocomplete/json");
            r.addArgument("key", apiKey.getText());
            r.addArgument("input", text);
            NetworkManager.getInstance().addToQueueAndWait(r);
            Map<String, Object> result = new JSONParser().parseJSON(new InputStreamReader(new
ByteArrayInputStream(r.getResponseData()), "UTF-8"));
            String[] res = Result.fromContent(result).getAsStringArray("//description");
            return res;
        }
    } catch(Exception err) {
        Log.e(err);
    }
    return null;
}
}

```



Figure 266. Autocomplete Text Field with a webservice

6.28.1. Using Images In AutoCompleteTextField

One question I got a few times is "How do you customize the results of the auto complete field"?

This sounds difficult to most people as we can only work with Strings so how do we represent additional data or format the date correctly?

The answer is actually pretty simple, we still need to work with Strings because auto-complete is first and foremost a text field. However, that doesn't preclude our custom renderer from fetching data that might be placed in a different location and associated with the result.

The following source code presents an auto-complete text field with images in the completion popup and two lines for every entry:

```

final String[] characters = { "Tyrion Lannister", "Jaime Lannister", "Cersei Lannister", "Daenerys Targaryen",
    "Jon Snow", "Petyr Baelish", "Jorah Mormont", "Sansa Stark", "Arya Stark", "Theon Greyjoy"
    // snipped the rest for clarity
};

Form current = new Form("AutoComplete", BoxLayout.y());

AutoCompleteTextField ac = new AutoCompleteTextField(characters);

final int size = Display.getInstance().convertToPixels(7);
final EncodedImage placeholder = EncodedImage.createFromImage(Image.createImage(size, size, 0xffffffff), true);

final String[] actors = { "Peter Dinklage", "Nikolaj Coster-Waldau", "Lena Headey"}; ①
final Image[] pictures = {
    URLImage.createToStorage(placeholder, "tyrion","http://i.lv3.hbo.com/assets/images/series/game-of-
    thrones/character/s5/tyrion-lannister-512x512.jpg"),
    URLImage.createToStorage(placeholder, "jaime","http://i.lv3.hbo.com/assets/images/series/game-of-
    thrones/character/s5/jamie-lannister-512x512.jpg"),
    URLImage.createToStorage(placeholder, "cersei","http://i.lv3.hbo.com/assets/images/series/game-of-
    thrones/character/s5/cersei-lannister-512x512.jpg")
};

ac.setCompletionRenderer(new ListCellRenderer() {
    private final Label focus = new Label(); ②
    private final Label line1 = new Label(characters[0]);
    private final Label line2 = new Label(actors[0]);
    private final Label icon = new Label(pictures[0]);
    private final Container selection = BorderLayout.center(
        BoxLayout.encloseY(line1, line2)).add(BorderLayout.EAST, icon);

    @Override
    public Component getListCellRendererComponent(com.codename1.ui.List list, Object value, int index, boolean
    isSelected) {
        for(int iter = 0 ; iter < characters.length ; iter++) {
            if(characters[iter].equals(value)) {
                line1.setText(characters[iter]);
                if(actors.length > iter) {
                    line2.setText(actors[iter]);
                    icon.setIcon(pictures[iter]);
                } else {
                    line2.setText(""); ③
                    icon.setIcon(placeholder);
                }
                break;
            }
        }
        return selection;
    }

    @Override
    public Component getListFocusComponent(com.codename1.ui.List list) {
        return focus;
    }
});

current.add(ac);

current.show();

```

- ① We have duplicate arrays that are only partial for clarity. This is a separate list of data element but you can fetch the additional data from anywhere
- ② We create the renderer UI instantly in the fields with the helper methods for wrapping elements which is pretty cool & terse

③ In a renderer it's important to always set the value especially if you don't have a value in place

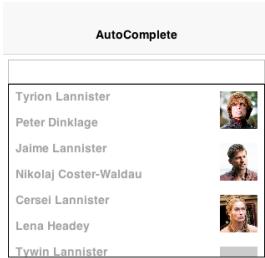


Figure 267. Auto complete with images

6.29. Picker

Picker [<https://www.codenameone.com/javadoc/com/codename1/ui/spinner/Picker.html>] occupies the limbo between native widget and lightweight widget. Picker is more like **TextField/TextArea** in the sense that it's a Codename One widget that calls the native code only during editing.

The reasoning for this is the highly native UX and functionality related to this widget type which should be quite obvious from the screenshots below.

At this time there are 4 types of pickers:

- Time
- Date & Time
- Date
- Strings

If a platform doesn't support native pickers an internal fallback implementation is used. This is the implementation we always use in the simulator so assume different behavior when building for the device.



While Android supports Date, Time native pickers it doesn't support the Date & Time native picker UX and will fallback in that case.

The sample below includes all picker types:

```

Form hi = new Form("Picker", new BoxLayout(BoxLayout.Y_AXIS));
Picker datePicker = new Picker();
datePicker.setType(Display.PICKER_TYPE_DATE);
Picker dateTimePicker = new Picker();
dateTimePicker.setType(Display.PICKER_TYPE_DATE_AND_TIME);
Picker timePicker = new Picker();
timePicker.setType(Display.PICKER_TYPE_TIME);
Picker stringPicker = new Picker();
stringPicker.setType(Display.PICKER_TYPE_STRINGS);
Picker durationPicker = new Picker();
durationPicker.setType(Display.PICKER_TYPE_DURATION);
Picker minuteDurationPicker = new Picker();
minuteDurationPicker.setType(Display.PICKER_TYPE_DURATION_MINUTES);
Picker hourDurationPicker = new Picker();
hourDurationPicker.setType(Display.PICKER_TYPE_DURATION_HOURS);

datePicker.setDate(new Date());
dateTimePicker.setDate(new Date());
timePicker.setTime(10 * 60); // 10:00AM = Minutes since midnight
stringPicker.setStrings("A Game of Thrones", "A Clash Of Kings", "A Storm Of Swords", "A Feast For Crows",
    "A Dance With Dragons", "The Winds of Winter", "A Dream of Spring");
stringPicker.setSelectedString("A Game of Thrones");

hi.add(datePicker).add(dateTimePicker).add(timePicker)
    .add(stringPicker).add(durationPicker)
    .add(minuteDurationPicker).add(hourDurationPicker);
hi.show();

```

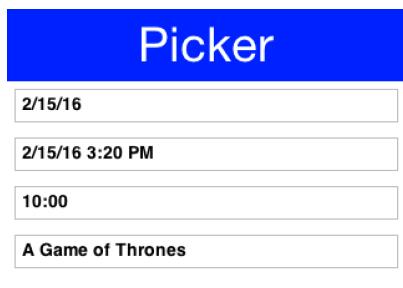


Figure 268. The various picker components

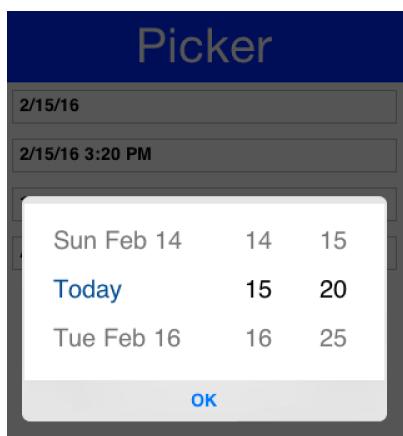


Figure 269. The date & time picker on the simulator

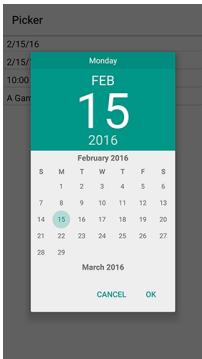


Figure 270. The date picker component on the Android device

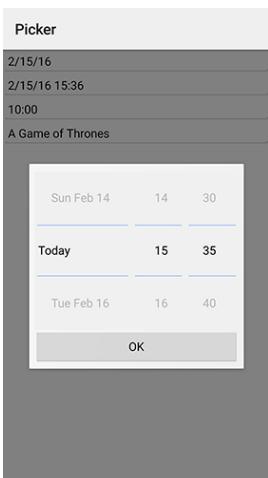


Figure 271. Date & time picker on Android. Notice it didn't use a builtin widget since there is none

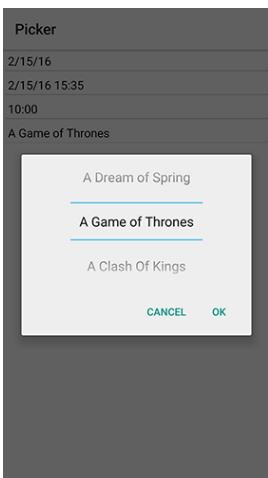


Figure 272. String picker on the native Android device



Figure 273. Time picker on the Android device



Figure 274. Duration picker on Android device



Figure 275. Hours Duration picker on Android device

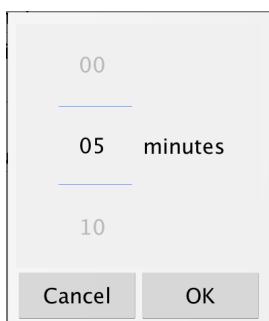


Figure 276. Minutes Duration picker on Android device

The text displayed by the picker on selection is generated automatically by the `updateValue()` method. You can override it to display a custom formatted value and call `setText(String)` with the correct display string.

A common use case is to format date values based on a specific appearance and `Picker` has builtin support for a custom display formatter. Just use the `setFormatter(SimpleDateFormat)` method and set the appearance for the field.

6.30. SwipeableContainer

The [SwipeableContainer](https://www.codenameone.com/javadoc/com/codename1/ui/SwipeableContainer.html) [https://www.codenameone.com/javadoc/com/codename1/ui/SwipeableContainer.html] allows us to place a component such as a [MultiButton](https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html) [https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html] on top of additional "options" that can be exposed by swiping the component to the side.

This swipe gesture is commonly used in touch interfaces to expose features such as delete, edit etc. It's trivial to use this component by just determining the components placed on top and bottom (the revealed component).

```
SwipeableContainer swip = new SwipeableContainer(bottom, top);
```

We can combine some of the demos above including the [Slider stars demo](#) to rank GRRM's books in an interactive way:

```
Form hi = new Form("Swipe", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(createRankWidget("A Game of Thrones", "1996"));
    add(createRankWidget("A Clash Of Kings", "1998"));
    add(createRankWidget("A Storm Of Swords", "2000"));
    add(createRankWidget("A Feast For Crows", "2005"));
    add(createRankWidget("A Dance With Dragons", "2011"));
    add(createRankWidget("The Winds of Winter", "TBD"));
    add(createRankWidget("A Dream of Spring", "TBD"));
hi.show();

public SwipeableContainer createRankWidget(String title, String year) {
    MultiButton button = new MultiButton(title);
    button.setTextLine2(year);
    return new SwipeableContainer(FlowLayout.encloseCenterMiddle(createStarRankSlider()),
        button);
}
```

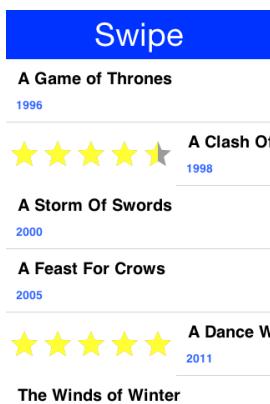


Figure 277. SwipeableContainer showing a common use case of ranking on swipe

6.31. EmbeddedContainer

[EmbeddedContainer](https://www.codenameone.com/javadoc/com/codename1/ui/util/EmbeddedContainer.html) [https://www.codenameone.com/javadoc/com/codename1/ui/util/EmbeddedContainer.html] solves a problem that exists only within the GUI builder and the class makes no sense outside of the context of the GUI builder. The necessity for [EmbeddedContainer](#) came about due to iPhone inspired designs that relied on tabs (iPhone style tabs at the bottom of the screen) where different features of the application are within a different tab.

This didn't mesh well with the GUI builder navigation logic and so we needed to rethink some of it. We wanted to reuse GUI as much as possible while still enjoying the advantage of navigation being completely managed for me.

Android does this with Activities and the iPhone itself has a view controller, both approaches are problematic for Codename One. The problem is that you have what is effectively two incompatible hierarchies to mix and match.

The Component/Container hierarchy is powerful enough to represent such a UI but we needed a "marker" to indicate to the [UIBuilder](https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html) [https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html] where a "root" component exists so navigation occurs only within the given "root". Here [EmbeddedContainer](#) comes into play, its a simple container that can only contain another GUI from the GUI builder. Nothing else. So we can place it in any form of UI and effectively have the UI change appropriately and navigation would default to "sensible values".

Navigation replaces the content of the embedded container; it finds the embedded container based on the component that broadcast the event. If you want to navigate manually just use the `showContainer()` method which accepts a component, you can give any component that is under the [EmbeddedContainer](#) you want to replace and Codename One will be smart enough to replace only that component.

The nice part about using the [EmbeddedContainer](#) is that the resulting UI can be very easily refactored to provide a more traditional form based UI without duplicating effort and can be easily adapted to a more tablet oriented UI (with a side bar) again without much effort.

6.32. MapComponent

 The MapComponent uses a somewhat outdated tiling API which is not as rich as modern native maps. We recommend using the [GoogleMap's Native cn1lib](https://github.com/codenameone/codenameone-google-maps/) [https://github.com/codenameone/codenameone-google-maps/] to integrate native mapping functionality into the Codename One app.

The [MapComponent](https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html) [https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html] uses the OpenStreetMap webservice by default to display a navigatable map.

The code was contributed by Roman Kamyk and was originally used for a LWUIT application.

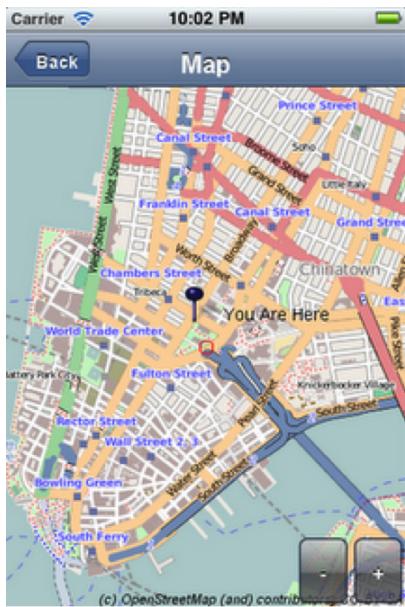


Figure 278. Map Component

The screenshot above was produced using the following code:

```
Form map = new Form("Map");
map.setLayout(new BorderLayout());
map.setScrollable(false);
final MapComponent mc = new MapComponent();

try {
    //get the current location from the Location API
    Location loc = LocationManager.getLocationManager().getCurrentLocation();

    Coord lastLocation = new Coord(loc.getLatitude(), loc.getLongitude());
    Image i = Image.createImage("/blue_pin.png");
    PointsLayer pl = new PointsLayer();
    pl.setPointIcon(i);
    PointLayer p = new PointLayer(lastLocation, "You Are Here", i);
    p.setDisplayName(true);
    pl.addPoint(p);
    mc.addLayer(pl);
} catch (IOException ex) {
    ex.printStackTrace();
}
mc.zoomToLayers();

map.addComponent(BorderLayout.CENTER, mc);
map.addCommand(new BackCommand());
map.setBackCommand(new BackCommand());
map.show();
```

The example below shows how to integrate the **MapComponent** [<https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html>] with the Google **Location** [<https://www.codenameone.com/javadoc/com/codename1/location/Location.html>] API. make sure to obtain your secret api key from the Google

[Location](https://www.codenameone.com/javadoc/com/codename1/location/Location.html) [https://www.codenameone.com/javadoc/com/codename1/location/Location.html] data API at: <https://developers.google.com/maps/documentation/places/>



Figure 279. MapComponent with Google Location API

```
final Form map = new Form("Map");
map.setLayout(new BorderLayout());
map.setScrollable(false);
final MapComponent mc = new MapComponent();
Location loc = LocationManager.getLocationManager().getCurrentLocation();
//use the code from above to show you on the map
putMeOnMap(mc);
map.addComponent(BorderLayout.CENTER, mc);
map.addCommand(new BackCommand());
map.setBackCommand(new BackCommand());

ConnectionRequest req = new ConnectionRequest() {

    protected void readResponse(InputStream input) throws IOException {
        JSONParser p = new JSONParser();
        Hashtable h = p.parse(new InputStreamReader(input));
        // "status" : "REQUEST_DENIED"
        String response = (String)h.get("status");
        if(response.equals("REQUEST_DENIED")){
            System.out.println("make sure to obtain a key from "
                + "https://developers.google.com/maps/documentation/places/");
            progress.dispose();
            Dialog.show("Info", "make sure to obtain an application key from "
                + "google places api's"
                , "Ok", null);
            return;
        }

        final Vector v = (Vector) h.get("results");

        Image im = Image.createImage("/red_pin.png");
        PointsLayer pl = new PointsLayer();
        pl.setPointIcon(im);
        pl.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent evt) {
                PointLayer p = (PointLayer) evt.getSource();
                System.out.println("pressed " + p);
            }
        });
    }
}
```

```

        Dialog.show("Details", "" + p.getName(), "Ok", null);
    }
});

for (int i = 0; i < v.size(); i++) {
    Hashtable entry = (Hashtable) v.elementAt(i);
    Hashtable geo = (Hashtable) entry.get("geometry");
    Hashtable loc = (Hashtable) geo.get("location");
    Double lat = (Double) loc.get("lat");
    Double lng = (Double) loc.get("lng");
    PointLayer point = new PointLayer(new Coord(lat.doubleValue(), lng.doubleValue()),
        (String) entry.get("name"), null);
    pl.addPoint(point);
}
progress.dispose();

mc.addLayer(pl);
map.show();
mc.zoomToLayers();

}
};

req.setUrl("https://maps.googleapis.com/maps/api/place/search/json");
req.setPost(false);
req.addArgument("location", "" + loc.getLatitude() + "," + loc.getLongitude());
req.addArgument("radius", "500");
req.addArgument("types", "food");
req.addArgument("sensor", "false");

//get your own key from https://developers.google.com/maps/documentation/places/
//and replace it here.
String key = "yourAPIKey";

req.addArgument("key", key);

NetworkManager.getInstance().addToQueue(req);
}
catch (IOException ex) {
    ex.printStackTrace();
}
}
}

```

6.33. Chart Component

The **charts** package enables Codename One developers to add charts and visualizations to their apps without having to include external libraries or embedding web views. We also wanted to harness the new features in the graphics pipeline to maximize performance.

6.33.1. Device Support

Since the charts package makes use of 2D transformations and shapes, it requires some of the graphics features that are not yet available on all platforms. Currently the following platforms are supported:

1. Simulator
2. Android
3. iOS

6.33.2. Features

1. **Built-in support for many common types of charts** including bar charts, line charts, stacked charts, scatter charts, pie charts and more.
2. **Pinch Zoom** - The [ChartComponent](https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html) [<https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html>] class includes optional pinch zoom support.
3. **Panning Support** - The [ChartComponent](https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html) [<https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html>] class includes optional support for panning.

6.33.3. Chart Types

The `com.codename1.charts` package includes models and renderers for many different types of charts. It is also extensible so that you can add your own chart types if required. The following screen shots demonstrate a small sampling of the types of charts that can be created.

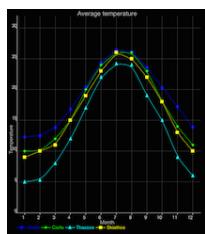


Figure 280. Line Charts

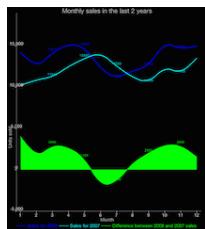


Figure 281. Cubic Line Charts

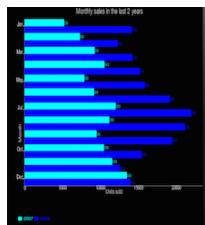


Figure 282. Bar Charts

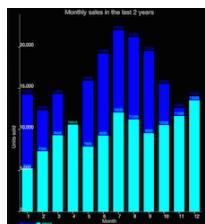


Figure 283. Stacked Bar Charts

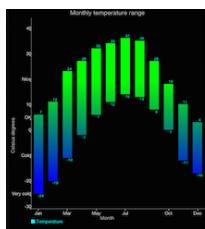


Figure 284. Range Bar Charts

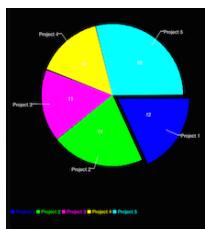


Figure 285. Pie Charts

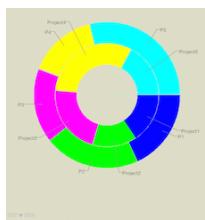


Figure 286. Doughnut Charts

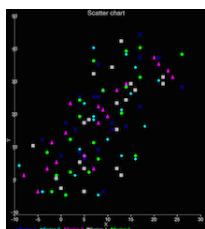


Figure 287. Scatter Charts

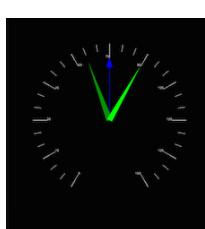


Figure 288. Dial Charts

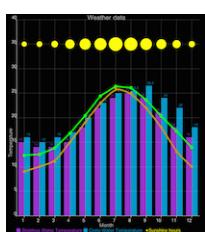


Figure 289. Combined Charts

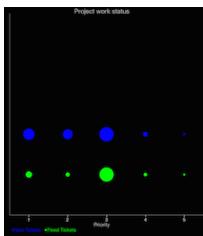


Figure 290. Bubble Charts

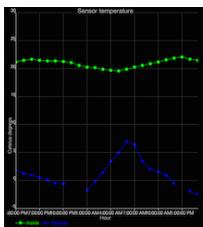


Figure 291. Time Charts



The above screenshots were taken from the [ChartsDemo app](https://github.com/codenameone/codenameone-demos/tree/master/ChartsDemo) [<https://github.com/codenameone/codenameone-demos/tree/master/ChartsDemo>]. You can start playing with this app by checking it out from our git repository.

6.33.4. How to Create A Chart

Adding a chart to your app involves four steps:

- Build the model.** You can construct a model (aka data set) for the chart using one of the existing model classes in the `com.codename1.charts.models` package. Essentially, this is just where you add the data that you want to display.
- Set up a renderer.** You can create a renderer for your chart using one of the existing renderer classes in the `com.codename1.charts.renderers` package. The renderer allows you to specify how the chart should look. E.g. the colors, fonts, styles, to use.
- Create the Chart View.** Use one of the existing `view` classes in the `com.codename1.charts.views` package.
- Create a `ChartComponent`** [<https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html>]. In order to add your chart to the UI, you need to wrap it in a `ChartComponent` [<https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html>] object.

You can check out the [ChartsDemo](https://github.com/codenameone/codenameone-demos/tree/master/ChartsDemo) [<https://github.com/codenameone/codenameone-demos/tree/master/ChartsDemo>] app for specific examples, but here is a high level view of some code that creates a Pie Chart.

```
/**
 * Creates a renderer for the specified colors.
 */
private DefaultRenderer buildCategoryRenderer(int[] colors) {
    DefaultRenderer renderer = new DefaultRenderer();
    renderer.setLabelsTextSize(15);
    renderer.setLegendTextSize(15);
    renderer.setMargins(new int[]{20, 30, 15, 0});
    for (int color : colors) {
```

```

        SimpleSeriesRenderer r = new SimpleSeriesRenderer();
        r.setColor(color);
        renderer.addSeriesRenderer(r);
    }
    return renderer;
}

/**
 * Builds a category series using the provided values.
 *
 * @param titles the series titles
 * @param values the values
 * @return the category series
 */
protected CategorySeries buildCategoryDataset(String title, double[] values) {
    CategorySeries series = new CategorySeries(title);
    int k = 0;
    for (double value : values) {
        series.add("Project " + ++k, value);
    }

    return series;
}

public Form createPieChartForm() {

    // Generate the values
    double[] values = new double[]{12, 14, 11, 10, 19};

    // Set up the renderer
    int[] colors = new int[]{ColorUtil.BLUE, ColorUtil.GREEN, ColorUtil.MAGENTA, ColorUtil.YELLOW, ColorUtil.CYAN};
    DefaultRenderer renderer = buildCategoryRenderer(colors);
    renderer.setZoomButtonsVisible(true);
    renderer.setZoomEnabled(true);
    renderer.setChartTitleTextSize(20);
    renderer.setDisplayValues(true);
    renderer.setShowLabels(true);
    SimpleSeriesRenderer r = renderer.getSeriesRendererAt(0);
    r.setGradientEnabled(true);
    r.setGradientStart(0, ColorUtil.BLUE);
    r.setGradientStop(0, ColorUtil.GREEN);
    r.setHighlighted(true);

    // Create the chart ... pass the values and renderer to the chart object.
    PieChart chart = new PieChart(buildCategoryDataset("Project budget", values), renderer);

    // Wrap the chart in a Component so we can add it to a form
    ChartComponent c = new ChartComponent(chart);

    // Create a form and show it.
    Form f = new Form("Budget");
    f.setLayout(new BorderLayout());
    f.addComponent(BorderLayout.CENTER, c);
    return f;
}

```

6.34. Calendar

The [Calendar](https://www.codenameone.com/javadoc/com/codename1/ui/Calendar.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Calendar.html] class allows us to display a traditional calendar picker and optionally highlight days in various ways.



We normally recommend developers use the [Picker UI](#) rather than use the calendar to pick a date. It looks better on the devices.

Simple usage of the [Calendar](#) class looks something like this:

```
Form hi = new Form("Calendar", new BorderLayout());
Calendar cld = new Calendar();
cld.addActionListener((e) -> Log.p("You picked: " + new Date(cld.getSelectedDay())));
hi.add(BorderLayout.CENTER, cld);
```

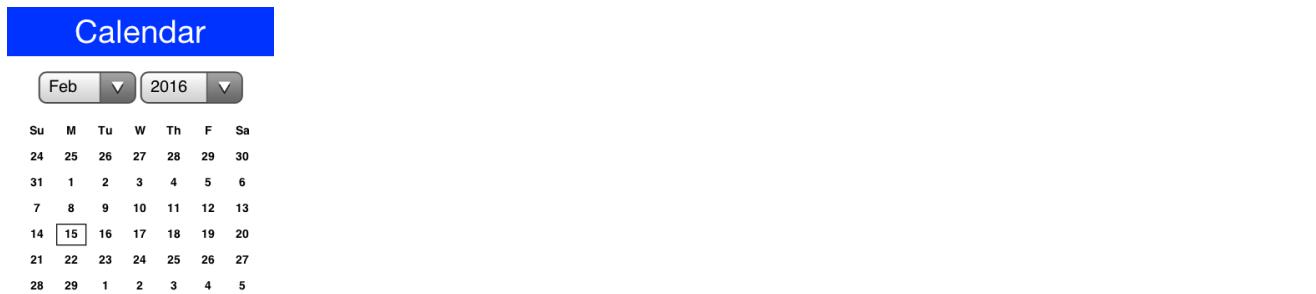


Figure 292. The *Calendar* component

6.35. [ToastBar](#)

The [ToastBar](#) [<https://www.codenameone.com/javadoc/com/codename1/components/ToastBar.html>] class allows us to display non-obtrusive status messages to the user at the bottom of the screen. This is useful for such things as informing the user of a long-running task (like downloading a file in the background), or popping up an error message that doesn't require a response from the user.

Simple usage of the [ToastBar](#) class looks something like this:

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Downloading your file...");
status.show();

// ... Later on when download completes
status.clear();
```

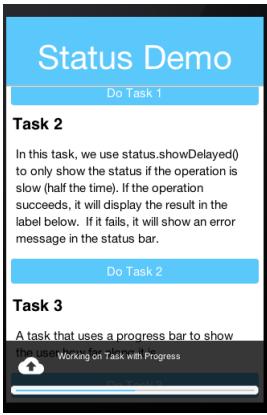


Figure 293. ToastBar with message

We can show a progress indicator in the ToastBar like this:

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Hello world");
status.setShowProgressIndicator(true);
status.show();
```



Figure 294. Status Message with Progress Bar

We can automatically clear a status message/progress after a timeout using the `setExpires` method as such:

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Hello world");
status.setExpires(3000); // only show the status for 3 seconds, then have it automatically clear
status.show();
```

We can also delay the showing of the status message using `showDelayed` as such:

```

Status status = ToastBar.getInstance().createStatus();
status.setMessage("Hello world");
status.showDelayed(300); // Wait 300 ms to show the status

// ... Some time later, clear the status... this may be before it shows at all
status.clear();

```

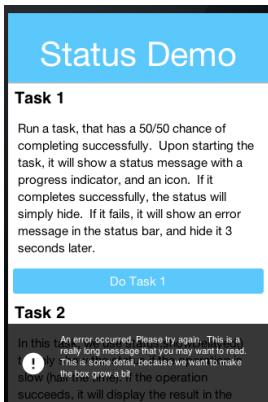


Figure 295. *ToastBar* with a multiline message

6.35.1. Actions In *ToastBar*

Probably the best usage example for actions in toast is in the gmail style undo. If you are not a gmail user then the gmail app essentially never prompts for confirmation!

It just does whatever you ask and pops a "toast message" with an option to undo. So if you clicked by mistake you have 3-4 seconds to take that back.

This simple example shows you how you can undo any addition to the UI in a similar way to gmail:

```

Form hi = new Form("Undo", BoxLayout.y());
Button add = new Button("Add");

add.addActionListener(e -> {
    Label l = new Label("Added this");
    hi.add(l);
    hi.revalidate();
    ToastBar.showMessage("Added, click here to undo...", FontImage.MATERIAL_UNDO,
        ee -> {
            l.remove();
            hi.revalidate();
        });
});
hi.add(add);
hi.show();

```

6.36. SignatureComponent

The [SignatureComponent](https://www.codenameone.com/javadoc/com/codename1/components/SignatureComponent.html) [https://www.codenameone.com/javadoc/com/codename1/components/SignatureComponent.html] provides a widget that allows users to draw their signature in the app.

Simple usage of the [SignatureComponent](#) class looks like:

```
Form hi = new Form("Signature Component");
hi.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
hi.add("Enter Your Name:");
hi.add(new TextField());
hi.add("Signature:");
SignatureComponent sig = new SignatureComponent();
sig.addActionListener(evt-> {
    System.out.println("The signature was changed");
    Image img = sig.getSignatureImage();
    // Now we can do whatever we want with the image of this signature.
});
hi.addComponent(sig);
hi.show();
```

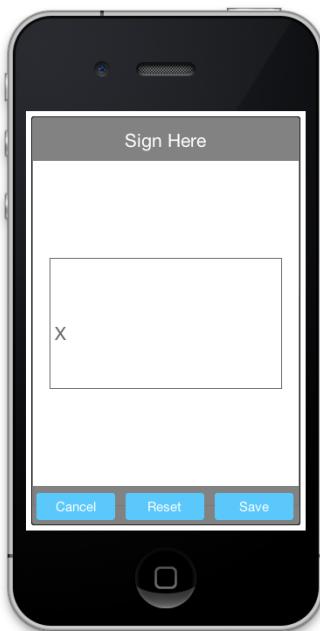


Figure 296. The signature Component

6.37. Accordion

The [Accordion](https://www.codenameone.com/javadoc/com/codename1/components/Accordion.html) [https://www.codenameone.com/javadoc/com/codename1/components/Accordion.html] displays collapsible content panels.

Simple usage of the [Accordion](#) class looks like:

```

Form f = new Form("Accordion", new BoxLayout(BoxLayout.Y_AXIS));
f.setScrollableY(true);
Accordion accr = new Accordion();
accr.addContent("Item1", new SpanLabel("The quick brown fox jumps over the lazy dog\n"
+ "The quick brown fox jumps over the lazy dog"));
accr.addContent("Item2", new SpanLabel("The quick brown fox jumps over the lazy dog\n"
+ "The quick brown fox jumps over the lazy dog\n"
+ "The quick brown fox jumps over the lazy dog\n"
+ "The quick brown fox jumps over the lazy dog\n"
+ ""));
accr.addContent("Item3", BoxLayout.encloseY(new Label("Label"), new TextField(), new Button("Button"), new CheckBox("CheckBox")));
f.add(accr);
f.show();

```

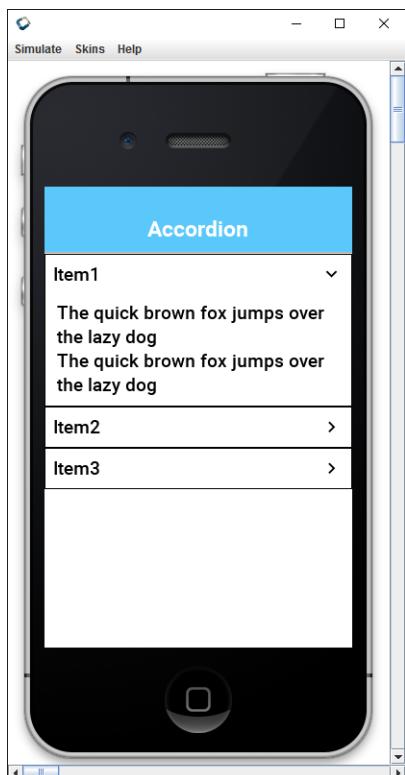


Figure 297. The Accordion Component

6.38. Floating Hint

[FloatingHint](https://www.codenameone.com/javadoc/com/codename1/components/FloatingHint.html) [https://www.codenameone.com/javadoc/com/codename1/components/FloatingHint.html] wraps a text component with a special container that can animate the hint label into a title label when the text component is edited or has content within it.

```

Form hi = new Form("Floating Hint", BoxLayout.y());
TextField first = new TextField("", "First Field");
TextField second = new TextField("", "Second Field");
hi.add(new FloatingHint(first));
    add(new FloatingHint(second));
    add(new Button("Go"));
hi.show();

```



Figure 298. The `FloatingHint` component with one component that contains text and another that doesn't

6.39. Floating Button

The material design floating action button is a powerful tool for promoting an action within your application.

FloatingActionButton

[\[https://www.codenameone.com/javadoc/com/codename1/components/FloatingActionButton.html\]](https://www.codenameone.com/javadoc/com/codename1/components/FloatingActionButton.html)

`FloatingActionButton` is a round button that resides on top of the UI typically in the bottom right hand side.

It has a drop shadow to distinguish it from the UI underneath and it can hide two or more additional actions under the surface. E.g. we can create a simple single click button such as this:

```

FloatingActionButton fab = FloatingActionButton.createFAB(FontImage.MATERIAL_ADD);
fab.addActionListener(e -> ToastBar.showErrorMessage("Not implemented yet..."));
fab.bindFabToContainer(form.getContentPane());

```

Which will place a + sign button that will perform the action. Alternatively we can create a nested action where a click on the button will produce a submenu for users to pick from e.g.:

```

FloatingActionButton fab = FloatingActionButton.createFAB(FontImage.MATERIAL_ADD);
fab.createSubFAB(FontImage.MATERIAL_PEOPLE, "");
fab.createSubFAB(FontImage.MATERIAL_IMPORT_CONTACTS, "");
fab.bindFabToContainer(form.getContentPane());

```

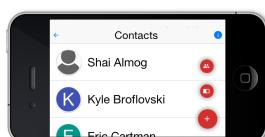


Figure 299. `FloatingActionButton` with submenu expanded

Those familiar with this widget know that there are many nuances to this UI that we might

implement/expose in the future. At the moment we chose to keep the API simple and minimal for the common use cases and refine it based on feedback.

6.39.1. Using Floating Button as a Badge

Floating buttons can also be used to badge an arbitrary component in the style popularized by iOS/Mac OS. A badge appears at the top right corner and includes special numeric details such as "unread count" ..

The code below adds a simple badge to an icon button:

```
Form hi = new Form("Badge");

Button chat = new Button("");
FontImage.setMaterialIcon(chat, FontImage.MATERIAL_CHAT, 7);

FloatingActionButton badge = FloatingActionButton.createBadge("33");
hi.add(badge.bindFabToContainer(chat, Component.RIGHT, Component.TOP));

TextField changeBadgeValue = new TextField("33");
changeBadgeValue.addDataChangedListener((i, ii) -> {
    badge.setText(changeBadgeValue.getText());
    badge.getParent().revalidate();
});
hi.add(changeBadgeValue);

hi.show();
```

The code above results in this, notice you can type into the text field to change the badge value:

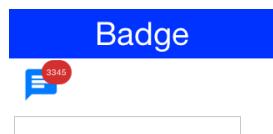


Figure 300. Badge floating button in action

6.40. SplitPane

The split pane component is a bit desktop specific but works reasonably well on devices. To get the image below we changed [SalesDemo.java](#) in the kitchen sink by changing this:

```

private Container encloseInMaximizableGrid(Component cmp1, Component cmp2) {
    GridLayout gl = new GridLayout(2, 1);
    Container grid = new Container(gl);
    gl.setHideZeroSized(true);

    grid.add(encloseInMaximize(grid, cmp1));
    add(encloseInMaximize(grid, cmp2));
    return grid;
}

```

To:

```

private Container encloseInMaximizableGrid(Component cmp1, Component cmp2) {
    return new SplitPane(SplitPane.VERTICAL_SPLIT, cmp1, cmp2, "25%", "50%", "75%");
}

```



Figure 301. Split Pane in the Kitchen Sink Demo

This is mostly self explanatory but only "mostly". We have 5 arguments the first 3 make sense:

- Split orientation
- Components to split

The last 3 arguments seem weird but they also make sense once you understand them, they are:

- The minimum position of the split - 1/4 of available space
- The default position of the split - middle of the screen
- The maximum position of the split - 3/4 of available space

The units don't have to be percentages they can be mm (millimeters) or px (pixels).

[2] String width is the real expensive part here, the complexity of font kerning and the recursion required to reflow text is a big

performance hurdle

[3] Image by RegisFrey - Own work, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=10298177>

[4] Fisheye is an effect where the selection stays in place as the list moves around it

[5] Image was fetched from <http://awoiaf.westeros.org/index.php/Portal:Books>

[6] The text below is from A Wiki of Ice & Fire: http://awoiaf.westeros.org/index.php/A_Game_of_Thrones

7. Animations

There are many ways to animate and liven the data within a Codename One application, layout animations are probably chief among them. But first we need to understand some basics such as layout refows.

7.1. Layout Reflow

Layout in tools such as HTML is implicit, when you add something into the UI it is automatically placed correctly. Other tools such as Codename One use explicit layout, that means you have to explicitly request the UI to layout itself after making changes!



Like many such rules exceptions occur. E.g. if the device is rotated or window size changes a layout will occur automatically.

When adding a component to a UI that is already visible, the component will not show by default.



When adding a component to a form which isn't shown on the screen, there is no need to tell the UI to repaint or reflow. This happens implicitly.

The chief advantage of explicit layout is performance.

E.g. imagine adding 100 components to a form. If the form was laid out automatically, layout would have happened 100 times instead of once when adding was finished. In fact layout refows are often considered the #1 performance issue for HTML/JavaScript applications.



Smart layout reflow logic can alleviate some of the pains of the automatic layout refows however since the process is implicit it's almost impossible to optimize complex usages across browsers/devices. A major JavaScript performance tip is to use absolute positioning which is akin to not using layouts at all!

That is why, when you add components to a form that is already showing, you should invoke `revalidate()` or animate the layout appropriately. This also enables the layout animation behavior explained below.

7.2. Layout Animations

To understand animations you need to understand a couple of things about Codename One components. When we add a component to a container, it's generally just added but not positioned anywhere. A novice might notice the `setX/setY/setWidth/setHeight` methods on a component and just try to position it absolutely.

This won't work since these methods are meant for the layout manager, which is implicitly invoked when a form is shown (internally in Codename One). The layout manager uses these methods to position/size the components based on the hints given to it.

If you add components to a form that is currently showing, it is your responsibility to invoke

`revalidate/layoutContainer` to arrange the newly added components (see [Layout Reflows](#)).

`animateLayout()` method is a fancy form of revalidate that animates the components into their laid out position. After changing the layout & invoking this method the components move to their new sizes/positions seamlessly.

This sort of behavior creates a special case where setting the size/position makes sense. When we set the size/position in the demo code here we are positioning the components at the animation start position above the frame.

```
Form hi = new Form("Layout Animations", new BoxLayout(BoxLayout.Y_AXIS));
Button fall = new Button("Fall"); ①
fall.addActionListener((e) -> {
    for(int iter = 0 ; iter < 10 ; iter++) {
        Label b = new Label ("Label " + iter);
        b.setWidth(fall.getWidth());
        b.setHeight(fall.getHeight());
        b.setY(-fall.getHeight());
        hi.add(b);
    }
    hi.getContentPane().animateLayout(2000); ②
});
hi.add(fall);
```

There are a couple of things that you should notice about this example:

- ① We used a button to do the animation rather than doing it on show. Since `show()` implicitly lays out the components it wouldn't have worked correctly.
- ② We used `hi.getContentPane().animateLayout(2000);` & not `hi.animateLayout(2000);`. You need to animate the "actual" container and `Form` is a special case.

This results in:

Layout Animations

Fall

Layout Animations



Layout Animations



Layout Animations

Label 0
Label 1
Label 2
Label 3
Label 4
Label 5
Label 6
Label 7
Label 8
Label 9

Layout Animations

Label 0
Label 1
Label 2
Label 3
Label 4
Label 5
Label 6
Label 7
Label 8
Label 9

Layout Animations

Fall
Label 0
Label 1
Label 2
Label 3
Label 4
Label 5
Label 6
Label 7
Label 8
Label 9

Layout Animations

Fall

Label 0
Label 1
Label 2
Label 3
Label 4
Label 5
Label 6
Label 7

7.2.1. Unlayout Animations

While layout animations are really powerful effects for adding elements into the UI and drawing attention to them. The inverse of removing an element from the UI is often more important. E.g. when we delete or remove an element we want to animate it out.

Layout animations don't really do that since they will try to bring the animated item into place. What we want is the exact opposite of a layout animation and that is the "unlayout animation".

The "unlayout animation" takes a valid laid out state and shifts the components to an invalid state that we defined in advance. E.g. we can fix the example above to flip the "fall" button into a "rise" button when the buttons come into place and this will allow the buttons to float back up to where they came from in the exact reverse order.



An unlayout animation always leaves the container in an invalidated state.

```
Form hi = new Form("Layout Animations", new BoxLayout(BoxLayout.Y_AXIS));
Button fall = new Button("Fall");
fall.addActionListener((e) -> {
    if(hi.getContentPane().getComponentCount() == 1) {
        fall.setText("Rise");
        for(int iter = 0 ; iter < 10 ; iter++) {
            Label b = new Label ("Label " + iter);
            b.setWidth(fall.getWidth());
            b.setHeight(fall.getHeight());
            b.setY(-fall.getHeight());
            hi.add(b);
        }
        hi.getContentPane().animateLayout(20000);
    } else {
        fall.setText("Fall");
        for(int iter = 1 ; iter < hi.getContentPane().getComponentCount() ; iter++) { ①
            Component c = hi.getContentPane().getComponentAt(iter);
            c.setY(-fall.getHeight()); ②
        }
        hi.getContentPane().animateUnlayoutAndWait(20000, 255); ③
        hi.removeAll(); ④
        hi.add(fall);
        hi.revalidate();
    }
});
hi.add(fall);
```

You will notice some similarities with the unlayout animation but the differences represent the exact opposite of the layout animation:

- ① We loop over existing components (not newly created ones)
- ② We set the desired end position not the desired starting position
- ③ We used the **AndWait** variant of the animate unlayout call. We could have used the **async** call as

well.

- ④ After the animation completes we need to actually remove the elements since the UI is now in an invalid position with elements outside of the screen but still physically there!

7.2.2. Hiding & Visibility

A common trick for animating Components in Codename One is to set their preferred size to `0` and then invoke `animateLayout()` thus triggering an animation to hide said Component. There are several issues with this trick but one of the biggest ones is the fact that `setPreferredSize` has been deprecated for quite a while.

Instead of using that trick you can use `setHidden/isHidden` who effectively encapsulate this functionality and a bit more.

One of the issues `setHidden` tries to solve is the fact that preferred size doesn't include the margin in the total and thus a component might still occupy space despite being hidden. To solve this the margin is set to `0` when hiding and restored to its original value when showing the component again by resetting the UIID (which resets all style modifications).

This functionality might be undesirable which is why there is a version of the `setHidden` method that accepts a boolean flag indicating whether the margin/UIID should be manipulated. You can effectively `hide/show` a component without deprecated code using something like this:

```
Button toHide = new Button("Will Be Hidden");
Button hide = new Button("Hide It");
hide.addActionListener((e) -> {
    hide.setEnabled(false);
    boolean t = !toHide.isHidden();
    toHide.setHidden(t);
    toHide.getParent().animateLayoutAndWait(200);
    toHide.setVisible(!t);
    hide.setEnabled(true);
});
```



Notice that the code above uses `setVisible()`, which shouldn't be confused with `setHidden`. `setVisible()` just toggles the visibility of the component it would still occupy the same amount of space

7.2.3. Synchronicity In Animations

Most animations have two or three variants:

- Standard animation e.g. `animateLayout(int)`
- And wait variant e.g. `animateLayoutAndWait(int)`
- Callback variant e.g. `animateUnlayout(int, int, Runnable)`

The standard animation is invoked when we don't care about the completion of the animation. We

can do this for a standard animation.



The unlayout animations don't have a standard variant. Since they leave the UI in an invalid state we must always do something once the animation completes so a standard variant makes no sense

The `AndWait` variant blocks the calling thread until the animation completes. This is really useful for sequencing animations one after the other e.g this code from the kitchen sink demo:

```
arrangeForInterlace/effects);
effects.animateUnlayoutAndWait(800, 20);
effects.animateLayoutFade(800, 20);
```

First the UI goes thru an "unlayout" animation, once that completes the layout itself is performed.



The `AndWait` calls needs to be invoked on the Event Dispatch Thread despite being "blocking". This is a common convention in Codename One powered by a unique capability of Codename One: `invokeAndBlock`.

You can learn more about `invokeAndBlock` in the `EDT` section [<https://www.codenameone.com/manual/edt.html>].

The callback variant is similar to the `invokeAndBlock` variant but uses a more conventional callback semantic which is more familiar to some developers. It accepts a `Runnable` callback that will be invoked after the fact. E.g. we can change the `unlayout` call from `before` to use the callback semantics as such:

```
hi.getContentPane().animateUnlayout(20000, 255, () -> {
    hi.removeAll();
    hi.add(fall);
    hi.revalidate();
});
```

Animation Fade and Hierarchy

There are several additional variations on the standard animate methods. Several methods accept a numeric `fade` argument. This is useful to fade out an element in an "unlayout" operation or fade in a regular animation.

The value for the `fade` argument is a number between 0 and 255 where 0 represents full transparency and 255 represents full opacity.

Some animate layout methods are hierarchy based. They work just like the regular `animateLayout` methods but recurse into the entire `Container` [<https://www.codenameone.com/javadoc/com/codename1/ui/Container.html>] hierarchy. These methods work well when you have components in a nested hierarchy that need to animate into place. This is demonstrated in the opening sequence of the kitchen sink demo:

```

for(int iter = 0 ; iter < demoComponents.size() ; iter++) {
    Component cmp = (Component)demoComponents.elementAt(iter);
    if(iter < componentsPerRow) {
        cmp.setX(-cmp.getWidth());
    } else {
        if(iter < componentsPerRow * 2) {
            cmp.setX(dw);
        } else {
            cmp.setX(-cmp.getWidth());
        }
    }
}
boxContainer.setShouldCalcPreferredSize(true);
boxContainer.animateHierarchyFade(3000, 30);

```

The `demoComponents Vector` contains components from separate containers and this code would not work with a simple animate layout.



We normally recommend avoiding the hierarchy version. Its slower but more importantly, it's flaky. Since the size/position of the `Container` might be affected by the layout the animation could get clipped and skip. These are very hard issues to debug.

7.2.4. Sequencing Animations Via AnimationManager

All the animations go thru a per-form queue: the [AnimationManager](https://www.codenameone.com/javadoc/com/codename1/ui/AnimationManager.html) [https://www.codenameone.com/javadoc/com/codename1/ui/AnimationManager.html]. This effectively prevents two animations from mutating the UI in parallel so we won't have collisions between two conflicting sides. Things get more interesting when we try to do something like this:

```

cnt.add(myButton);
int componentCount = cnt.getComponentCount();
cnt.animateLayout(300);
cnt.removeComponent(myButton);
if(componentCount == cnt.getComponentCount()) {
    // this will happen...
}

```

The reason this happens is that the second remove gets postponed to the end of the animation so it won't break the animation. This works for remove and add operations on a `Container` [https://www.codenameone.com/javadoc/com/codename1/ui/Container.html] as well as other animations.

The simple **yet problematic** fix would be:

```

cnt.add(myButton);
int componentCount = cnt.getComponentCount();
cnt.animateLayoutAndWait(300);
cnt.removeComponent(myButton);
if(componentCount == cnt.getComponentCount()) {
    // this probably won't happen...
}

```

So why that might still fail?

Events come in constantly during the run of the EDT ^[7], so an event might come in that might trigger an animation in your code. Even if you are on the EDT keep in mind that you don't actually block it and an event might come in.

In those cases an animation might start and you might be unaware of that animation and it might still be in action when you expect remove to work.

Animation Manager to the Rescue

[AnimationManager](https://www.codenameone.com/javadoc/com/codename1/ui/AnimationManager.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/AnimationManager.html>] has builtin support to fix this exact issue.

We can flush the animation queue and run synchronously after all the animations finished and before new ones come in by using something like this:

```

cnt.add(myButton);
int componentCount = cnt.getComponentCount();
cnt.animateLayout(300);
cnt.getAnimationManager().flushAnimation(() -> {
    cnt.removeComponent(myButton);
    if(componentCount == cnt.getComponentCount()) {
        // this shouldn't happen...
    }
});

```

7.3. Low Level Animations

The Codename One event dispatch thread has a special animation “pulse” allowing an animation to update its state and draw itself. Code can make use of this pulse to implement repetitive polling tasks that have very little to do with drawing.

This is helpful since the callback will always occur on the event dispatch thread.

Every component in Codename One contains an `animate()` method that returns a boolean value, you can also implement the [Animation](https://www.codenameone.com/javadoc/com/codename1/ui/animations/Animation.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/animations/Animation.html>] interface in an arbitrary component to implement your own animation. In order to receive animation events you need to register yourself within the parent form, it is the responsibility of the parent for to call `animate()`.

If the `animate` method returns true then the animation will be painted (the `paint` method of the `Animation` interface would be invoked).



It is important to deregister animations when they aren't needed to conserve battery life.

If you derive from a component, which has its own animation logic you might damage its animation behavior by deregistering it, so tread gently with the low level API's.

E.g. you can add additional animation logic using code like this:

```
myForm.registerAnimated(this);

private int spinValue;
public boolean animate() {
    if(userStatusPending) {
        spinValue++;
        super.animate();
        return true;
    }
    return super.animate();
}
```

7.3.1. Why Not Just Write Code In Paint?

Animations are comprised of two parts, the logic (deciding the position etc) and the painting. The `paint` method should be dedicated to painting only, not to the actual moving of the components.

The separation of concerns allows us to avoid redundant painting e.g. if `animate` didn't trigger a change just return `false` to avoid the overhead related to animations.

We discuss low level animations in more details within the [animation section of the clock demo](#) [<https://www.codenameone.com/manual/graphics.html#clock-animation-section>].

7.4. Transitions

Transitions allow us to replace one component with another, most typically forms or dialogs are replaced with a transition however a transition can be applied to replace any arbitrary component.

Developers can implement their own custom transition and install it to components by deriving the `Transition` [<https://www.codenameone.com/javadoc/com/codenname1/ui/animations/Transition.html>] class, although most commonly the built in `CommonTransitions` [<https://www.codenameone.com/javadoc/com/codename1/ui/animations/CommonTransitions.html>] class is used for almost everything.

You can define transitions for forms/dialogs/menus globally either via the theme constants or via the `LookAndFeel` [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/LookAndFeel.html>] class. Alternatively you can install a transition on top-level components via setter methods.

In/Out Transitions

When defining a transition we define the entering transition and the exiting transition. For most cases only one of those is necessary and we default to the exiting (out transition) as a convention.

So for almost all cases the method `setFormTransitonIn` should go unused. That API exists for some elaborate custom transitions that might need to have a special effect both when transitioning in and out of a specific form. However, most of these effects are easier to achieve with layout animations (e.g. components dropping into place etc.).

In the case of `Dialog` the transition in shows its appearance and the transition out shows its disposal. So in that case both transitions make a lot of sense.

Back/Forward Transitions

Transitions have a direction and can all be played either in incoming or outgoing direction. A transition can be flipped (played in reverse) when we use an RTL language ^[8] or when we simply traverse backwards in the form navigation hierarchy.

Normally `Form.show()` displays the next `Form` with an incoming transition based on the current RTL mode. If we use `Form.showBack()` it will play the transition in reverse.



When working with high level animations you can select `Slow Motion` option in the simulator to slow down animations and inspect their details

Themes define the default transitions used when showing a form, these differ based on the OS. In most platforms the default is `Slide` whereas in iOS the default is `SlideFade` which slides the content pane and title while fading in/out the content of the title area.



`SlideFade` is problematic without a title area. If you have a `Form` that lacks a title area we would recommend to disable `SlideFade` at least for that `Form`.

Check out the full set of theme constants in the [Theme Constants Section](#) [<https://www.codenameone.com/manual/advanced-theming.html#theme-constants-section>].

7.4.1. Replace

To apply a transition to a component we can just use the `Container.replace()` method as such:

```
Form hi = new Form("Replace", new BoxLayout(BoxLayout.Y_AXIS));
Button replace = new Button("Replace Pending");
Label replaceDestiny = new Label("Destination Replace");
hi.add(replace);
replace.addActionListener((e) -> {
    replace.getParent().replaceAndWait(replace, replaceDestiny,
CommonTransitions.createCover(CommonTransitions.SLIDE_VERTICAL, true, 800));
    replaceDestiny.getParent().replaceAndWait(replaceDestiny, replace,
CommonTransitions.createUncover(CommonTransitions.SLIDE_VERTICAL, true, 800));
});
```



Replace even works when you have a layout constraint in place e.g. replacing a component in a border layout will do the "right thing". However, some layouts such as `TableLayout` might be tricky in such cases so we recommend wrapping a potentially replaceable `Component` in a border layout and replacing the content.

`Container.replace()` can also be used with a null transition at which point it replaces instantly with no transition.

7.4.2. Slide Transitions

The slide transitions are used to move the `Form/Component` in a sliding motion to the side or up/down. There are 4 basic types of slide transitions:

1. Slide - the most commonly used transition
2. Fast Slide - historically this provided better performance for old device types. It is no longer recommended for newer devices
3. Slide Fade - the iOS default where the title area features a fade transition
4. Cover/Uncover - a type of slide transition where only the source or destination form slides while the other remains static in place

The code below demonstrates the usage of all the main transitions:

```

Toolbar.setGlobalToolbar(true);
Form hi = new Form("Transitions", new BoxLayout(BoxLayout.Y_AXIS));
Style bg = hi.getContentPane().getUnselectedStyle();
bg.setBgTransparency(255);
bg.setBgColor(0xff0000);
Button showTransition = new Button("Show");
Picker pick = new Picker();
pick.setStrings("Slide", "SlideFade", "Cover", "Uncover", "Fade", "Flip");
pick.setSelectedString("Slide");
TextField duration = new TextField("10000", "Duration", 6, TextArea.NUMERIC);
CheckBox horizontal = CheckBox.createToggle("Horizontal");
pick.addActionListener((e) -> {
    String s = pick.getSelectedString().toLowerCase();
    horizontal.setEnabled(s.equals("slide") || s.indexOf("cover") > -1);
});
horizontal.setSelected(true);
hi.add(showTransition);
add(pick);
add(duration);
add(horizontal);

Form dest = new Form("Destination");
bg = dest.getContentPane().getUnselectedStyle();
bg.setBgTransparency(255);
bg.setBgColor(0xff);
dest.setBackCommand(
    dest.getToolbar().addCommandToLeftBar("Back", null, (e) -> hi.showBack()));

showTransition.addActionListener((e) -> {
    int h = CommonTransitions.SLIDE_HORIZONTAL;
    if(!horizontal.isSelected()) {
        h = CommonTransitions.SLIDE_VERTICAL;
    }
    switch(pick.getSelectedString()) {
        case "Slide":
            hi.setTransitionOutAnimator(CommonTransitions.createSlide(h, true, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createSlide(h, true, duration.getAsInt(3000)));
            break;
        case "SlideFade":
            hi.setTransitionOutAnimator(CommonTransitions.createSlideFadeTitle(true, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createSlideFadeTitle(true, duration.getAsInt(3000)));
            break;
        case "Cover":
            hi.setTransitionOutAnimator(CommonTransitions.createCover(h, true, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createCover(h, true, duration.getAsInt(3000)));
            break;
        case "Uncover":
            hi.setTransitionOutAnimator(CommonTransitions.createUncover(h, true, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createUncover(h, true, duration.getAsInt(3000)));
            break;
        case "Fade":
            hi.setTransitionOutAnimator(CommonTransitions.createFade(duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createFade(duration.getAsInt(3000)));
            break;
        case "Flip":
            hi.setTransitionOutAnimator(new FlipTransition(-1, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(new FlipTransition(-1, duration.getAsInt(3000)));
            break;
    }
    dest.show();
});
hi.show();
});

```

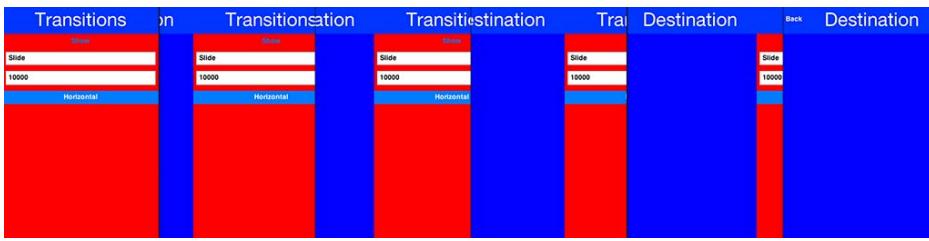


Figure 302. The slide transition moves both incoming and outgoing forms together

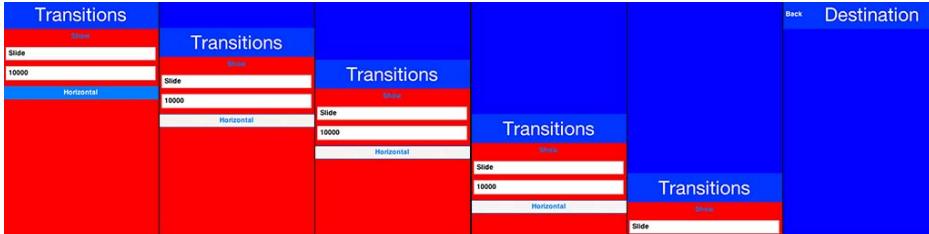


Figure 303. The slide transition can be applied vertically as well



Figure 304. Slide fade fades in the destination title while sliding the content pane it is the default on iOS



SlideFade is problematic without a title area. If you have a **Form** that lacks a title area we would recommend to disable **SlideFade** at least for that **Form**.

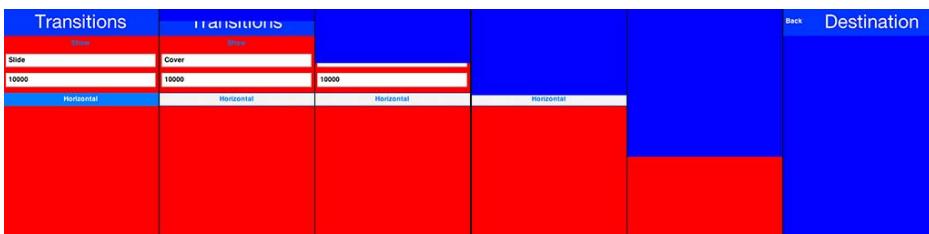


Figure 305. With cover transitions the source form stays in place as it is covered by the destination. This transition can be played both horizontally and vertically

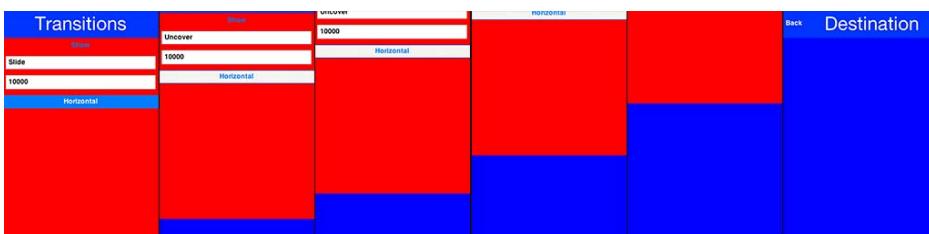


Figure 306. Uncover is the inverse of cover. The destination form stays in place while the departing form moves away

7.4.3. Fade and Flip Transitions

The fade transition is pretty trivial and only accepts a time value since it has no directional context.

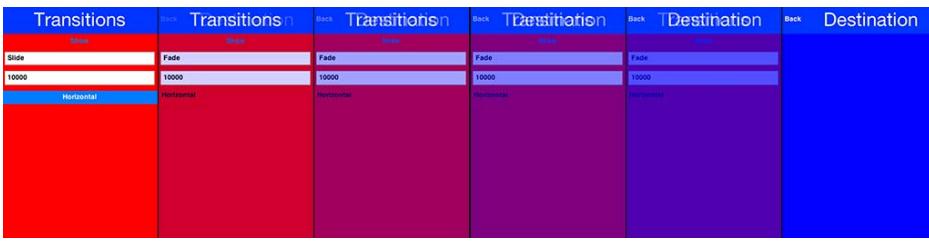


Figure 307. Fade transition is probably the simplest one around

The [FlipTransition](https://www.codenameone.com/javadoc/com/codename1/ui/animations/FlipTransition.html) [https://www.codenameone.com/javadoc/com/codename1/ui/animations/FlipTransition.html] is also pretty simple but unlike the others it isn't a part of the [CommonTransitions](#). It has its own [FlipTransition](#) class.



This transition looks very different on devices as it uses native perspective transforms available only there



Figure 308. Fade transition is probably the simplest one around

7.4.4. Bubble Transition

[BubbleTransiton](https://www.codenameone.com/javadoc/com/codename1/ui/animations/BubbleTransition.html) [https://www.codenameone.com/javadoc/com/codename1/ui/animations/BubbleTransition.html] morphs a component into another component using a circular growth motion.

The [BubbleTransition](#) accepts the component that will grow into the bubble effect as one of its arguments. It's generally designed for [Dialog](#) transitions although it could work for more creative use cases:



The code below manipulates styles and look. This is done to make the code more "self contained". Real world code should probably use the theme

```

Form hi = new Form("Bubble");
Button showBubble = new Button("+");
showBubble.setName("BubbleButton");
Style buttonStyle = showBubble.getAllStyles();
buttonStyle.setBorder(Border.createEmpty());
buttonStyle.setFgColor(0xffffffff);
buttonStyle.setBgPainter((g, rect) -> {
    g.setColor(0xff);
    int actualWidth = rect.getWidth();
    int actualHeight = rect.getHeight();
    int xPos, yPos;
    int size;
    if(actualWidth > actualHeight) {
        yPos = rect.getY();
        xPos = rect.getX() + (actualWidth - actualHeight) / 2;
        size = actualHeight;
    } else {
        yPos = rect.getY() + (actualHeight - actualWidth) / 2;
        xPos = rect.getX();
        size = actualWidth;
    }
    g.setAntiAliased(true);
    g.fillArc(xPos, yPos, size, size, 0, 360);
});
hi.add(showBubble);
hi.setTintColor(0);
showBubble.addActionListener((e) -> {
    Dialog dlg = new Dialog("Bubbled");
    dlg.setLayout(new BorderLayout());
    SpanLabel sl = new SpanLabel("This dialog should appear with a bubble transition from the button", "DialogBody");
    sl.getTextUnselectedStyle().setFgColor(0xffffffff);
    dlg.add(BorderLayout.CENTER, sl);
    dlg.setTransitionInAnimator(new BubbleTransition(500, "BubbleButton"));
    dlg.setTransitionOutAnimator(new BubbleTransition(500, "BubbleButton"));
    dlg.setDisposeWhenPointerOutOfBounds(true);
    dlg.getTitleStyle().setFgColor(0xffffffff);

    Style dlgStyle = dlg.getDialogStyle();
    dlgStyle.setBorder(Border.createEmpty());
    dlgStyle.setBgColor(0xff);
    dlgStyle.setBgTransparency(0xff);
    dlg.showPacked(BorderLayout.NORTH, true);
});
hi.show();

```



Figure 309. Bubble transition converting a circular button to a Dialog

7.4.5. Morph Transitions

Android's material design has a morphing effect where an element from the previous form (activity) animates into a different component on a new activity. Codename One has a morph effect in the [Container](https://www.codenameone.com/javadoc/com/codenmae1/ui/Container.html) [https://www.codenameone.com/javadoc/com/codenmae1/ui/Container.html] class but it doesn't work as a transition between forms and doesn't allow for multiple separate components to transition at once.



Figure 310. Morph Transition

To support this behavior we have the [MorphTransition](https://www.codenameone.com/javadoc/com/codenmae1/ui/animations/MorphTransition.html) [https://www.codenameone.com/javadoc/com/codenmae1/ui/animations/MorphTransition.html] class that provides this same effect coupled with a fade to the rest of the UI (see [Figure 310, “Morph Transition”](#)).

Since the transition is created before the form exists we can't reference explicit components within the form when creating the morph transition (in order to indicate which component becomes which) so we need to refer to them by name. This means we need to use `setName(String)` on the components in the source/destination forms so the transition will be able to find them.

```
Form demoForm = new Form(currentDemo.getDisplayName());
demoForm.setScrollable(false);
demoForm.setLayout(new BorderLayout());
Label demoLabel = new Label(currentDemo.getDisplayName());
demoLabel.setIcon(currentDemo.getDemoIcon());
demoLabel.setName("DemoLabel");
demoForm.addComponent(BorderLayout.NORTH, demoLabel);
demoForm.addComponent(BorderLayout.CENTER, wrapInShelves(n));
....
demoForm.setBackCommand(backCommand);
demoForm.setTransitionOutAnimator(
    MorphTransition.create(3000).morph(
        currentDemo.getDisplayName(),
        "DemoLabel"));
f.setTransitionOutAnimator(
    MorphTransition.create(3000).
        morph(currentDemo.getDisplayName(),
        "DemoLabel"));
demoForm.show();
```

7.4.6. SwipeBackSupport

iOS7+ allows swiping back one form to the previous form, Codenmae One has an API to enable back swipe transition:

```
SwipeBackSupport.bindBack(Form currentForm, LazyValue<Form> destination);
```

That one command will enable swiping back from `currentForm`. `LazyValue` [<https://www.codenameone.com/javadoc/com/codename1/util/LazyValue.html>] allows us to pass a value lazily:

```
public interface LazyValue<T> {  
    public T get(Object... args);  
}
```

This effectively allows us to pass a form and only create it as necessary (e.g. for a GUI builder app we don't have the actual previous form instance), notice that the arguments aren't used for this case but will be used in other cases.

The code below should work for the transition sample above. Notice that this API was designed to work with "Slide Fade" transition and might have issues with other transition types:

```
SwipeBackSupport.bindBack(dest, (args) -> hi);
```

[7] Event Dispatch Thread

[8] Right to left/bidi language such as Hebrew or Arabic

8. The EDT - Event Dispatch Thread

8.1. What Is The EDT

Codename One allows developers to create as many threads as they want; however in order to interact with the Codename One user interface components a developer must use the EDT. The EDT stands for "Event Dispatch Thread" but it handles a lot more than just "events".

The EDT is the main thread of Codename One, by using just one thread Codename One can avoid complex synchronization code and focus on simple functionality that assumes only one thread.



This has huge advantages for your code. You can normally assume that all code will occur on a single thread and avoid complex synchronization logic.

You can visualize the EDT as a loop such as this:

```
while(codenameOneRunning) {  
    performEventCallbacks();  
    performCallSeriallyCalls();  
    drawGraphicsAndAnimations();  
    sleepUntilNextEDTCycle();  
}
```

Normally, every call you receive from Codename One will occur on the EDT. E.g. every event, calls to paint(), lifecycle calls (start etc.) should all occur on the EDT.

This is pretty powerful, however it means that as long as your code is processing nothing else can happen in Codename One!



If your code takes too long to execute then no painting or event processing will occur during that time, so a call to `Thread.sleep()` will actually stop everything!

The solution is pretty simple, if you need to perform something that requires intensive CPU you can spawn a thread.

Codename One's networking code automatically spawns its own network thread (see the [NetworkManager](#) [https://www.codenameone.com/javadoc/com/codename1/io/NetworkManager.html]). However, this also poses a problem...

Codename One assumes all modifications to the UI are performed on the EDT but if we spawned a separate thread. How do we force our modifications back into the EDT?

Codename One includes 3 methods in the [Display](#) [https://www.codenameone.com/javadoc/com/codename1/ui/Display.html] class to help in these situations: `isEDT()`, `callSerially(Runnable)` & `callSeriallyAndWait(Runnable)`.

`isEDT()` is useful for generic code that needs to test whether the current code is executing on the EDT.

8.2. Call Serially (And Wait)

`callSerially(Runnable)` should normally be called off the EDT (in a separate thread), the run method within the submitted runnable will be invoked on the EDT.



The Runnable passed to the `callSerially` and `callSeriallyAndWait` methods is not a `Thread`. We just use the `Runnable` interface as a convenient callback interface.

```
// this code is executing in a separate thread
final String res = methodThatTakesALongTime();
Display.getInstance().callSerially(new Runnable() {
    public void run() {
        // this occurs on the EDT so I can make changes to UI components
        resultLabel.setText(res);
    }
});
```



You can write this code more concisely using Java 8 lambda code as such:

```
// this code is executing in a separate thread
String res = methodThatTakesALongTime();
Display.getInstance().callSerially(() -> resultLabel.setText(res));
```

This allows code to leave the EDT and then later on return to it to perform things within the EDT.

The `callSeriallyAndWait(Runnable)` method blocks the current thread until the method completes, this is useful for cases such as user notification e.g.:

```
// this code is executing in a separate thread
methodThatTakesALongTime();
Display.getInstance().callSeriallyAndWait(() -> {
    // this occurs on the EDT so I can make changes to UI components
    globalFlag = Dialog.show("Are You Sure?", "Do you want to continue?", "Continue", "Stop");
});
// this code is executing the separate thread
// global flag was already set by the call above
if(!globalFlag) {
    return;
}
otherMethod();
```



If you are unsure use `callSerially`. The use cases for `callSeriallyAndWait` are very rare.

8.2.1. callSerially On The EDT

One of the misunderstood topics is why would we ever want to invoke `callSerially` when we are still on the EDT. This is best explained by example. Say we have a button that has quite a bit of functionality tied to its events e.g.:

1. A user added an action listener to show a Dialog.
2. A framework the user installed added some logging to the button.
3. The button repaints a release animation as its being released.

However, this might cause a problem if the first event that we handle (the dialog) might cause an issue to the following events. E.g. a dialog will block the EDT (using `invokeAndBlock`), events will keep happening but since the event we are in "already happened" the button repaint and the framework logging won't occur. This might also happen if we show a form which might trigger logic that relies on the current form still being present.

One of the solutions to this problem is to just wrap the action listeners body with a `callSerially`. In this case the `callSerially` will postpone the event to the next cycle (loop) of the EDT and let the other events in the chain complete. Notice that you shouldn't use this normally since it includes an overhead and complicates application flow, however when you run into issues in event processing we suggest trying this to see if its the cause.



You should never invoke `callSeriallyAndWait` on the EDT since this would effectively mean sleeping on the EDT. We made that method throw an exception if its invoked from the EDT.

8.3. Debugging EDT Violations

There are two types of EDT violations:

1. Blocking the EDT thread so the UI performance is considerably slower.
2. Invoking UI code on a separate thread

Codename One provides a tool to help you detect some of these violations some caveats may apply though...

It's an imperfect tool. It might fire "false positives" meaning it might detect a violation for perfectly legal code and it might miss some illegal calls. However, it is a valuable tool in the process of detecting hard to track bugs that are sometimes only reproducible on the devices (due to race condition behavior).

To activate this tool just select the Debug EDT menu option in the simulator and pick the level of output you wish to receive:

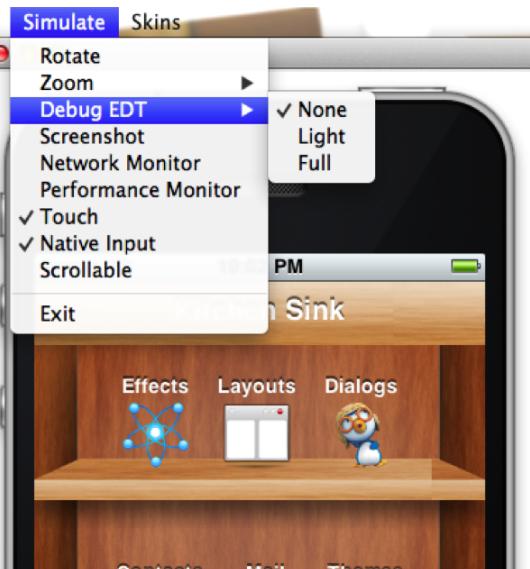


Figure 311. Debug EDT

Full output will include stack traces to the area in the code that is suspected in the violation.

8.4. Invoke And Block

Invoke and block is the exact opposite of `callSeriallyAndWait()`, it blocks the EDT and opens a separate thread for the runnable call. This functionality is inspired by the [Foxtrot](http://foxtrot.sourceforge.net/) [<http://foxtrot.sourceforge.net/>] API, which is a remarkably powerful tool most Swing developers don't know about.

This is best explained by an example. When we write typical code in Java we like that code is in sequence as such:

```
doOperationA();
doOperationB();
doOperationC();
```

This works well normally but on the EDT it might be a problem, if one of the operations is slow it might slow the whole EDT (painting, event processing etc.). Normally we can just move operations into a separate thread e.g.:

```
doOperationA();
new Thread() {
    public void run() {
        doOperationB();
    }
}.start();
doOperationC();
```

Unfortunately, this means that operation C will happen in parallel to operation B which might be a problem...

E.g. instead of using operation names lets use a more "real world" example:

```
updateUIToLoadingStatus();
readAndParseFile();
updateUIWithContentOfFile();
```

Notice that the first and last operations must be conducted on the EDT but the middle operation might be really slow! Since `updateUIWithContentOfFile` needs `readAndParseFile` to occur before it starts doing the new thread won't be enough.

A simplistic approach is to do something like this:

```
updateUIToLoadingStatus();
new Thread() {
    public void run() {
        readAndParseFile();
        updateUIWithContentOfFile();
    }
}).start();
```

But `updateUIWithContentOfFile` should be executed on the EDT and not on a random thread. So the right way to do this would be something like this:

```
updateUIToLoadingStatus();
new Thread() {
    public void run() {
        readAndParseFile();
        Display.getInstance().callSerially(new Runnable() {
            public void run() {
                updateUIWithContentOfFile();
            }
        });
    }
}).start();
```

This is perfectly legal and would work reasonably well, however it gets complicated as we add more and more features that need to be chained serially after all these are just 3 methods!

Invoke and block solves this in a unique way you can get almost the exact same behavior by using this:

```
updateUIToLoadingStatus();
Display.getInstance().invokeAndBlock(new Runnable() {
    public void run() {
        readAndParseFile();
    }
});
updateUIWithContentOfFile();
```

Or this with Java 8 syntax:

```
updateUIToLoadingStatus();
Display.getInstance().invokeAndBlock(() -> readAndParseFile());
updateUIWithContentOfFile();
```

Invoke and block effectively blocks the current EDT in a legal way. It spawns a separate thread that runs the `run()` method and when that run method completes it goes back to the EDT.

All events and EDT behavior still work while `invokeAndBlock` is running, this is because `invokeAndBlock()` keeps calling the main thread loop internally.

 Notice that `invokeAndBlock` comes at a slight performance penalty. Also notice that nesting `invokeAndBlock` calls (or over using them) isn't recommended.
However, they are very convenient when working with multiple threads/UI.

Even if you never call `invokeAndBlock` directly you are probably using it indirectly in API's such as `Dialog` [<https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html>] that show a dialog while blocking the current thread e.g.:

```
public void actionPerformed(ActionEvent ev) {
    // will return true if the user clicks "OK"
    if(!Dialog.show("Question", "How Are You", "OK", "Not OK")) {
        // ask what went wrong...
    }
}
```

Notice that the dialog show method will block the calling thread until the user clicks OK or Not OK...

 Other API's such as `NetworkManager.addToQueueAndWait()` also make use of this feature. Pretty much every "AndWait" method or blocking method uses this API internally!

To explain how `invokeAndBlock` works we can return to the sample above of how the EDT works:

```
while(codenameOneRunning) {
    performEventCallbacks();
    performCallSeriallyCalls();
    drawGraphicsAndAnimations();
    sleepUntilNextEDTCycle();
}
```

`invokeAndBlock()` works in a similar way to this pseudo code:

```
void invokeAndBlock(Runnable r) {  
    openThreadForR(r);  
    while(r is still running) {  
        performEventCallbacks();  
        performCallSeriallyCalls();  
        drawGraphicsAndAnimations();  
        sleepUntilNextEDTCycle();  
    }  
}
```

So the EDT is effectively "blocked" but we "redo it" within the `invokeAndBlock` method...

As you can see this is a very simple approach for thread programming in UI, you don't need to block your flow and track the UI thread. You can just program in a way that seems sequential (top to bottom) but really uses multi-threading correctly without blocking the EDT.

9. Monetization

Codename One tries to make the lives of software developers easier by integrating several forms of built-in monetization solutions such as ad network support, in-app-purchase etc.

A lot of the monetization options are available as 3rd party [cn1lib's](https://www.codenameone.com/cn1libs.html) [<https://www.codenameone.com/cn1libs.html>] that you can install thru the Codename One website.

9.1. Google Play Ads

The most effective network is the simplest banner ad support. To enable mobile ads just [create an ad unit](https://apps.admob.com/?pli=1#monetize/adunit:create) [<https://apps.admob.com/?pli=1#monetize/adunit:create>] in Admob's website, you should end up with the key similar to this:

`ca-app-pub-8610616152754010/3413603324`

To enable this for Android just define the `android.googleAdUnitId=ca-app-pub-8610616152754010/3413603324` in the build arguments and for iOS use the same as in `ios.googleAdUnitId`. The rest is seamless, the right ad will be created for you at the bottom of the screen and the form should automatically shrink to fit the ad. This shrinking is implemented differently between iOS and Android due to some constraints but the result should be similar and this should work reasonably well with device rotation as well.

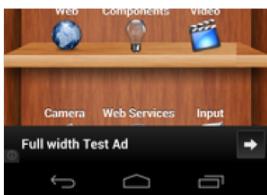


Figure 312. Google play ads

There's a special ad unit id to use for test ads. If you specify `ca-app-pub-3940256099942544/6300978111`, you'll get test ads for your development phase. This is important because you're not allowed to click on your own ads. When it's time to create a production release, you should replace this with the real value you generated in adMob.

9.2. In App Purchase

In-app purchase is a helpful tool for making app development profitable. Codename One supports in-app purchases of consumable/non-consumable products on Android & iOS. It also features support for subscriptions. For such a seemingly simple task, in-app purchase involves a lot of moving parts - especially when it comes to subscriptions.

9.2.1. The SKU

In-app purchase support centers around your set of SKUs that you want to sell. Each product that you sell, whether it be a 1-month subscription, an upgrade to the "Pro" version, "10 disco credits", will have a SKU (stock-keeping-unit). Ideally you will be able to use the same SKU across all the stores that you sell your app in.

9.2.2. Types of Products

There are generally 4 classifications for products:

1. **Non-consumable Product** - This is a product that the user purchases once to "own". They cannot re-purchase it. One example is a product that upgrades your app to a "Pro" version.
2. **Consumable Product** - This is a product that the user can buy more than once. E.g. You might have a product for "10 Credits" that allows the user to buy items in a game.
3. **Non-Renewable Subscription** - A subscription that you buy once, and will not be "auto-renewed" by the app store. These are almost identical to consumable products, except that subscriptions need to be transferable across all the user's devices. This means that non-renewable subscriptions require that you have a server that keeps track of the subscriptions.
4. **Renewable Subscriptions** - A subscription that the app store manages. The user will be automatically billed when the subscription period ends, and the subscription will renew.

 These subscription categories may not be explicitly supported by a given store, or they may carry a different name. You can integrate each product type in a cross platform way using Codename One. E.g. In Google Play there is no distinction between consumable products and non-renewable subscriptions, but in iTunes there is a distinction.

9.2.3. The "Hello World" of In-App Purchase

Let's start with a simple example of an app that sells "Worlds". The first thing we do is pick the SKU for our product. I'll choose "com.codename1.world" for the SKU.

```
public static final String SKU_WORLD = "com.codename1.world";
```



While we chose to use the package name convention for an SKU you can use any name you want e.g **UA8879**

Next, our app's main class needs to implement the **PurchaseCallback** interface

```
public class HelloWorldIAP implements PurchaseCallback {  
    ...  
  
    @Override  
    public void itemPurchased(String sku) {  
        ...  
    }  
  
    @Override  
    public void itemPurchaseError(String sku, String errorMessage) {  
        ...  
    }  
  
    @Override  
    public void itemRefunded(String sku) {  
        ...  
    }  
  
    @Override  
    public void subscriptionStarted(String sku) {  
        ...  
    }  
  
    @Override  
    public void subscriptionCanceled(String sku) {  
        ...  
    }  
  
    @Override  
    public void paymentFailed(String paymentCode, String failureReason) {  
        ...  
    }  
  
    @Override  
    public void paymentSucceeded(String paymentCode, double amount, String currency) {  
        ...  
    }  
}
```

Using these callbacks, we'll be notified whenever something changes in our purchases. For our simple app we're only interested in `itemPurchased()` and `itemPurchaseError()`.

Now in the start method, we'll add a button that allows the user to buy the world:

```

public void start() {
    if(current != null){
        current.show();
        return;
    }
    Form hi = new Form("Hi World");
    Button buyWorld = new Button("Buy World");
    buyWorld.addActionListener(e->{
        if (Purchase.getInAppPurchase().wasPurchased(SKU_WORLD)) {
            Dialog.show("Can't Buy It", "You already Own It", "OK", null);
        } else {
            Purchase.getInAppPurchase().purchase(SKU_WORLD);
        }
    });
    hi.addComponent(buyWorld);
    hi.show();
}

```

At this point, we already have a functional app that will track the sale of the world. To make it more interesting, let's add some feedback with the **ToastBar** to show when the purchase completes.

```

@Override
public void itemPurchased(String sku) {
    ToastBar.showMessage("Thanks. You now own the world", FontImage.MATERIAL_THUMB_UP);
}

@Override
public void itemPurchaseError(String sku, String errorMessage) {
    ToastBar.showErrorMessage("Failure occurred: "+errorMessage);
}

```

 You can test out this code in the simulator without doing any additional setup and it will work. If you want the code to work properly on Android and iOS, you'll need to set up the app and in-app purchase settings in the Google Play and iTunes stores respectively as explained below

When the app first opens we see our button:

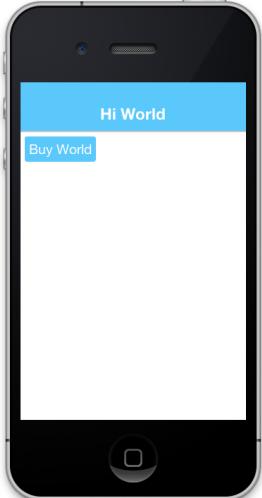


Figure 313. In-app purchase demo app

In the simulator, clicking on the "Buy World" button will bring up a prompt to ask you if you want to approve the purchase.

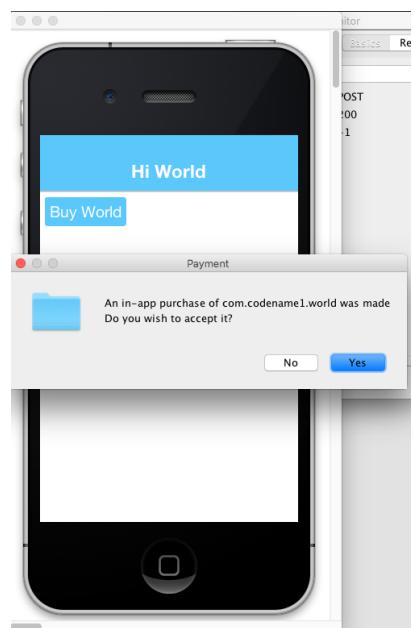


Figure 314. Approving the purchase in the simulator

Now if I try to buy the product again, it pops up the dialog to let me know that I already own it.

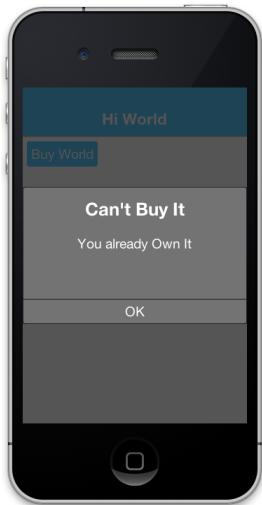


Figure 315. In App purchase already owned

9.2.4. Making it Consumable

In the "Buy World" example above, the "world" product was non-consumable, since we could only buy the world once. We could change it to a consumable product by disregarding whether it was purchased before & keeping track of how many times it had been purchased.

We'll use storage to keep track of the number of worlds that the user purchased. We need two methods to manage this count. One method gets the number of worlds that we own, and another adds a world to this count.

```
private static final String NUM_WORLDS_KEY = "NUM_WORLDS.dat";
public int getNumWorlds() {
    synchronized (NUM_WORLDS_KEY) {
        Storage s = Storage.getInstance();
        if (s.exists(NUM_WORLDS_KEY)) {
            return (Integer)s.readObject(NUM_WORLDS_KEY);
        } else {
            return 0;
        }
    }
}

public void addWorld() {
    synchronized (NUM_WORLDS_KEY) {
        Storage s = Storage.getInstance();
        int count = 0;
        if (s.exists(NUM_WORLDS_KEY)) {
            count = (Integer)s.readObject(NUM_WORLDS_KEY);
        }
        count++;
        s.writeObject(NUM_WORLDS_KEY, new Integer(count));
    }
}
```

Now we'll change our purchase code as follows:

```
buyWorld.addActionListener(e->{
    if (Dialog.show("Confirm", "You own "+getNumWorlds()+
        " worlds. Do you want to buy another one?", "Yes", "No")) {
        Purchase.getInAppPurchase().purchase(SKU_WORLD);
    }
});
```

And our `itemPurchased()` callback will need to add a world:

```
@Override
public void itemPurchased(String sku) {
    addWorld();
    ToastBar.showMessage("Thanks. You now own "+getNumWorlds()+" worlds", FontImage.MATERIAL_THUMB_UP);
}
```



When we set up the products in the iTunes store we will need to mark the product as a consumable product or iTunes will prevent us from purchasing it more than once

9.2.5. Non-Renewable Subscriptions

As we discussed before, there are two types of subscriptions:

1. Non-renewable
2. Auto-renewable

Non-renewable subscriptions are the same as consumable products, except that they are shareable across devices. Auto-renewable subscriptions will continue as long as the user doesn't cancel the subscription. They will be re-billed automatically by the appropriate app-store when the chosen period expires, and the app-store handles the management details itself.



The concept of an "Non-renewable" subscription is unique to iTunes. Google Play has no formal similar option. In order to create a non-renewable subscription SKU that behaves the same in your iOS and Android apps you would create it as a **regular product** in Google play, and a Non-renewable subscription in the iTunes store. We'll learn more about that in a later post when we go into the specifics of app store setup.



The `Purchase` class includes both a `purchase()` method and a `subscribe()` method. On some platforms it makes no difference which one you use, but on Android it matters. If the product is set up as a subscription in Google Play, then you **must** use `subscribe()` to purchase the product. If it is set up as a regular product, then you **must** use `purchase()`. Since we enter "Non-renewable" subscriptions as regular products in the play store, we would use the `purchase()` method.

9.2.6. The Server-Side

Since a subscription purchased on one user device **needs** to be available across the user's devices (Apple's rules for non-renewable subscriptions), our app will need to have a server-component. In this section, we'll gloss over that & "mock" the server interface. We'll go into the specifics of the server-side below.

The Receipts API

Subscriptions, in Codename One use the "Receipts" API. It's up to you to register a receipt store with the In-App purchase instance, which allows Codename one to load receipts (from your server), and submit new receipts to your server. A **Receipt** includes information such as:

1. Store code (since you may be dealing with receipts from itunes, google play & Microsoft)
2. SKU
3. Transaction ID (store specific)
4. Expiry Date
5. Cancellation date
6. Purchase date
7. Order Data (that you can use on the server-side to verify the receipt and load receipt details directly from the store it originated from).

The **Purchase** provides a set of methods for interacting with the receipt store, such as:

1. `isSubscribed([skus])` - Checks to see if the user is currently subscribed to any of the provided skus.
2. `getExpiryDate([skus])` - Checks the expiry date for a set of skus.
3. `synchronizeReceipts()` - Synchronizes the receipts with the receipt store. This will attempt to submit any pending purchase receipts to the receipt store, and the reload receipts from the receipt store.

In order for any of this to work, you must implement the **ReceiptStore** interface, and register it with the Purchase instance. Your receipt store must implement two methods:

1. `fetchReceipts(SuccessCallback<Receipt[]> callback)` - Loads all of the receipts from your receipt store for the current user.
2. `submitReceipt(Receipt receipt, SuccessCallback<Boolean> callback)` - Submits a receipt to your receipt store. This gives you an opportunity to add details to the receipt such as an expiry date.

9.2.7. The "Hello World" of Non-Renewable Subscriptions

We'll expand on the theme of "Buying" the world for this app, except, this time we will "Rent" the world for a period of time. We'll have two products:

1. A 1 month subscription
2. A 1 year subscription

```
public static final String SKU_WORLD_1_MONTH = "com.codename1.world.subscribe.1month";
public static final String SKU_WORLD_1_YEAR = "com.codename1.world.subscribe.1year";

public static final String[] PRODUCTS = {
    SKU_WORLD_1_MONTH,
    SKU_WORLD_1_YEAR
};
```

Notice that we create two separate SKUs for the 1 month and 1 year subscription. **Each subscription period must have its own SKU**. I have created an array (**PRODUCTS**) that contains both of the SKUs. This is handy, as you'll see in the examples ahead, because the APIs for checking status and expiry date of a subscription take the SKUs in a "subscription group" as input.



Different SKUs that sell the same service/product but for different periods form a "subscription group". Conceptually, customers are not subscribing to a particular SKU, they are subscribing to the subscription group of which that SKU is a member. As an example, if a user purchases a 1 month subscription to "the world", they are actually subscribing to "the world" subscription group.

It's up to you to know the grouping of your SKUs. Any methods in the **Purchase** class that check subscription status or expiry date of a SKU should be passed **all** SKUs of that subscription group. E.g. If you want to know if the user is subscribed to the **SKU_WORLD_1_MONTH** subscription, it would not be sufficient to call **iap.isSubscribed(SKU_WORLD_1_MONTH)**, because that wouldn't take into account if the user had purchased a 1 year subscription. The correct way is to always call **iap.isSubscribed(SKU_WORLD_1_MONTH, SKU_WORLD_1_YEAR)**, or simply **iap.isSubscribed(PRODUCTS)** since we have placed both SKUs into our PRODUCTS array.

Implementing the Receipt Store



The receipt store is intended to interface with a server so that the subscriptions can be synced with multiple devices, as required by Apple's guidelines. For this post we'll just store our receipts on device using internal storage. Moving the logic to a server is a simple matter that we will cover in a future post when we cover the server-side.

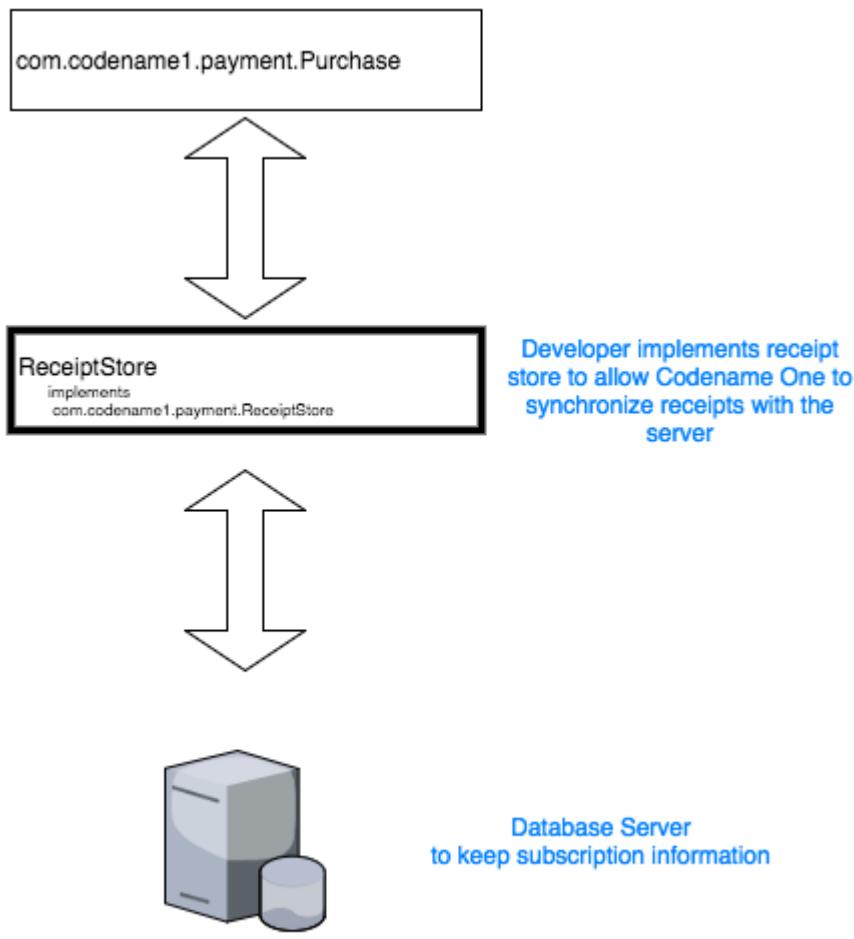


Figure 316. The Receipt store is a layer between your server and Codename One

A basic receipt store needs to implement just two methods:

1. `fetchReceipts`
2. `submitReceipt`

Generally we'll register it in our app's init() method so that it's always available.

```

public void init(Object context) {
    ...
    Purchase.getInAppPurchase().setReceiptStore(new ReceiptStore() {
        @Override
        public void fetchReceipts(SuccessCallback<Receipt[]> callback) {
            // Fetch receipts from storage and pass them to the callback
        }

        @Override
        public void submitReceipt(Receipt receipt, SuccessCallback<Boolean> callback) {
            // Save a receipt to storage. Make sure to call callback when done.
        }
    });
}

```

These methods are designed to be asynchronous since real-world apps will always be connecting to some sort of network service. Therefore, instead of returning a value, both of these methods are passed instances of the `SuccessCallback` class. It's important to make sure to call `callback.onSuccess()` **ALWAYS** when the methods have completed, even if there is an error, or the Purchase class will just assume that you're taking a long time to complete the task, and will continue to wait for you to finish.

Once implemented, our `fetchReceipts()` method will look like:

```

// static declarations used by receipt store

// Storage key where list of receipts are stored
private static final String RECEIPTS_KEY = "RECEIPTS.dat";

@Override
public void fetchReceipts(SuccessCallback<Receipt[]> callback) {
    Storage s = Storage.getInstance();
    Receipt[] found;
    synchronized(RECEIPTS_KEY) {
        if (s.exists(RECEIPTS_KEY)) {
            List<Receipt> receipts = (List<Receipt>)s.readObject(RECEIPTS_KEY);
            found = receipts.toArray(new Receipt[receipts.size()]);
        } else {
            found = new Receipt[0];
        }
    }
    // Make sure this is outside the synchronized block
    callback.onSuccess(found);
}

```

This is fairly straight forward. We're checking to see if we already have a list of receipts stored. If so

we return that list to the callback. If not we return an empty array of receipts.



`Receipt` implements `Externalizable` so you are able to write instances directly to Storage.

The `submitReceipt()` method is a little more complex, as it needs to calculate the new expiry date for our subscription.

```
@Override
public void submitReceipt(Receipt receipt, SuccessCallback<Boolean> callback) {
    Storage s = Storage.getInstance();
    synchronized(RECEIPTS_KEY) {
        List<Receipt> receipts;
        if (s.exists(RECEIPTS_KEY)) {
            receipts = (List<Receipt>)s.readObject(RECEIPTS_KEY);
        } else {
            receipts = new ArrayList<Receipt>();
        }
        // Check to see if this receipt already exists
        // This probably won't ever happen (that we'll be asked to submit an
        // existing receipt, but better safe than sorry
        for (Receipt r : receipts) {
            if (r.getStoreCode().equals(receipt.getStoreCode()) &&
                r.getTransactionId().equals(receipt.getTransactionId())) {
                // If we've already got this receipt, we'll just skip this submission.
                return;
            }
        }
        // Now try to find the current expiry date
        Date currExpiry = new Date();
        List<String> lProducts = Arrays.asList(PRODUCTS);
        for (Receipt r : receipts) {
            if (!lProducts.contains(receipt.getSku())) {
                continue;
            }
            if (r.getCancellationDate() != null) {
                continue;
            }
            if (r.getExpiryDate() == null) {
                continue;
            }
            if (r.getExpiryDate().getTime() > currExpiry.getTime()) {
                currExpiry = r.getExpiryDate();
            }
        }
        // Now set the appropriate expiry date by adding time onto
        // the end of the current expiry date
        Calendar cal = Calendar.getInstance();
```

```

        cal.setTime(currExpiry);
        switch (receipt.getSku()) {
            case SKU_WORLD_1_MONTH:
                cal.add(Calendar.MONTH, 1);
                break;
            case SKU_WORLD_1_YEAR:
                cal.add(Calendar.YEAR, 1);
        }
        Date newExpiry = cal.getTime();

        receipt.setExpiryDate(newExpiry);
        receipts.add(receipt);
        s.writeObject(RECEIPTS_KEY, receipts);

    }
    // Make sure this is outside the synchronized block
    callback.onSuccess(Boolean.TRUE);
}

```

The main logic of this method involves iterating through all of the existing receipts to find the **latest** current expiry date, so that when the user purchases a subscription, it's added onto the end of the current subscription (if one exists) rather than going from today's date. This enables users to safely renew their subscription before the subscription has expired.

In the real-world, we would implement this logic on the server-side.



The iTunes store and Play store have no knowledge of your subscription durations. This is why it's up to you to set the expiry date in the `submitReceipt` method. Non-renewable subscriptions are essentially no different than regular consumable products. It's up to you to manage the subscription logic - and Apple, in particular, requires you to do so using a server.

Synchronizing Receipts

In order for your app to provide you with current data about the user's subscriptions and expiry dates, you need to synchronize the receipts with your receipt store. `Purchase` provides a set of methods for doing this. Generally I'll call one of them inside the `start()` method, and I may resynchronize at other strategic times if I suspect that the information may have changed.

The following methods can be used for synchronization:

1. `synchronizeReceipts()` - Asynchronously synchronizes receipts in the background. You won't be notified when it's complete.
2. `synchronizeReceiptsSync()` - Synchronously synchronizes receipts, and blocks until it's complete. This is safe to use on the EDT as it employs `invokeAndBlock` under the covers.
3. `synchronizeReceipts(final long ifOlderThanMs, final SuccessCallback<Boolean> callback)` - Asynchronously synchronizes receipts, but only if they haven't been synchronized in the specified time period. E.g. In your `start()` method you might decide that you only want to

synchronize receipts once per day. This also includes a callback that will be called when synchronization is complete.

4. `synchronizeReceiptsSync(long ifOlderThanMs)` - A synchronous version that will only refetch if data is older than given time.

In our hello world app we synchronize the subscriptions in a few places.

At the end of the `start()` method:

```
public void start() {  
  
    ...  
  
    // Now synchronize the receipts  
    iap.synchronizeReceipts(0, res->{  
        // Update the UI as necessary to reflect  
  
    });  
}
```

And we also provide a button to allow the user to manually synchronize the receipts.

```
Button syncReceipts = new Button("Synchronize Receipts");  
  
syncReceipts.addActionListener(e->{  
  
    iap.synchronizeReceipts(0, res->{  
        // Update the UI  
    });  
});
```

Expiry Dates and Subscription Status

Now that we have a receipt store registered, and we have synchronized our receipts, we can query the `Purchase` instance to see if a SKU or set of SKUs is currently subscribed. There are three useful methods in this realm:

1. `boolean isSubscribed(String... skus)` - Checks to see if the user is currently subscribed to any of the provided SKUs.
2. `Date getExpiryDate(String... skus)` - Gets the latest expiry date of a set of SKUs.
3. `Receipt getFirstReceiptExpiringAfter(Date dt, String... skus)` - This method will return the earliest receipt with an expiry date after the given date. This is needed in cases where you need to decide if the user should have access to some content based on its publication date. E.g. If you published an issue of your e-zine on March 1, and the user purchased a subscription on March 15th, then they should get access to the March 1st issue even though it doesn't necessarily fall in the subscription period. Being able to easily fetch the first receipt after a given date makes it easier to determine if a particular issue should be covered by a subscription.

If you need to know more information about subscriptions, you can always just call `getReceipts()` to obtain a list of all of the current receipts and determine for yourself what the user should have access to.

In the hello world app we'll use this information in a few different places. On our main form we'll include a label to show the current expiry date, and we allow the user to press a button to synchronize receipts manually if they think the value is out of date.

```
// ...  
  
SpanLabel rentalStatus = new SpanLabel("Loading rental details...");  
Button syncReceipts = new Button("Synchronize Receipts");  
  
syncReceipts.addActionListener(e->{  
  
    iap.synchronizeReceipts(0, res->{  
        if (iap.isSubscribed(PRODUCTS)) {  
            rentalStatus.setText("World rental expires "+iap.getExpiryDate(PRODUCTS));  
        } else {  
            rentalStatus.setText("You don't currently have a subscription to the world");  
        }  
        hi.revalidate();  
    });  
});
```

Allowing the User to Purchase the Subscription

You should now have all of the background required to implement the Hello World Subscription app. So we'll return to the code and see how the user purchases a subscription.

In the main form, we want two buttons to subscribe to the "World", for one month and one year respectively. They look like:

```

Purchase iap = Purchase.getInAppPurchase();
// ...
Button rentWorld1M = new Button("Rent World 1 Month");
rentWorld1M.addActionListener(e->{
    String msg = null;
    if (iap.isSubscribed(PRODUCTS)) { ①
        msg = "You are already renting the world until "
            +iap.getExpiryDate(PRODUCTS) ②
            +". Extend it for one more month?";
    } else {
        msg = "Rent the world for 1 month?";
    }
    if (Dialog.show("Confirm", msg, "Yes", "No")) {
        Purchase.getInAppPurchase().purchase(SKU_WORLD_1_MONTH); ③
        // Note: since this is a non-renewable subscription it is just a regular
        // product in the play store - therefore we use the purchase() method.
        // If it were a "subscription" product in the play store, then we
        // would use subscribe() instead.
    }
});
Button rentWorld1Y = new Button("Rent World 1 Year");
rentWorld1Y.addActionListener(e->{
    String msg = null;
    if (iap.isSubscribed(PRODUCTS)) {
        msg = "You are already renting the world until "
            +iap.getExpiryDate(PRODUCTS) +
            ". Extend it for one more year?";
    } else {
        msg = "Rent the world for 1 year?";
    }
    if (Dialog.show("Confirm", msg, "Yes", "No")) {
        Purchase.getInAppPurchase().purchase(SKU_WORLD_1_YEAR);
        // Note: since this is a non-renewable subscription it is just a regular
        // product in the play store - therefore we use the purchase() method.
        // If it were a "subscription" product in the play store, then we
        // would use subscribe() instead.
    }
});

```

- ① In the event handler we check if the user is subscribed by calling `isSubscribed(PRODUCTS)`. Notice that we check it against the array of both the one month and one year subscription SKUs.
- ② We are able to tell the user when the current expiry date is so that they can gauge whether to proceed.
- ③ Since this is a non-renewable subscription, we use the `Purchase.purchase()` method. See following note about `subscribe()` vs `purchase()`

9.2.8. subscribe() vs purchase()

The `Purchase` class includes two methods for initiating a purchase:

1. `purchase(sku)`
2. `subscribe(sku)`

Which one you use depends on the type of product that is being purchased. If your product is set up as a subscription in the Google Play store, then you should use `subscribe(sku)`. Otherwise, you should use `purchase(sku)`.

Handling Purchase Callbacks

The purchase callbacks are very similar to the ones that we implemented in the regular in-app purchase examples:

```
@Override  
public void itemPurchased(String sku) {  
    Purchase iap = Purchase.getInAppPurchase();  
  
    // Force us to reload the receipts from the store.  
    iap.synchronizeReceiptsSync(0);  
    ToastBar.showMessage("Your subscription has been extended to "+iap.getExpiryDate(PRODUCTS),  
    FontImage.MATERIAL_THUMB_UP);  
}  
  
@Override  
public void itemPurchaseError(String sku, String errorMessage) {  
    ToastBar.showErrorMessage("Failure occurred: "+errorMessage);  
}
```

Notice that, in `itemPurchased()` we don't need to explicitly create any receipts or submit anything to the receipt store. This is handled for you automatically. We do make a call to `synchronizeReceiptsSync()` but this is just to ensure that our toast message has the new expiry date loaded already.

9.2.9. Screenshots



Figure 317. Main form

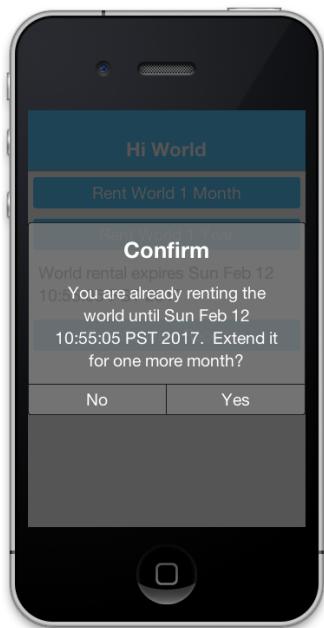


Figure 318. Dialog shown when subscribing to a product

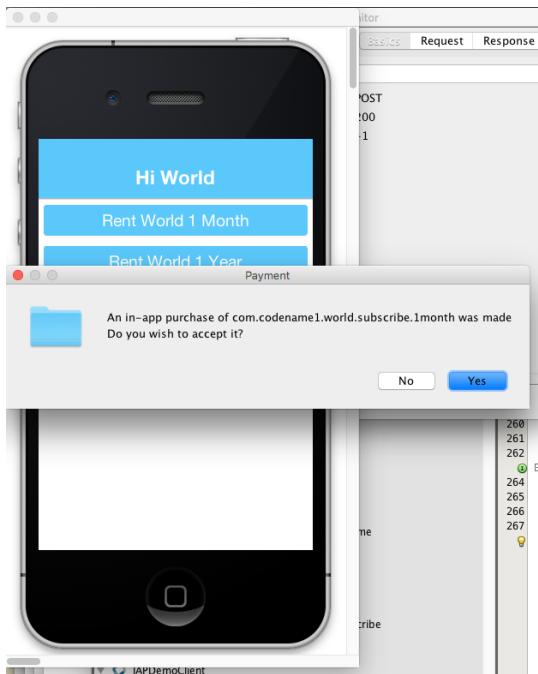


Figure 319. Simulator confirm dialog when purchasing a subscription

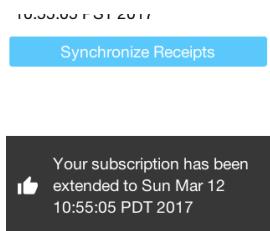


Figure 320. Upon successful purchase, the toastbar message is shown

9.2.10. Summary

This section demonstrated how to set up an app to use non-renewable subscriptions using in-app purchase. Non-renewable subscriptions are the same as regular consumable products except for the fact that they are shared by all of the user's devices, and thus, require a server component. The app store has no knowledge of the duration of your non-renewable subscriptions. It's up to you to specify the expiry date of purchased subscriptions on their receipts when they are submitted. Google play doesn't formally have a "non-renewable" subscription product type. To implement them in Google play, you would just set up a regular product. It's how you handle it internally that makes it a subscription, and not just a regular product.

Codename One uses the [Receipt](#) class as the foundation for its subscriptions infrastructure. You, as the developer, are responsible for implementing the [ReceiptStore](#) interface to provide the receipts. The [Purchase](#) instance will load receipts from your ReceiptStore, and use them to determine whether the user is currently subscribed to a subscription, and when the subscription expires.

9.2.11. Auto-Renewable Subscriptions

Auto-renewable subscriptions provide, arguably, an easier path to recurring revenue than non-renewable subscriptions because all of the subscription stuff is handled by the app store. You defer almost entirely to the app store (iTunes for iOS, and Play for Android) for billing and management.

If there is a down-side, it would be that you are also subject to the rules of each app store - and they

take their cut of the revenue.

1. For more information about Apple's auto-renewable subscription features and rules see [this document](#) [<https://developer.apple.com/app-store/subscriptions/>].
2. For more information about subscriptions in Google play, see [this document](#) [https://developer.android.com/google/play/billing/billing_subscriptions.html].

9.2.12. Auto-Renewable vs Non-Renewable. Best Choice?

When deciding between auto-renewable and non-renewable subscriptions, as always, the answer will depend on your needs and preferences. Auto-renewables are nice because it takes the process completely out of your hands. You just get paid. On the other hand, there are valid reasons to want to use non-renewables. E.g. You can't cancel an auto-renewable subscription for a user. They have to do that themselves. You may also want more control over the subscription and renewal process, in which case a non-renewable might make more sense.



There are some developers [that are opposed to auto-renewable subscriptions](#) [<https://marco.org/2013/12/02/auto-renewable-subscriptions>], we don't have enough information to make a solid recommendation on this matter

On a practical level, if you are using auto-renewable subscriptions (and therefore subscription products in the Google play store) you must use the `Purchase.subscribe(sku)` method for initiating the purchase workflow. For non-renewable subscriptions (and therefore regular products in the Google play store), you must use the `Purchase.purchase(sku)` method.

9.2.13. Learning By Example

In this section we'll describe the general workflow of subscription management on the server. We also demonstrate how use Apple's and Google's web services to validate receipts and stay informed of important events (such as when users cancel or renew their subscriptions).

9.2.14. Building the IAP Demo Project

To aid in this process, we've created a fully-functional in-app purchase demo project that includes both a [client app](#) [<https://gist.github.com/shannah/b61b9b6b35ea0eac923a54163f5d4deb>] and a [server app](#) [<https://github.com/shannah/cn1-iap-demo-server>].

Setting up the Client Project

1. Create a new Codename One project in Netbeans, and choose the "Bare-bones Hello World Template". You should make your package name something unique so that you are able to create real corresponding apps in both Google Play and iTunes connect.
2. Once the project is created, copy [this source file](#) [<https://gist.github.com/shannah/b61b9b6b35ea0eac923a54163f5d4deb>] contents into your main class file. Then change the package name, and class name in the file to match your project settings. E.g. change `package ca.weblite.iapdemo;` to `package <your.package.name.here>;` and `class IAPDemo implements PurchaseCallback` to `class YourClassName implements PurchaseCallback.`

3. Add the [Generic Web Service Client](https://github.com/shannah/cn1-generic-webservice-client) [https://github.com/shannah/cn1-generic-webservice-client] library to your project by going to "Codename Settings" > "Extensions", finding that library, and click "Download". Then "Refresh CN1 libs" as it suggests.
4. Change the `localhost` property to point to your local machine's network address. Using "http://localhost" is not going to cut it here because when the app is running on a phone, it needs to be able to connect to your web server over the network. This address will be your local network address (e.g. 192.168.0.9, or something like that).

```
private static final String localhost = "http://10.0.1.32";
```

5. Add the `ios.plistInject` build hint to your project with the value "`<key>NSAppTransportSecurity</key> <dict> <key>NSAllowsArbitraryLoads</key> <true/> </dict>`". This is so that we can use http urls in iOS. Since we don't intend to fully publish this app, we can cut corners like this. If you were creating a real app, you would use proper secure URLs.

Setting up the Server Project

Download the CN1-IAP-Server demo project from Github, and run its "install-deps" ANT task in order to download and install its dependencies to your local Maven repo.



For the following commands to work, make sure you have "ant", "mvn", and "git" in your environment PATH.

```
$ git clone https://github.com/shannah/cn1-iap-demo-server  
$ cd cn1-iap-demo-server  
$ ant install-deps
```

Open the project in Netbeans

Setting up the Database

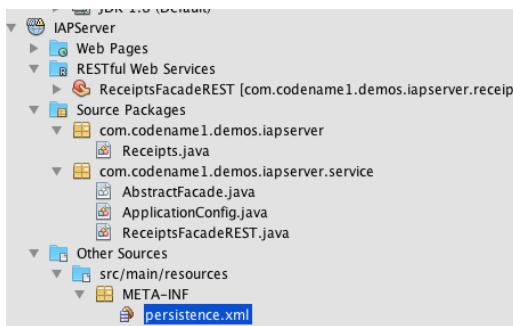
1. Create a new database in your preferred DBMS. Call it anything you like.
2. Create a new table named "RECEIPTS" in this database with the following structure:

```

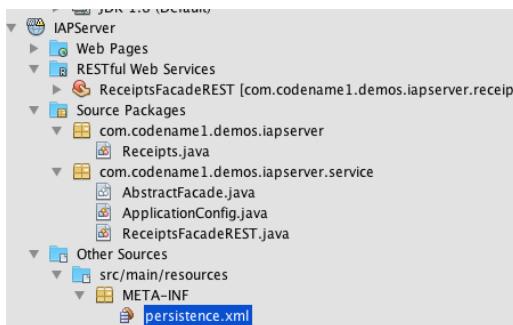
create TABLE RECEIPTS
(
    TRANSACTION_ID VARCHAR(128) not null,
    USERNAME VARCHAR(64) not null,
    SKU VARCHAR(128) not null,
    ORDER_DATA VARCHAR(32000),
    PURCHASE_DATE BIGINT,
    EXPIRY_DATE BIGINT,
    CANCELLATION_DATE BIGINT,
    LAST_VALIDATED BIGINT,
    STORE_CODE VARCHAR(20) default '' not null,
    primary key (TRANSACTION_ID, STORE_CODE)
)

```

3. Open the "persistence.xml" file in the server netbeans project.



4. Change the data source to the database you just created.



If you're not sure how to create a data source, see my [previous tutorial on connecting to a MySQL database](https://www.codenameone.com/blog/connecting-to-a-mysql-database-part-2.html) [<https://www.codenameone.com/blog/connecting-to-a-mysql-database-part-2.html>].

Testing the Project

At this point we should be able to test out the project in the Codename One simulator to make sure it's working.

1. Build and Run the server project in Netbeans. You may need to tell it which application server you wish to run it on. I am running it on the Glassfish 4.1 that comes bundled with Netbeans.
2. Build and run the client project in Netbeans. This should open the Codename One simulator.

When the app first opens you'll see a screen as follows:

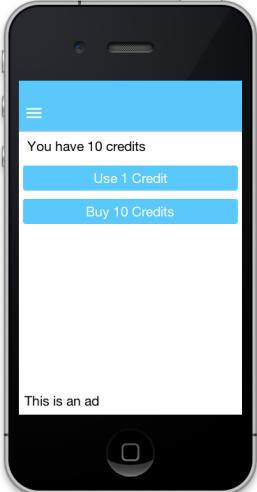


Figure 321. First screen of app

This screen is for testing consumable products, so we won't be making use of this right now.

Open the hamburger menu and select "Subscriptions". You should see something like this:

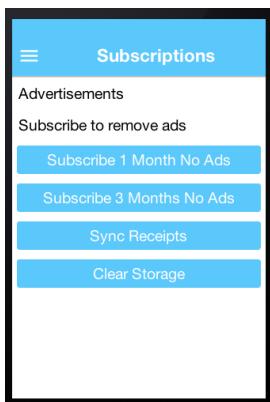


Figure 322. Subscriptions form

Click on the "Subscribe 1 Month No Ads" button. You will be prompted to accept the purchase:

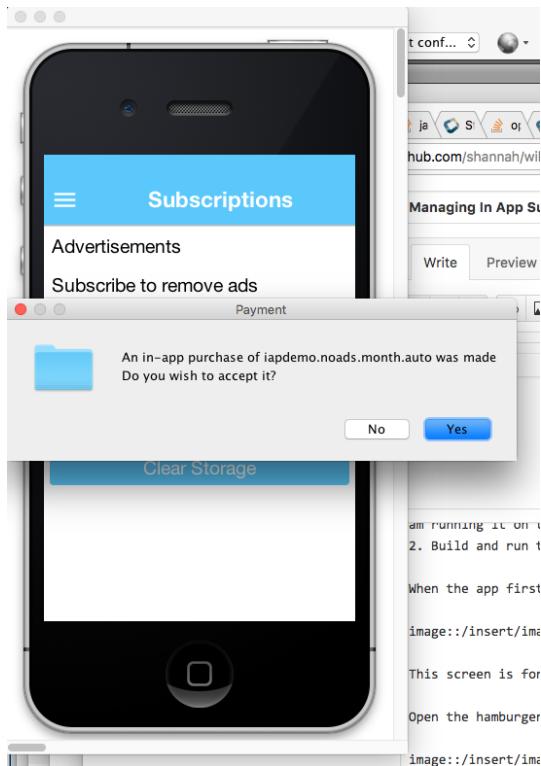


Figure 323. Approve purchase dialog

Upon completion, the app will submit the purchase to your server, and if all went well, it will retrieve the updated list of receipts from your server also, and update the label on this form to say "No Ads. Expires <some date>":

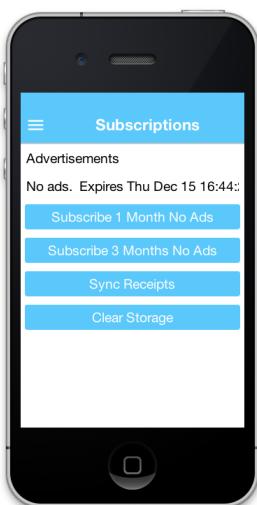


Figure 324. After successful purchase



This project is set up to use an expedited expiry date schedule for purchases from the simulator. 1 month = 5 minutes. 3 months = 15 minutes. This helps for testing. That is why your expiry date may be different than expected.

Just to verify that the receipt was inserted correctly, you should check the contents of your "RECEIPTS" table in your database. In Netbeans, I can do this easily from the "Services" pane. Expand the database connection down to the RECEIPTS table, right click "RECEIPTS" and select "View Data". This will open a data table similar the the following:

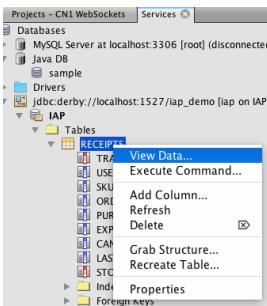


Figure 325. Receipts table after insertion

TRANSACTION_ID	USERNAME	ORDER_DATE	PURCHASE_DATE	CANCELLATION_DATE	AMT_UNPAID	STATE_CODE
1	admin	1481311115700	1481311115700		0.00	simulator
2	admin	1481311115700	1481311115700		0.00	simulator
3	admin	1481311115700	1481311115700		0.00	simulator
4	admin	1481311115700	1481311115700		0.00	simulator
5	admin	1481311115700	1481311115700		0.00	simulator
6	admin	1481311115700	1481311115700		0.00	simulator
7	admin	1481311115700	1481311115700		0.00	simulator
8	admin	1481311115700	1481311115700		0.00	simulator
9	admin	1481311115700	1481311115700		0.00	simulator
10	admin	1481311115700	1481311115700		0.00	simulator

Figure 326. Table view

A few things to mention here:

1. The "username" was provided by the client. It's hard-coded to "admin", but the idea is that you would have the user log in and you would have access to their real username.
2. All dates are stored as unix timestamps in milliseconds.

If you delete the receipt from your database, then press the "Synchronize Receipts" button in your app, the app will again say "No subscriptions." Similarly if you wait 5 minutes and hit "Synchronize receipts" the app will say no subscriptions found, and the "ads" will be back.

Troubleshooting

Let's not pretend that everything worked for you on the first try. There's a lot that could go wrong here. If you make a purchase and nothing appears to happen, the first thing you should do is check the Network Monitor in the simulator ("Simulate" > "Network" > "Network Monitor"). You should see a list of network requests. Some will be GET requests and there will be at least one POST request. Check the response of these requests to see if they succeeded.

Also check the Glassfish server log to see if there is an exception.

Common problems would be that the URL you have set in the client app for `endpointURL` is incorrect, or that there is a database connection problem.

9.2.15. Looking at the Source of the App

Now that we've set up and built the app, let's take a look at the source code so you can see how it all works.

Client Side

I use the [Generic Webservice Client Library](https://github.com/shannah/cn1-generic-webservice-client) [<https://github.com/shannah/cn1-generic-webservice-client>]

from inside my `ReceiptStore` implementation to load receipts from the web service, and insert new receipts to the database.

The source for my `ReceiptStore` is as follows:

```
private ReceiptStore createReceiptStore() {
    return new ReceiptStore() {

        RESTfulWebServiceClient client = createRESTClient(receiptsEndpoint);

        @Override
        public void fetchReceipts(SuccessCallback<Receipt[]> callback) {
            RESTfulWebServiceClient.Query query = new RESTfulWebServiceClient.Query() {

                @Override
                protected void setupConnectionRequest(RESTfulWebServiceClient client, ConnectionRequest req) {
                    super.setupConnectionRequest(client, req);
                    req.setUrl(receiptsEndpoint);
                }

            };
            client.find(query, rowset->{
                List<Receipt> out = new ArrayList<Receipt>();
                for (Map m : rowset) {
                    Result res = Result.fromContent(m);
                    Receipt r = new Receipt();
                    r.setTransactionId(res.getAsString("transactionId"));
                    r.setPurchaseDate(new Date(res.getAsLong("purchaseDate")));
                    r.setQuantity(1);
                    r.setStoreCode(m.getAsString("storeCode"));
                    r.setSku(res.getAsString("sku"));

                    if (m.containsKey("cancellationDate") && m.get("cancellationDate") != null) {
                        r.setCancellationDate(new Date(res.getAsLong("cancellationDate")));
                    }
                    if (m.containsKey("expiryDate") && m.get("expiryDate") != null) {
                        r.setExpiryDate(new Date(res.getAsLong("expiryDate")));
                    }
                    out.add(r);
                }
                callback.onSuccess(out.toArray(new Receipt[out.size()]));
            });
        }

        @Override
        public void submitReceipt(Receipt r, SuccessCallback<Boolean> callback) {
            Map m = new HashMap();
            m.put("transactionId", r.getTransactionId());
            m.put("sku", r.getSku());
            m.put("purchaseDate", r.getPurchaseDate().getTime());
            m.put("orderData", r.getOrderData());
            m.put("storeCode", r.getStoreCode());
            client.create(m, callback);
        }

    };
}
```

Notice that we are not doing any calculation of expiry dates in our client app, as we did in the previous post (on non-renewable receipts). Since we are using a server now, it makes sense to move all of that logic over to the server.

The `createRESTClient()` method shown there simply creates a `RESTfulWebServiceClient` and configuring it to use basic authentication with a username and password. The idea is that your user would have logged into your app at some point, and you would have a username and password on hand to pass back to the web service with the receipt data so that you can connect the subscription to a user account. The source of that method is listed here:

```
/*
 * Creates a REST client to connect to a particular endpoint.  The REST client
 * generated here will automatically add the Authorization header
 * which tells the service what platform we are on.
 * @param url The url of the endpoint.
 * @return
 */
private RESTfulWebServiceClient createRESTClient(String url) {
    return new RESTfulWebServiceClient(url) {

        @Override
        protected void setupConnectionRequest(ConnectionRequest req) {
            try {
                req.addRequestHeader("Authorization", "Basic " +
Base64.encode((getUsername() + ":" + getPassword()).getBytes("UTF-8")));
            } catch (Exception ex) {}
        }

    };
}
```

Server-Side

On the server-side, our REST controller is a standard JAX-RS REST interface. I used Netbeans web service wizard to generate it and then modified it to suit my purposes. The methods of the `ReceiptsFacadeREST` class pertaining to the REST API are shown here:

```

@Stateless
@Path("com.codename1.demos.iapserver.receipts")
public class ReceiptsFacadeREST extends AbstractFacade<Receipts> {

    // ...

    @POST
    @Consumes({"application/xml", "application/json"})
    public void create(Receipts entity) {

        String username = credentialsWithBasicAuthentication(request).getName();
        entity.setUsername(username);

        // Save the receipt first in case something goes wrong in the validation stage
        super.create(entity);

        // Let's validate the receipt
        validateAndSaveReceipt(entity);
        // validates the receipt against appropriate web service
        // and updates database if expiry date has changed.
    }

    // ...
    @GET
    @Override
    @Produces({"application/xml", "application/json"})
    public List<Receipts> findAll() {
        String username = credentialsWithBasicAuthentication(request).getName();
        return getEntityManager()
            .createNamedQuery("Receipts.findByUsername")
            .setParameter("username", username)
            .getResultList();
    }
}

```

The magic happens inside that `validateAndSaveReceipt()` method, which I'll cover in detail soon.

Notifications

It's important to note that you will not be notified by apple or google when changes are made to subscriptions. It's up to you to periodically "poll" their web service to find if any changes have been made. Changes we would be interested in are primarily renewals and cancellations. In order to deal with this, set up a method to run periodically (once-per day might be enough). For testing, I actually set it up to run once per minute as shown below:

```

private static final long ONE_DAY = 24 * 60 * 60 * 1000;
private static final long ONE_DAY_SANDBOX = 10 * 1000;
@Schedule(hour="*", minute="*")
public void validateSubscriptionsCron() {
    System.out.println("----- DOING TIMED TASK -----");
    List<Receipts> res = null;
    final Set<String> completedTransactionIds = new HashSet<String>();
    for (String storeCode : new String[]{Receipt.STORE_CODE_ITUNES, Receipt.STORE_CODE_PLAY}) {
        while (!(res = getEntityManager().createNamedQuery("Receipts.findNextToValidate")
            .setParameter("threshold", System.currentTimeMillis() - ONE_DAY_SANDBOX)
            .setParameter("storeCode", storeCode)
            .setMaxResults(1)
            .getResultList()).isEmpty() &&
            !completedTransactionIds.contains(res.get(0).getTransactionId())))
    }

    final Receipts curr = res.get(0);
    completedTransactionIds.add(curr.getTransactionId());
    Receipts[] validatedReceipts = validateAndSaveReceipt(curr);
    em.flush();
    for (Receipts r : validatedReceipts) {
        completedTransactionIds.add(r.getTransactionId());
    }
}

}
}

```

That method simply finds all of the receipts in the database that haven't been validated in some period of time, and validates it. Again, the magic happens inside the `validateAndSaveReceipt()` method which we cover later.



In this example we only validate receipts from the iTunes and Play stores because those are the only ones that we currently support auto-renewing subscriptions on.

9.2.16. The CN1-IAP-Validator Library

For the purpose of this tutorial, I created a library to handle receipt validation in a way that hides as much of the complexity as possible. It supports both Google Play receipts and iTunes receipts.

The general usage is as follows:

```

IAPValidator validator = IAPValidator.getValidatorForPlatform(receipt.getStoreCode());
if (validator == null) {
    // no validators were found for this store
    // Do custom validation
} else {
    validator.setAppleSecret(APPLE_SECRET);
    validator.setGoogleClientId(GOOGLE_DEVELOPER_API_CLIENT_ID);
    validator.setGooglePrivateKey(GOOGLE_DEVELOPER_PRIVATE_KEY);
    Receipt[] result = validator.validate(receipt);
    ...
}

```

As you can see from this snippet, the complexity of receipt validation has been reduced to entering three configuration strings:

1. **APPLE_SECRET** - This is a "secret" string that you will get from iTunes connect when you set up your in-app products.
2. **GOOGLE_DEVELOPER_API_CLIENT_ID** - A client ID that you'll get from the google developer API console when you set up your API service credentials.
3. **GOOGLE_DEVELOPER_PRIVATE_KEY** - A PKCS8 encoded string with an RSA private key that you'll receive at the same time as the **GOOGLE_DEVELOPER_API_CLIENT_ID**.

I will go through the steps to obtain these values soon.

9.2.17. The `validateAndSaveReceipt()` Method

You are now ready to see the full magic of the `validateAndSaveReceipt()` method in all its glory:

```

/**
 * Validates a given receipt, updating the expiry date,
 * @param receipt The receipt to be validated
 * @param forInsert If true, then an expiry date will be calculated even if there is no validator.
 */
private Receipts[] validateAndSaveReceipt(Receipts receipt) {
    EntityManager em = getEntityManager();
    Receipts managedReceipt = getManagedReceipt(receipt);
    // managedReceipt == receipt if receipt is in database or null otherwise

    if (Receipt.STORE_CODE_SIMULATOR.equals(receipt.getStoreCode())) { ①
        if (receipt.getExpiryDate() == null && managedReceipt == null) {
            //Not inserted yet and no expiry date set yet
            Date dt = calculateExpiryDate(receipt.getSku(), true);
            if (dt != null) {
                receipt.setExpiryDate(dt.getTime());
            }
        }
        if (managedReceipt == null) {
            // Receipt is not in the database yet. Add it
            em.persist(receipt);
            return new Receipts[]{receipt};
        }
    }
}

```

```

    } else {
        // The receipt is already in the database. Update it.
        em.merge(managedReceipt);
        return new Receipts[]{managedReceipt};
    }
} else {
    // It's not a simulator receipt
    IAPValidator validator = IAPValidator.getValidatorForPlatform(receipt.getStoreCode());
    if (validator == null) {
        // Receipt must have come from a platform other than iTunes or Play
        // Because there is no validator

        if (receipt.getExpiryDate() == null && managedReceipt == null) {
            // No expiry date.
            // Generate one.
            Date dt = calculateExpiryDate(receipt.getSku(), false);
            if (dt != null) {
                receipt.setExpiryDate(dt.getTime());
            }
        }
        if (managedReceipt == null) {
            em.persist(receipt);
            return new Receipts[]{receipt};
        } else {
            em.merge(managedReceipt);
            return new Receipts[]{managedReceipt};
        }
    }
}

// Set credentials for the validator
validator.setAppleSecret(APPLE_SECRET);
validator.setGoogleClientId(GOOGLE_DEVELOPER_API_CLIENT_ID);
validator.setGooglePrivateKey(GOOGLE_DEVELOPER_PRIVATE_KEY);

// Create a dummy receipt with only transaction ID and order data to pass
// to the validator. Really all it needs is order data to be able to validate
Receipt r2 = Receipt();
r2.setTransactionId(receipt.getTransactionId());
r2.setOrderData(receipt.getOrderData());
try {
    Receipt[] result = validator.validate(r2);
    // Depending on the platform, result may contain many receipts or a single receipt
    // matching our receipt. In the case of iTunes, none of the receipt transaction IDs
    // might match the original receipt's transactionId because the validator
    // will set the transaction ID to the *original* receipt's transaction ID.
    // If none match, then we should remove our receipt, and update each of the returned
    // receipts in the database.
    Receipt matchingValidatedReceipt = null;
    for (Receipt r3 : result) {
        if (r3.getTransactionId().equals(receipt.getTransactionId())) {
            matchingValidatedReceipt = r3;
            break;
        }
    }
}

```

```

        if (matchingValidatedReceipt == null) {
            // Since the validator didn't find our receipt,
            // we should remove the receipt. The equivalent
            // is stored under the original receipt's transaction ID
            if (managedReceipt != null) {
                em.remove(managedReceipt);
                managedReceipt = null;
            }
        }
        List<Receipts> out = new ArrayList<Receipts>();
        // Now go through and
        for (Receipt r3 : result) {
            if (r3.getOrderData() == null) {
                // No order data found in receipt. Setting it to the original order data
                r3.setOrderData(receipt.getOrderData());
            }
            Receipts eReceipt = new Receipts();
            eReceipt.setTransactionId(r3.getTransactionId());
            eReceipt.setStoreCode(receipt.getStoreCode());
            Receipts eManagedReceipt = getManagedReceipt(eReceipt);
            if (eManagedReceipt == null) {
                copy(eReceipt, r3);
                eReceipt.setUsername(receipt.getUsername());
                eReceipt.setLastValidated(System.currentTimeMillis());
                em.persist(eReceipt);
                out.add(eReceipt);
            } else {
                copy(eManagedReceipt, r3);
                eManagedReceipt.setUsername(receipt.getUsername());
                eManagedReceipt.setLastValidated(System.currentTimeMillis());
                em.merge(eManagedReceipt);
                out.add(eManagedReceipt);
            }
        }
    }

    return out.toArray(new Receipts[out.size()]);
}

} catch (Exception ex) {
    // We should probably store some info about the failure in the
    // database to make it easier to find receipts that aren't validating,
    // but for now we'll just log it.
    Log.p("Failed to validate receipt "+r2);
    Log.p("Reason: "+ex.getMessage());
    Log.e(ex);
    return new Receipts[]{receipt};
}
}
}
}

```

- ① We need to handle the case where the app is being used in the CN1 simulator. We'll treat this as a non-renewable receipt, and we'll calculate the expiry date using an "accelerated" clock to assist in testing.



In many of the code snippets for the Server-side code, you'll see references to both a `Receipts` class and a `Receipt` class. I know this is slightly confusing. The `Receipts` class is a JPA entity that encapsulates a row from the "receipts" table of our SQL database. The `Receipt` class is `com.codename1.payment.Receipt`. It's used to interface with the IAP validation library.

9.2.18. Google Play Setup

Creating the App in Google Play

In order to test out in-app purchase on an Android device, you'll need to create an app in the [Google Play Developer Console](https://play.google.com/apps/publish/) [<https://play.google.com/apps/publish/>]. I won't describe the process in this section, but there is plenty of information around the internet on how to do this. Some useful references for this include:

1. [Getting Started With Publishing](https://developer.android.com/distribute/googleplay/start.html) [<https://developer.android.com/distribute/googleplay/start.html>] - If you don't already have an account with Google to publish your apps.
2. [Launch Checklist](https://developer.android.com/distribute/tools/launch-checklist.html) [<https://developer.android.com/distribute/tools/launch-checklist.html>]

Graphics, Icons, etc..

You are required to upload some screenshots and feature graphics. Don't waste time making these perfect. For the screenshots, you can just use the "Screenshot" option in the simulator. (Use the Nexus 5 skin). For the feature graphics, I used [this site](https://www.norio.be/android-feature-graphic-generator/) [<https://www.norio.be/android-feature-graphic-generator/>] that will generate the graphics in the correct dimensions for Google Play. You can also just leave the icon as the default Codename One icon.

Creating Test Accounts



You cannot purchase in-app products from your app using your publisher account. You need to set up at least one test account for the purpose of testing the app.

In order to test your app, you need to set up a test account. A test account must be associated with a real gmail email address. If you have a domain that is managed by Google apps, then you can also use an address from that domain.

The full process for testing in-app billing can be found in [this google document](https://developer.android.com/google/play/billing/billing_testing.html) [https://developer.android.com/google/play/billing/billing_testing.html]. However, I personally found this documentation difficult to follow.

For your purposes, you'll need to set up a tester list in Google Play. Choose "Settings" > "Tester Lists". Then create a list with all of the email address that you want to have treated as test accounts. Any purchases made by these email addresses will be treated as "Sandbox" purchases, and won't require real money to change hands.

Alpha Channel Distribution

In order to test in-app purchase on Android, you **must** first publish your app. You can't just build and install your app manually. The app needs to be published on the Play store, and it must be

installed **through** the play store for in-app purchase to work. Luckily you can publish to an Alpha channel so that your app won't be publicly available.

For more information about setting up alpha testing on Google play see [this Google support document on the subject](https://support.google.com/googleplay/android-developer/answer/3131213?hl=en) [<https://support.google.com/googleplay/android-developer/answer/3131213?hl=en>].

Once you have set your app up for alpha testing, you can send an invite link to your test accounts. You can find the link in the Google Play console under the APK section, under the "Alpha" tab (and assuming you've enabled alpha testing).

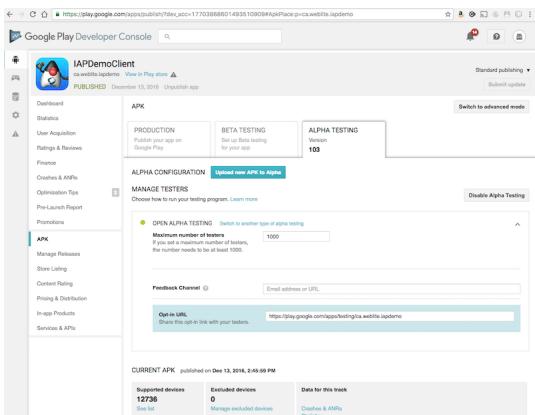


Figure 327. Alpha testing tab in google play

The format of the link is <https://play.google.com/apps/testing/your-app-id> in case you can't find it. You can email this to your alpha testers. Make sure that you have added all testers to your tester lists so that their purchases will be made in the sandbox environment.

Also, before proceeding with testing in-app purchases, you need to add the in-app products in Google Play.

Adding In-App Products

After you have published your APK to the alpha channel, you can create the products. For the purposes of this tutorial, we'll just add two products:

1. **iapdemo.noads.month.auto** - The 1 month subscription.
2. **iapdemo.noads.3month.auto** - The 3 month subscription.

Since we will be adding products as "Subscriptions" in the pay store, your app **must** use the `Purchase.subscribe(sku)` method for initiating a purchase on these products, and **not** the `Purchase.purchase(sku)` method. If you accidentally use `purchase()` to purchase a subscription on Android, the payment will go through, but your purchase callback will receive an error.

Adding 1 month Subscription

1. Open Google Play Developer Console, and navigate to your app.
2. Click on "In-app Products" in the menu. Then click the "Add New Product" button.
3. Select "Subscription", and enter "iapdemo.noads.month.auto" for the Product ID. Then click "Continue"

ADD NEW PRODUCT

What type of product would you like to add? *

Managed product Subscription

Subscriptions let you sell content, services, or features in your app with automated, recurring billing. [Learn more](#)

Product ID *

iapdemo.noads.month.auto
24 of 150 characters

Please note that you can NOT change the product type and product ID later and that you cannot re-use the product ID again. [Learn more](#)

Continue Cancel

Figure 328. Add new product dialog

Now fill in the form. You can choose your own price and name for the product. The following is a screenshot of the options I chose.

The screenshot shows the 'In-app Products' section of the Google Play Developer Console. A new subscription product named '3 MONTHS NO ADS' has been created. The product ID is 'iapdemo.noads.3month.auto'. The title is '3 months no ads' and the description is 'No ads for 3 months'. The price is set to CAD 2.99 and the billing period is set to '3 months'. The free trial period is 0 days. The product status is 'Active'.

Figure 329. Add product to google

Adding 3 month Subscription

Follow the same process as for the 1 month subscription except use "iapdemo.noads.3month.auto" for the product ID, and select "3 months" for the billing period instead of "Monthly".

Testing The App

At this point we should be ready to test our app. Assuming you've installed the app using the invite link you sent yourself from Google play, **as a test account that is listed on your testers list**, you should be good to go.

Open the app, click on "Subscriptions", and try to purchase a 1-month subscription. If all goes well, it should insert the subscription into your database. But with no expiry date, since we haven't yet implemented receipt validation yet. We'll do that next.

Creating Google Play Receipt Validation Credentials

Google play receipt validation is accomplished via the [android-publisher Purchases: get API](#) [<https://developers.google.com/android-publisher/api-ref/purchases/subscriptions/get>]. The CN1-IAP-Validation library shields you from most of the complexities of using this API, but you still need to obtain a

"private key" and a "client id" to access this API. Both of these are provided when you set up an [OAuth2 Service Account](https://developers.google.com/identity/protocols/OAuth2ServiceAccount) [<https://developers.google.com/identity/protocols/OAuth2ServiceAccount>] for your app.



The following steps assume that you have already created your app in Google play and have published it to at least the alpha channel. See my previous post on this topic here (Link to be provided).

Steps:

1. Open the [Google API Developer Console](https://console.developers.google.com/apis) [<https://console.developers.google.com/apis>], and select your App from the the menu.
2. Click on the "Library" menu item in the left menu, and then click the "Google Play Developer API" link.



Figure 330. Google Play Developer API Link

3. Click on the button that says "Enable". (If you already have it enabled, then just proceed to the next step).



Figure 331. Enable API button

4. Click on the "Credentials" menu item in the left menu.
5. In the "Credentials" drop-down menu, select the "Service Account Key" option.

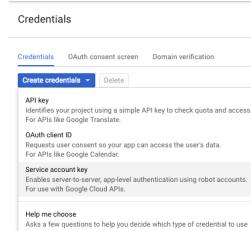


Figure 332. Credentials dropdown

6. You will be presented with a new form. In the "Service Account" drop-down, select "New Service Account". This will give you some additional options.



Figure 333. Create service account key

7. Enter anything you like for the "Service account name". For the role, we'll select "Project" >

"Owner" for now just so we don't run into permissions issues. You'll probably want to investigate further to fine a more limited role that only allows receipt verification, but for now, I don't want any unnecessary road blocks for getting this to work. We're probably going to run into "permission denied" errors at first anyways, so the fewer reasons for this, the better.

8. It will auto-generate an account ID for you.
9. Finally, for the "Key type", select "JSON". Then click the "Create" button.

This should prompt the download of a JSON file that will have contents similar to the following:

```
{  
  "type": "service_account",  
  "project_id": "iapdemo-152500",  
  "private_key_id": "1b1d39f2bc083026b164b10a444ff7d839826b8a",  
  "private_key": "-----BEGIN PRIVATE KEY----- ... some private key string -----END PRIVATE KEY-----\n",  
  "client_email": "iapdemo@iapdemo-152500.iam.gserviceaccount.com",  
  "client_id": "11760157263333082772",  
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",  
  "token_uri": "https://accounts.google.com/o/oauth2/token",  
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",  
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/iapdemo%40iapdemo-  
152500.iam.gserviceaccount.com"  
}
```

This is where we get the information we're looking for. The "client_email" is what we'll use for your `googleClientId`, and the "private_key" is what we'll use for the `googlePrivateKey`.



Use the "client_email" value as our client ID, not the "client_id" value as you might be tempted to do.

We'll set these in our constants:

```
public static final String GOOGLE_DEVELOPER_API_CLIENT_ID="iapdemo@iapdemo-152500.iam.gserviceaccount.com";  
public static final String GOOGLE_DEVELOPER_PRIVATE_KEY="-----BEGIN PRIVATE KEY----- ... -----END PRIVATE KEY-----\n";  
  
...  
  
validator.setGoogleClientId(GOOGLE_DEVELOPER_API_CLIENT_ID);  
validator.setGooglePrivateKey(GOOGLE_DEVELOPER_PRIVATE_KEY);
```

NOT DONE YET

Before we can use these credentials to verify receipts for our app, we need to link our app to this new service account from within Google Play.

Steps:

1. Open the [Google Play Developer Console](https://play.google.com/apps/publish/) [<https://play.google.com/apps/publish/>], then click on "Settings" > "API Access".
2. You should see your app listed on this page. Click the "Link" button next to your app.



Figure 334. Link to API

3. This should reveal some more options on the page. You should see a "Service Accounts" section with a list of all of the service accounts that you have created. Find the one we just created, and click the "Grant Access" button in its row.



Figure 335. Grant access

4. This will open a dialog titled "Add New User". Leave everything default, except change the "Role" to "Administrator". This provides "ALL" permissions to this account, which probably isn't a good idea for production. Later on, after everything is working, you can circle back and try to refine permissions. For the purpose of this tutorial, I just want to pull out all of the potential road blocks.

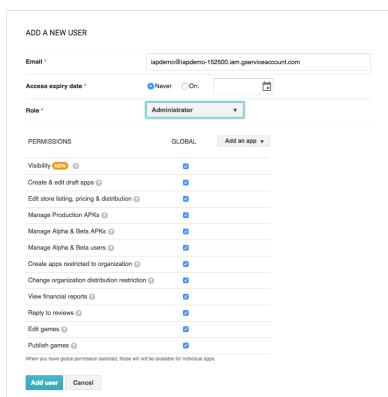


Figure 336. New User

5. Press the "Add User" button.

At this point, the service account **should** be active so we can try to validate receipts.

Testing Receipt Validation

The `ReceiptsFacadeREST` class includes a flag to enable/disable play store validation. By default it's disabled. Let's enable it:

```
public static final boolean DISABLE_PLAY_STORE_VALIDATION=true;
```

Change this to `false`.

Then build and run the server app. The `validateSubscriptionsCron()` method is set to run once per minute, so we just need to wait for the timer to come up and it should try to validate all of the play store receipts.



I'm assuming you've already added a receipt in the previous test that we did. If necessary, you should purchase the subscription again in your app.

After a minute or so, you should see "----- VALIDATING RECEIPTS -----" written in the Glassfish

log, and it will validate your receipts. If it works, your receipt's expiry date will get populated in the database, and you can press "Synchronize Receipts" in your app to see this reflected. If it fails, there will likely be a big ugly stack trace and exception readout with some clues about what went wrong.

Realistically, your first attempt will fail for some reason. Use the error codes and stack traces to help lead you to the problem. And feel free to post questions here.

9.2.19. iTunes Connect Setup

The process for setting up and testing your app on iOS is much simpler than on Android (IMHO). It took me a couple hours to get the iTunes version working, vs a couple days on the Google Play side of things. One notable difference that makes things simpler is that you don't need to actually upload your app to the store to test in-app purchase. You can just use your debug build on your device. It's also **much** easier to roll a bunch of test accounts than on Google Play. You don't need to set up an alpha program, you just create a few "test accounts" (and this is easy to do) in your iTunes connect account, and then make sure to use one of these accounts when making a purchase. You can easily switch accounts on your device from the "Settings" app, where you can just log out of the iTunes store - which will cause you to be prompted in your app the next time you make a purchase.

Setting up In-App Products

The process to add products in iTunes connect is outlined [in this apple developer document](https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnectInAppPurchase_Guide/Chapters/CreatingInAppPurchaseProducts.html#/apple_ref/doc/uid/TP40013727-CH3-SW1) [https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnectInAppPurchase_Guide/Chapters/CreatingInAppPurchaseProducts.html#/apple_ref/doc/uid/TP40013727-CH3-SW1]. We'll add our two SKUs:

1. **iapdemo.noads.month.auto** - The 1 month subscription.
2. **iapdemo.noads.3month.auto** - The 3 month subscription.

Just make sure you add them as auto-renewable subscriptions, and that you specify the appropriate renewal periods. Use the SKU as the product ID. Both of these products will be added to the same subscription group. Call the group whatever you like.

Creating Test Accounts

In order to test purchases, you need to create some test accounts. See [this apple document](https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/SettingUpUserAccounts.html#/apple_ref/doc/uid/TP40011225-CH25-SW10) [https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/SettingUpUserAccounts.html#/apple_ref/doc/uid/TP40011225-CH25-SW10] for details on how to create these test accounts. Don't worry, the process is much simpler than for Android. It should take you under 5 minutes.

Once you have the test accounts created, you should be set to test the app.

1. Make sure your server is running.
2. Log out from the app store. The process is described [here](https://support.apple.com/en-ca/HT203983) [<https://support.apple.com/en-ca/HT203983>].
3. Open your app.
4. Try to purchase a 1-month subscription

If all went well, you should see the receipt listed in the RECEIPTS table of your database. But the expiry date will be null. We need to set up receipt verification in order for this to work.

Setting up Receipt Verification

In order for receipt verification to work we simply need to generate a shared secret in iTunes connect. The process is described [here](https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnectInAppPurchase_Guide/Chapters/CreatingInAppPurchaseProducts.html) [https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnectInAppPurchase_Guide/Chapters/CreatingInAppPurchaseProducts.html].

Once you have a shared secret, update the ReceiptsFacadeREST class with the value:

```
public static final String APPLE_SECRET = "your-shared-secret-here";
```

And enable iTunes store validation:

```
public static final boolean DISABLE_ITUNES_STORE_VALIDATION=true;
```

Change this to **false**.

If you rebuild and run the server project, and wait for the `validateSubscriptionsCron()` method to run, it should validate the receipt. After about a minute (or less), you'll see the text "----- VALIDATING RECEIPTS -----" written to the Glassfish log file, followed by some output from connecting to the iTunes validation service. If all went well, you should see your receipt expiration date updated in the database. If not, you'll likely see some exception stack traces in the Glassfish log.

 Sandbox receipts in the iTunes store are set to run on an accelerated schedule. A 1 month subscription is actually 5 minutes, 3 months is 15 minutes etc... Also sandbox subscriptions don't seem to persist in perpetuity until the user has cancelled it. I have found that they usually renew only 4 or 5 times before they are allowed to lapse by Apple.

10. Graphics, Drawing, Images & Fonts



Drawing is considered a low level API that might introduce some platform fragmentation.

10.1. Basics - Where & How Do I Draw Manually?

The [Graphics](https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html>] class is responsible for drawing basics, shapes, images and text, it is never instantiated by the developer and is always passed on by the Codename One API.

You can gain access to a [Graphics](#) using one of the following methods:

- **Derive Component** [<https://www.codenameone.com/javadoc/com/codename1/ui/Component.html>] or a subclass of Component - within [Component](#) there are several methods that allow developers to modify the drawing behavior. These can be overridden to change the way the component is drawn:
 - [paint\(Graphics\)](#) - invoked to draw the component, this can be overridden to draw the component from scratch.
 - [paintBackground\(Graphics\)](#) / [paintBackgrounds\(Graphics\)](#) - these allow overriding the way the component background is painted although you would probably be better off implementing a painter (see below).
 - [paintBorder\(Graphics\)](#) - allows overriding the process of drawing a border, notice that border drawing might differ based on the style of the component.
 - [paintComponent\(Graphics\)](#) - allows painting only the components contents while leaving the default paint behavior to the style.
 - [paintScrollbars\(Graphics\)](#), [paintScrollbarX\(Graphics\)](#), [paintScrollbarY\(Graphics\)](#) allows overriding the behavior of scrollbar painting.
- **Implement the painter interface**, this interface can be used as a [GlassPane](#) or a background painter.

The painter interface is a simple interface that includes 1 paint method, this is a useful way to allow developers to perform custom painting without subclassing [Component](#). Painters can be chained together to create elaborate paint behavior by using the [PainterChain](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/painter/PainterChain.html>] class.

- **Glass pane** - a glass pane allows developers to paint on top of the form painting. This allows an overlay effect on top of a form.

For a novice it might seem that a glass pane is similar to overriding the Form's paint method and drawing after [super.paint\(g\)](#) completed. This isn't the case. When a component repaints (by invoking the [repaint\(\)](#) method) only that component is drawn and [Form](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Form.html>]'s [paint\(\)](#) method wouldn't be invoked. However, the glass pane painter is invoked for such cases and would work exactly as expected.

[Container](https://www.codenameone.com/javadoc/com/codename1/ui/Container.html) [https://www.codenameone.com/javadoc/com/codename1/ui/Container.html] has a glass pane method called `paintGlass(Graphics)`, which can be overridden to provide a similar effect on a `Container` level. This is especially useful for complex containers such as [Table](https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html) [https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html] which draws its lines using such a methodology.

- **Background painter** - the background painter is installed via the style, by default Codename installs a custom background painter of its own. Installing a custom painter allows a developer to completely define how the background of the component is drawn. Notice that a lot of the background style behaviors can be achieved using styles alone.

A common mistake developers make is overriding the `paint(Graphics)` method of [Form](#). The problem with that is that form has child components which might request a repaint. To prevent that either place a paintable component in the center of the [Form](#), override the `glasspane` or the background painter.

A paint method can be implemented as such:

```
// hide the title
Form hi = new Form("", new BorderLayout());
hi.add(BorderLayout.CENTER, new Component() {
    @Override
    public void paint(Graphics g) {
        // red color
        g.setColor(0xff0000);

        // paint the screen in red
        g.fillRect(getX(), getY(), getWidth(), getHeight());

        // draw hi world in white text at the top left corner of the screen
        g.setColor(0xffffffff);
        g.drawString("Hi World", getX(), getY());
    }
});
hi.show();
```



Figure 337. Hi world demo code, notice that the blue bar on top is the iOS7+ status bar

10.2. Glass Pane

The **GlassPane** 'in Codename One is inspired by the Swing 'GlassPane & LayeredPane with quite a few twists. We tried to imagine how Swing developers would have implemented the glass pane knowing what they do now about painters and Swings learning curve. But first: what is the glass pane?

A typical Codename One application is essentially composed of 3 layers (this is a gross simplification though), the background painters are responsible for drawing the background of all components including the main form. The component draws its own content (which might overrule the painter) and the glass pane paints last...



Figure 338. Form layout graphic

Essentially the glass pane is a painter that allows us to draw an overlay on top of the Codename One application.

Overriding the paint method of a form isn't a substitute for **glasspane** as it would appear to work initially, when you enter a **Form**. However, when modifying an element within the form only that element gets repainted not the entire **Form**!

So if we have a form with a **Button** [<https://www.codenameone.com/javadoc/com/codename1/ui/Button.html>] and text drawn on top using the Form's paint method it would get erased whenever the button gets focus.

The glass pane is called whenever a component gets painted, it only paints within the clipping region of the component hence it won't break the rest of the components on the **Form** which weren't modified.

You can set a painter on a form using code like this:

```
hi.setGlassPane(new Painter() {
    @Override
    public void paint(Graphics g, Rectangle rect) {
    }
});
```

Or you can use Java 8 lambdas to tighten the code a bit:

```
hi.setGlassPane((g, rect) -> {  
});
```

PainterChain [<https://www.codenameone.com/javadoc/com/codename1/ui/painter/PainterChain.html>] allows us

to chain several painters together to perform different logical tasks such as a validation painter coupled with a fade out painter. The sample below shows a crude validation panel that allows us to draw error icons next to components while exceeding their physical bounds as is common in many user interfaces

```
Form hi = new Form("Glass Pane", new BoxLayout(BoxLayout.Y_AXIS));
Style s = UIManager.getInstance().getComponentStyle("Label");
s.setFgColor(0xff0000);
s.setBgTransparency(0);
Image warningImage = FontImage.createMaterial(FontImage.MATERIAL_WARNING, s).toImage();
TextField tf1 = new TextField("My Field");
tf1.getAllStyles().setMarginUnit(Style.UNIT_TYPE_DIPS);
tf1.getAllStyles().setMargin(5, 5, 5, 5);
hi.add(tf1);
hi.setGlassPane((g, rect) -> {
    int x = tf1.getAbsoluteX() + tf1.getWidth();
    int y = tf1.getAbsoluteY();
    x -= warningImage.getWidth() / 2;
    y += (tf1.getHeight() / 2 - warningImage.getHeight() / 2);
    g.drawImage(warningImage, x, y);
});
hi.show();
```

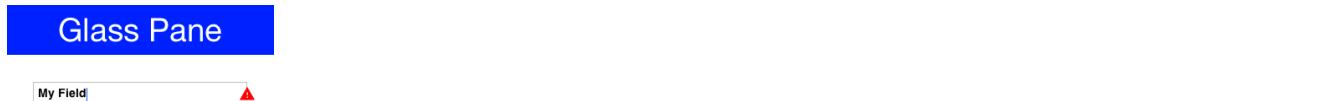


Figure 339. The glass pane draws the warning sign on the border of the component partially peeking out

10.3. Shapes & Transforms

The graphics API provides a high performance shape API that allows drawing arbitrary shapes by defining paths and curves and caching the shape drawn in the GPU.

10.4. Device Support

Shapes and transforms are available on most smartphone platforms with some caveats for the current Windows Phone port.

Notice that perspective transform is missing from the desktop/simulator port. Unfortunately there is no real equivalent to perspective transform in JavaSE that we could use.

10.5. A 2D Drawing App

We can demonstrate shape drawing with a simple example of a drawing app, that allows the user to tap the screen to draw a contour picture.

The app works by simply keeping a [GeneralPath](https://www.codenameone.com/javadoc/com/codename1/) [https://www.codenameone.com/javadoc/com/codename1/]

[ui/geom/GeneralPath.html](#)] in memory, and continually adding points as bezier curves. Whenever a point is added, the path is redrawn to the screen.

The center of the app is the **DrawingCanvas** class, which extends **Component** [[https://www.codenameone.com/javadoc/com/codename1/ui/Component.html](#)].

```
public class DrawingCanvas extends Component {  
    GeneralPath p = new GeneralPath();  
    int strokeColor = 0x0000ff;  
    int strokeWidth = 10;  
  
    public void addPoint(float x, float y){  
        // To be written  
    }  
  
    @Override  
    protected void paintBackground(Graphics g) {  
        super.paintBackground(g);  
        Stroke stroke = new Stroke(  
            strokeWidth,  
            Stroke.CAP_BUTT,  
            Stroke.JOIN_ROUND, 1f  
        );  
        g.setColor(strokeColor);  
  
        // Draw the shape  
        g.drawShape(p, stroke);  
    }  
  
    @Override  
    public void pointerPressed(int x, int y) {  
        addPoint(x-getParent().getAbsoluteX(), y-getParent().getAbsoluteY());  
    }  
}
```

Conceptually this is very basic component. We will be overriding the **paintBackground()** [[https://www.codenameone.com/javadoc/com/codename1/ui/Component.html#paintBackground\(com.codename1.ui.Graphics\)](#)] method to draw the path. We keep a reference to a **GeneralPath** [[https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html](#)] object (which is the concrete implementation of the **Shape** [[https://www.codenameone.com/javadoc/com/codename1/ui/geom/Shape.html](#)] interface in Codename One) to store each successive point in the drawing. We also parametrize the stroke width and color.

The implementation of the **paintBackground()** method (shown above) should be fairly straight forward. It creates a stroke of the appropriate width, and sets the color on the graphics context. Then it calls **drawShape()** to render the path of points.

10.5.1. Implementing addPoint()

The addPoint method is designed to allow us to add points to the drawing. A simple implementation that uses straight lines rather than curves might look like this:

```
private float lastX = -1;  
private float lastY = -1;  
  
public void addPoint(float x, float y) {  
    if (lastX == -1) {  
        // this is the first point... don't draw a line yet  
        p.moveTo(x, y);  
    } else {  
        p.lineTo(x, y);  
    }  
    lastX = x;  
    lastY = y;  
  
    repaint();  
}
```

We introduced a couple house-keeping member vars (`lastX` and `lastY`) to store the last point that was added so that we know whether this is the first tap or a subsequent tap. The first tap triggers a `moveTo()` call, whereas subsequent taps trigger `lineTo()` calls, which draw lines from the last point to the current point.

A drawing might look like this:

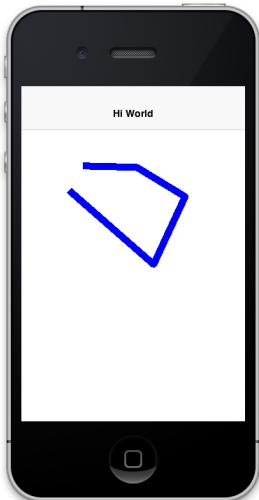


Figure 340. `lineTo` example

10.5.2. Using Bezier Curves

Our previous implementation of `addPoint()` used lines for each segment of the drawing. Let's make an adjustment to allow for smoother edges by using quadratic curves instead of lines.

Codename One's `GeneralPath` class includes two methods for drawing curves:

1. **quadTo()** [https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html#quadTo(float,%20float,%20float,%20float)] : Appends a quadratic bezier curve. It takes 2 points: a control point, and an end point.
2. **curveTo()** [https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html#curveTo(float,%20float,%20float,%20float,%20float,%20float)] : Appends a cubic bezier curve, taking 3 points: 2 control points, and an end point.

See the [General Path javadocs](#) [https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html] for the full API.

We will make use of the **quadTo()** [https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html#quadTo(float,%20float,%20float,%20float)] method to append curves to the drawing as follows:

```
private boolean odd=true;
public void addPoint(float x, float y){
    if ( lastX == -1 ){
        p.moveTo(x, y);

    } else {
        float controlX = odd ? lastX : x;
        float controlY = odd ? y : lastY;
        p.quadTo(controlX, controlY, x, y);
    }
    odd = !odd;
    lastX = x;
    lastY = y;
    repaint();
}
```

This change should be fairly straight forward except, perhaps, the business with the **odd** variable. Since quadratic curves require two points (in addition to the implied starting point), we can't simply take the last tap point and the current tap point. We need a point between them to act as a control point. This is where we get the curve from. The control point works by exerting a sort of "gravity" on the line segment, to pull the line towards it. This results in the line being curved. I use the **odd** marker to alternate the control point between positions above the line and below the line.

A drawing from the resulting app looks like:

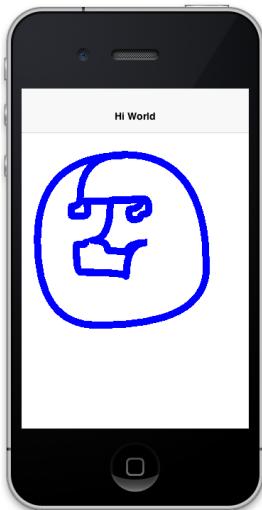


Figure 341. Result of quadTo example

10.5.3. Detecting Platform Support

The [DrawingCanvas](#) example is a bit naive in that it assumes that the device supports the shape API. If I were to run this code on a device that doesn't support the [Shape](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/geom/Shape.html>] API, it would just draw a blank canvas where I expected my shape to be drawn. You can fall back gracefully if you make use of the [Graphics.isShapeSupported\(\)](#) [[https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html#isShapeSupported\(\)](https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html#isShapeSupported())] method. E.g.

```
@Override  
protected void paintBackground(Graphics g) {  
    super.paintBackground(g);  
    if ( g.isShapeSupported() ){  
        // do my shape drawing code here  
    } else {  
        // draw an alternate representation for device  
        // that doesn't support shapes.  
        // E.g. you could defer to the Pisces  
        // library in this case  
    }  
}
```

10.6. Transforms

The [Graphics](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html>] class has included limited support for 2D transformations for some time now including scaling, rotation, and translation:

- [scale\(x, y\)](#) : Scales drawing operations by a factor in each direction.
- [translate\(x, y\)](#) : Translates drawing operations by an offset in each direction.
- [rotate\(angle\)](#) : Rotates about the origin.
- [rotate\(angle, px, py\)](#) : Rotates about a pivot point.



`scale()` and `rotate()` methods are only available on platforms that support Affine transforms. See table X for a compatibility list.

10.6.1. Device Support

As of this writing, not all devices support transforms (i.e. `scale()` and `rotate()`). The following is a list of platforms and their respective levels of support.

Table 7. Transforms Device Support

Platform	Affine Supported
Simulator	Yes
iOS	Yes
Android	Yes
JavaScript	Yes
J2ME	No
BlackBerry (4.2 & 5)	No
Windows Phone	No (pending)

You can check if a particular `Graphics` [<https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html>] context supports rotation and scaling using the `isAffineSupported()` method.

e.g.

```
public void paint(Graphics g){  
    if ( g.isAffineSupported() ){  
        // Do something that requires rotation and scaling  
  
    } else {  
        // Fallback behavior here  
    }  
}
```

10.7. Example: Drawing an Analog Clock

In the following sections, I will implement an analog clock component. This will demonstrate three key concepts in Codename One's graphics:

1. Using the `GeneralPath` class for drawing arbitrary shapes.
2. Using `Graphics.translate()` to translate our drawing position by an offset.
3. Using `Graphics.rotate()` to rotate our drawing position.

There are three separate things that need to be drawn in a clock:

1. **The tick marks.** E.g. most clocks will have a tick mark for each second, larger tick marks for

each hour, and sometimes even larger tick marks for each quarter hour.

2. **The numbers.** We will draw the clock numbers (1 through 12) in the appropriate positions.
3. **The hands.** We will draw the clock hands to point at the appropriate points to display the current time.

10.7.1. The AnalogClock Component

Our clock will extend the **Component** [<https://www.codenameone.com/javadoc/com/codename1/ui/Component.html>] class, and override the **paintBackground()** method to draw the clock as follows:

```
public class AnalogClock extends Component {  
    Date currentTime = new Date();  
  
    @Override  
    public void paintBackground(Graphics g) {  
        // Draw the clock in this method  
    }  
}
```

10.7.2. Setting up the Parameters

Before we actually draw anything, let's take a moment to figure out what values we need to know in order to draw an effective clock. Minimally, we need two values:

1. The center point of the clock.
2. The radius of the clock.

In addition, I am adding the following parameters to help customize how the clock is rendered:

1. **The padding** (i.e. the space between the edge of the component and the edge of the clock circle).
2. **The tick lengths.** I will be using 3 different lengths of tick marks on this clock. The longest ticks will be displayed at quarter points (i.e. 12, 3, 6, and 9). Slightly shorter ticks will be displayed at the five-minute marks (i.e. where the numbers appear), and the remaining marks (corresponding with seconds) will be quite short.

```
// Hard code the padding at 10 pixels for now
double padding = 10;

// Clock radius
double r = Math.min(getWidth(), getHeight())/2-padding;

// Center point.
double cX = getX()+getWidth()/2;
double cY = getY()+getHeight()/2;

//Tick Styles
int tickLen = 10; // short tick
int medTickLen = 30; // at 5-minute intervals
int longTickLen = 50; // at the quarters
int tickColor = 0xCCCCCC;
Stroke tickStroke = new Stroke(2f, Stroke.CAP_BUTT, Stroke.JOIN_ROUND, 1f);
```

10.7.3. Drawing the Tick Marks

For the tick marks, we will use a single [GeneralPath](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html>] object, making use of the `moveTo()` and `lineTo()` methods to draw each individual tick.

```

// Draw a tick for each "second" (1 through 60)
for ( int i=1; i<= 60; i++){
    // default tick length is short
    int len = tickLen;
    if ( i % 15 == 0 ){
        // Longest tick on quarters (every 15 ticks)
        len = longTickLen;
    } else if ( i % 5 == 0 ){
        // Medium ticks on the '5's (every 5 ticks)
        len = medTickLen;
    }

    double di = (double)i; // tick num as double for easier math

    // Get the angle from 12 O'Clock to this tick (radians)
    double angleFrom12 = di/60.0*2.0*Math.PI;

    // Get the angle from 3 O'Clock to this tick
    // Note: 3 O'Clock corresponds with zero angle in unit circle
    // Makes it easier to do the math.
    double angleFrom3 = Math.PI/2.0-angleFrom12;

    // Move to the outer edge of the circle at correct position
    // for this tick.
    ticksPath.moveTo(
        (float)(cX+Math.cos(angleFrom3)*r),
        (float)(cY-Math.sin(angleFrom3)*r)
    );

    // Draw line inward along radius for length of tick mark
    ticksPath.lineTo(
        (float)(cX+Math.cos(angleFrom3)*(r-len)),
        (float)(cY-Math.sin(angleFrom3)*(r-len))
    );
}

// Draw the full shape onto the graphics context.
g.setColor(tickColor);
g.drawShape(ticksPath, tickStroke);

```



This example uses a little bit of trigonometry to calculate the (x, y) coordinates of the tick marks based on the angle and the radius. If math isn't your thing, don't worry. This example just makes use of the identities: $x=r\cos\theta$ and $y=r\sin\theta$.

At this point our clock should include a series of tick marks orbiting a blank center as shown below:

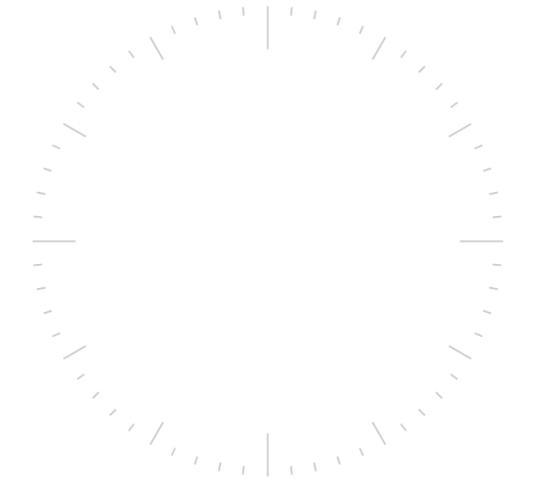


Figure 342. Drawing tick marks on the watch face

10.7.4. Drawing the Numbers

The `Graphics.drawString(str, x, y)` method allows you to draw text at any point of a component. The tricky part here is calculating the correct `x` and `y` values for each string so that the number appears in the correct location.

For the purposes of this tutorial, we will use the following strategy. For each number (1 through 12):

1. Use the `Graphics.translate(x,y)` method to apply a translation from the clock's center point to the point where the number should appear.
2. Draw number (using `drawString()`) at the clock's center. It should be rendered at the correct point due to our translation.
3. Invert the translation performed in step 1.

```

for ( int i=1; i<=12; i++){
    // Calculate the string width and height so we can center it properly
    String numStr = ""+i;
    int charWidth = g.getFont().stringWidth(numStr);
    int charHeight = g.getFont().getHeight();

    double di = (double)i; // number as double for easier math

    // Calculate the position along the edge of the clock where the number should
    // be drawn
    // Get the angle from 12 O'Clock to this tick (radians)
    double angleFrom12 = di/12.0*2.0*Math.PI;

    // Get the angle from 3 O'Clock to this tick
    // Note: 3 O'Clock corresponds with zero angle in unit circle
    // Makes it easier to do the math.
    double angleFrom3 = Math.PI/2.0-angleFrom12;

    // Get diff between number position and clock center
    int tx = (int)(Math.cos(angleFrom3)*(r-longTickLen));
    int ty = (int)(-Math.sin(angleFrom3)*(r-longTickLen));

    // For 6 and 12 we will shift number slightly so they are more even
    if ( i == 6 ){
        ty -= charHeight/2;
    } else if ( i == 12 ){
        ty += charHeight/2;
    }

    // Translate the graphics context by delta between clock center and
    // number position
    g.translate(
        tx,
        ty
    );

    // Draw number at clock center.
    g.drawString(numStr, (int)cX-charWidth/2, (int)cY-charHeight/2);

    // Undo translation
    g.translate(-tx, -ty);

}

```

 This example is, admittedly, a little contrived to allow for a demonstration of the `Graphics.translate()` method. We could have just as easily passed the exact location of the number to `drawString()` rather than draw at the clock center and translate to the correct location.

Now, we should have a clock with tick marks *and* numbers as shown below:

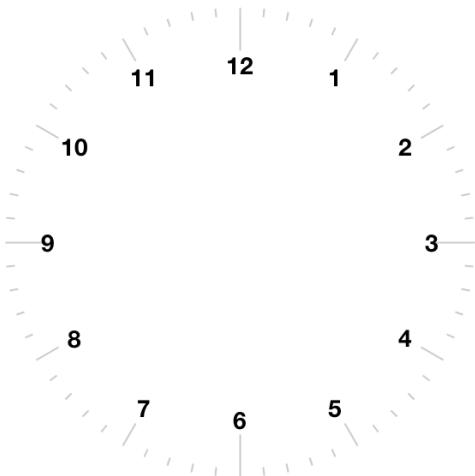


Figure 343. Drawing the numbers on the watch face

10.7.5. Drawing the Hands

The clock will include three hands: Hour, Minute, and Second. We will use a separate [GeneralPath](https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html) [https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html] object for each hand. For the positioning/angle of each, I will employ the following strategy:

1. Draw the hand at the clock center pointing toward 12 (straight up).
2. Translate the hand slightly down so that it overlaps the center.
3. Rotate the hand at the appropriate angle for the current time, using the clock center as a pivot point.

Drawing the Second Hand:

For the "second" hand, we will just use a simple line from the clock center to the inside edge of the medium tick mark at the 12 o'clock position.

```
GeneralPath secondHand = new GeneralPath();
secondHand.moveTo((float)cX, (float)cY);
secondHand.lineTo((float)cX, (float)(cY-(r-medTickLen)));
```

And we will translate it down slightly so that it overlaps the center. This translation will be performed on the [GeneralPath](#) object directly rather than through the [Graphics](#) context:

```
Shape translatedSecondHand = secondHand.createTransformedShape(
    Transform.makeTranslation(0f, 5)
);
```

Rotating the Second Hand::

The rotation of the second hand will be performed in the [Graphics](#) context via the [rotate\(angle, px, py\)](#) method. This requires us to calculate the angle. The [px](#) and [py](#) arguments constitute the pivot

point of the rotation, which, in our case will be the clock center.



The rotation pivot point is expected to be in absolute screen coordinates rather than relative coordinates of the component. Therefore we need to get the absolute clock center position in order to perform the rotation.

```
// Calculate the angle of the second hand
Calendar calendar = Calendar.getInstance(TimeZone.getDefault());
double second = (double)(calendar.get(Calendar.SECOND));
double secondAngle = second/60.0*2.0*Math.PI;

// Get absolute center position of the clock
double absCX = getAbsoluteX()+cX-getX();
double absCY = getAbsoluteY()+cY-getY();

g.rotate((float)secondAngle, (int)absCX, (int)absCY);
g.setColor(0xff0000);
g.drawShape(
    translatedSecondHand,
    new Stroke(2f, Stroke.CAP_BUTT, Stroke.JOIN_BEVEL, 1f)
);
g.resetAffine();
```



Remember to call `resetAffine()` after you're done with the rotation, or you will see some unexpected results on your form.

Drawing the Minute And Hour Hands:

The mechanism for drawing the hour and minute hands is largely the same as for the minute hand. There are a couple of added complexities though:

1. We'll make these hands trapezoidal, and almost triangular rather than just using a simple line. Therefore the `GeneralPath` construction will be slightly more complex.
2. Calculation of the angles will be slightly more complex because they need to take into account multiple parameters. E.g. The hour hand angle is informed by both the hour of the day and the minute of the hour.

The remaining drawing code is as follows:

```

// Draw the minute hand
GeneralPath minuteHand = new GeneralPath();
minuteHand.moveTo((float)cX, (float)cY);
minuteHand.lineTo((float)cX+6, (float)cY);
minuteHand.lineTo((float)cX+2, (float)(cY-(r-tickLen)));
minuteHand.lineTo((float)cX-2, (float)(cY-(r-tickLen)));
minuteHand.lineTo((float)cX-6, (float)cY);
minuteHand.closePath();

// Translate the minute hand slightly down so it overlaps the center
Shape translatedMinuteHand = minuteHand.createTransformedShape(
    Transform.makeTranslation(0f, 5)
);

double minute = (double)(calendar.get(Calendar.MINUTE)) +
    (double)(calendar.get(Calendar.SECOND))/60.0;

double minuteAngle = minute/60.0*2.0*Math.PI;

// Rotate and draw the minute hand
g.rotate((float)minuteAngle, (int)absCX, (int)absCY);
g.setColor(0x000000);
g.fillShape(translatedMinuteHand);
g.resetAffine();


// Draw the hour hand
GeneralPath hourHand = new GeneralPath();
hourHand.moveTo((float)cX, (float)cY);
hourHand.lineTo((float)cX+4, (float)cY);
hourHand.lineTo((float)cX+1, (float)(cY-(r-longTickLen)*0.75));
hourHand.lineTo((float)cX-1, (float)(cY-(r-longTickLen)*0.75));
hourHand.lineTo((float)cX-4, (float)cY);
hourHand.closePath();

Shape translatedHourHand = hourHand.createTransformedShape(
    Transform.makeTranslation(0f, 5)
);

//Calendar cal = Calendar.getInstance().get
double hour = (double)(calendar.get(Calendar.HOUR_OF_DAY)%12) +
    (double)(calendar.get(Calendar.MINUTE))/60.0;

double angle = hour/12.0*2.0*Math.PI;
g.rotate((float)angle, (int)absCX, (int)absCY);
g.setColor(0x000000);
g.fillShape(translatedHourHand);
g.resetAffine();

```

10.7.6. The Final Result

At this point, we have a complete clock as shown below:

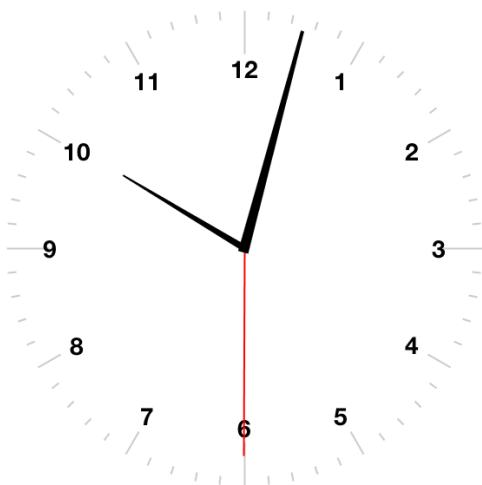


Figure 344. The final result - fully rendered watch face

10.7.7. Animating the Clock

The current clock component is cool, but it is static. It just displays the time at the point the clock was created. We discussed low level animations in the animation section of the guide, here we will show a somewhat more elaborate example.

In order to animate our clock so that it updates once per second, we only need to do two things:

1. Implement the `animate()` method to indicate when the clock needs to be updated/re-drawn.
2. Register the component with the form so that it will receive animation "pulses".

The `animate()` method in the `AnalogClock` class:

```
Date currentTime = new Date();
long lastRenderedTime = 0;

@Override
public boolean animate() {
    if ( System.currentTimeMillis()/1000 != lastRenderedTime/1000){
        currentTime.setTime(System.currentTimeMillis());
        return true;
    }
    return false;
}
```

This method will be invoked on each "pulse" of the EDT. It checks the last time the clock was rendered and returns `true` only if the clock hasn't been rendered in the current "time second" interval. Otherwise it returns false. This ensures that the clock will only be redrawn when the time changes.

10.8. Starting and Stopping the Animation

Animations can be started and stopped via the `Form.registerAnimated(component)` and `Form.deregisterAnimated(component)` methods. We chose to encapsulate these calls in `start()` and `stop()` methods in the component as follows:

```
public void start(){
    getComponentForm().registerAnimated(this);
}

public void stop(){
    getComponentForm().deregisterAnimated(this);
}
```

So the code to instantiate the clock, and start the animation would be something like:

```
AnalogClock clock = new AnalogClock();
parent.addComponent(clock);
clock.start();
```

10.9. Shape Clipping

Clipping is one of the core tenants of graphics programming, you define the boundaries for drawing and when you exceed said boundaries things aren't drawn. Shape clipping allows us to clip based on any arbitrary `Shape` and not just a rectangle, this allows some unique effects generated in runtime.

E.g. this code allows us to draw a rather complex image of duke:

```

Image duke = null;
try {
    // duke.png is just the default Codename One icon copied into place
    duke = Image.createImage("/duke.png");
} catch(IOException err) {
    Log.e(err);
}
final Image finalDuke = duke;

Form hi = new Form("Shape Clip");

// We create a 50 x 100 shape, this is arbitrary since we can scale it easily
GeneralPath path = new GeneralPath();
path.moveTo(20, 0);
path.lineTo(30, 0);
path.lineTo(30, 100);
path.lineTo(20, 100);
path.lineTo(20, 15);
path.lineTo(5, 40);
path.lineTo(5, 25);
path.lineTo(20, 0);

Stroke stroke = new Stroke(0.5f, Stroke.CAP_ROUND, Stroke.JOIN_ROUND, 4);
hi.getContentPane().getUnselectedStyle().setBgPainter((Graphics g, Rectangle rect) -> {
    g.setColor(0xff);
    float widthRatio = ((float)rect.getWidth()) / 50f;
    float heightRatio = ((float)rect.getHeight()) / 100f;
    g.scale(widthRatio, heightRatio);
    g.translate((int)((float)rect.getX() / widthRatio), (int)((float)rect.getY() / heightRatio));
    g.setClip(path);
    g.setAntiAliased(true);
    g.drawImage(finalDuke, 0, 0, 50, 100);
    g.setClip(path.getBounds());
    g.drawShape(path, stroke);
    g.translate(-(int)((float)rect.getX() / widthRatio), -(int)((float)rect.getY() / heightRatio));
    g.resetAffine();
});

hi.show();

```

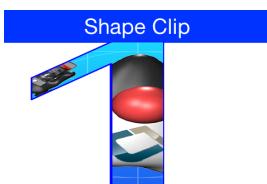


Figure 345. Shape Clipping used to clip the image of duke within the given shape



Notice that this functionality isn't available on all platforms so you normally need to test if shaped clipping is supported using [isShapeClipSupported\(\)](#) [<https://www.codenameone.com/javadoc/com/codenname1/ui/Graphics.html#isShapeClipSupported-->].

10.10. The Coordinate System

The Codename One coordinate system follows the example of Swing (and many other - but not all-graphics libraries) and places the origin in the upper left corner of the screen. X-values grow to the right, and Y-values grow downward as illustrated below:

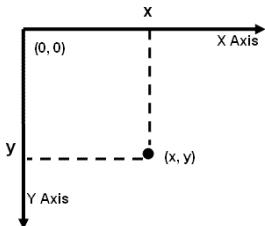


Figure 346. The Codename One graphics coordinate space

Therefore the screen origin is at the top left corner of the screen. Given this information, consider the method call on the [Graphics](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html>] context `g`:

```
g.drawRect(10,10, 100, 100);
```

Where would this rectangle be drawn on the screen?

If you answered something like "10 pixels from the top, and 10 pixels from the left of the screen", you *might* be right. It depends on whether the graphics has a translation or transform applied to it. If there is currently a translation of `(20,20)` (i.e. 20 pixels to the right, and 20 pixels down), then the rectangle would be rendered at `(30, 30)`.

You can always find out the current translation of the graphics context using the `Graphics.getTranslateX()` and `Graphics.getTranslateY()` methods:

```
// Find out the current translation
int currX = g.getTranslateX();
int currY = g.getTranslateY();

// Reset the translation to zeroes
g.translate(-currX, -currY);

// Now we are working in absolute screen coordinates
g.drawRect(10, 10, 100, 100);

// This rectangle should now be drawn at the exact screen
// coordinates (10,10).

// Restore the translation
g.translate(currX, currY);
```

i This example glosses over issues such as clipping and transforms which may cause it to not work as you expect. E.g. When painting a component inside its `paint()` method, there is a clip applied to the context so that only the content you draw within the bounds of the component will be seen.

If, in addition, there is a transform applied that rotates the context 45 degrees clockwise, then the rectangle will be drawn at a 45 degree angle with its top left corner somewhere on the left edge of the screen.

Luckily you usually don't have to worry about the exact screen coordinates for the things you paint. Most of the time, you will only be concerned with relative coordinates.

10.10.1. Relative Coordinates

Usually, when you are drawing onto a `Graphics` context, you are doing so within the context of a Component's `paint()` method (or one of its variants). In this case, you generally don't care what the exact screen coordinates are of your drawing. You are only concerned with their relative location within the coordinate. You can leave the positioning (and even sizing) of the coordinate up to Codename One. Thank you for reading.

To demonstrate this, let's create a simple component called `Rectangle` [<https://www.codenameone.com/javadoc/com/codename1/ui/geom/Rectangle.html>] component, that simply draws a rectangle on the screen. We will use the component's position and size to dictate the size of the rectangle to be drawn. And we will keep a 5 pixel padding between the edge of the component and the edge of our rectangle.

```
class RectangleComponent extends Component {  
    public void paint(Graphics g){  
        g.setColor(0x0000ff);  
        g.drawRect(getX()+5, getY()+5, getWidth()-10, getHeight()-10);  
    }  
}
```

The result is as follows:



Figure 347. The rectangle component

i The `x` and `y` coordinates that are passed to the `drawRect(x,y,w,h)` method are relative to the component's *parent's* origin—not the component itself .. its **parent**. This is why we the `x` position is `getX()+5` and not just `5`.

10.10.2. Transforms and Rotations

Unlike the `Graphics drawXXX` primitives, methods for setting transformations, including `scale(x,y)` and `rotate(angle)`, are always applied in terms of screen coordinates. This can be confusing at first, because you may be unsure whether to provide a relative coordinate or an absolute coordinate for a given method.

The general rule is:

1. All coordinates passed to the `drawXXX()` and `fillXXX()` methods will be subject to the graphics context's transform and translation settings.
2. All coordinates passed to the context's transformation settings are considered to be screen coordinates, and are not subject to current transform and translation settings.

Let's take our `RectangleComponent` as an example. Suppose we want to rotate the rectangle by 45 degrees, our first attempt might look something like:

```
class RectangleComponent extends Component {  
  
    @Override  
    protected Dimension calcPreferredSize() {  
        return new Dimension(250,250);  
    }  
  
    public void paint(Graphics g) {  
        g.setColor(0x0000ff);  
        g.rotate((float) (Math.PI / 4.0));  
        g.drawRect(getX() + 5, getY() + 5, getWidth() - 10, getHeight() - 10);  
        g.rotate(-(float) (Math.PI / 4.0));  
    }  
}
```



When performing rotations and transformations inside a `paint()` method, always remember to revert your transformations at the end of the method so that it doesn't pollute the rendering pipeline for subsequent components.

The behavior of this rotation will vary based on where the component is rendered on the screen. To demonstrate this, let's try to place five of these components on a form inside a `BorderLayout` [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html>] and see how it looks:

```

class MyForm extends Form {

    public MyForm() {
        super("Rectangle Rotations");
        for ( int i=0; i< 10; i++ ){
            this.addComponent(new RectangleComponent());
        }
    }
}

```

The result is as follows:

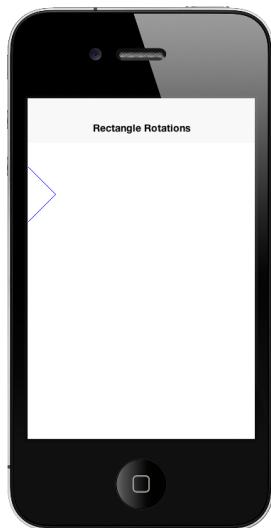


Figure 348. Rotating the rectangle

This may not be an intuitive outcome since we drew 10 rectangle components, but we only see a portion of one rectangle. The reason is that the `rotate(angle)` method uses the screen origin as the pivot point for the rotation. Components nearer to this pivot point will experience a less dramatic effect than components farther from it. In our case, the rotation has caused all rectangles except the first one to be rotated outside the bounds of their containing component - so they are being clipped. A more sensible solution for our component would be to place the rotation pivot point somewhere inside the component. That way all of the components would look the same. Some possibilities would be:

Top Left Corner:

```

public void paint(Graphics g) {
    g.setColor(0x0000ff);
    g.rotate((float)(Math.PI/4.0), getAbsoluteX(), getAbsoluteY());
    g.drawRect(getX() + 5, getY() + 5, getWidth() - 10, getHeight() - 10);
    g.rotate(-(float) (Math.PI / 4.0), getAbsoluteX(), getAbsoluteY());
}

```



Figure 349. Rotating the rectangle with wrong pivot point

Center:

```
public void paint(Graphics g) {  
    g.setColor(0x0000ff);  
    g.rotate(  
        (float)(Math.PI/4.0),  
        getAbsoluteX() + getWidth()/2,  
        getAbsoluteY() + getHeight()/2  
    );  
    g.drawRect(getX() + 5, getY() + 5, getWidth() - 10, getHeight() - 10);  
    g.rotate(  
        -(float)(Math.PI/4.0),  
        getAbsoluteX() + getWidth()/2,  
        getAbsoluteY() + getHeight()/2  
    );  
}
```

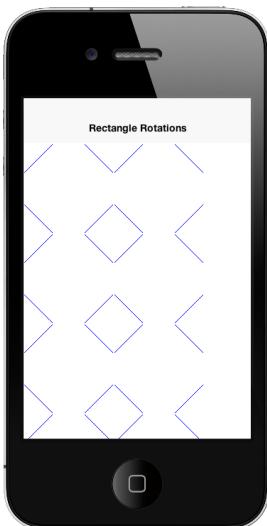


Figure 350. Rotating the rectangle with the center pivot point

You could also use the `Graphics.setTransform()` class to apply rotations and other complex transformations (including 3D perspective transforms), but I'll leave that for its own topic as it is a

little bit more complex.

10.10.3. Event Coordinates

The coordinate system and event handling are closely tied. You can listen for touch events on a component by overriding the `pointerPressed(x,y)` method. The coordinates received in this method will be **absolute screen coordinates**, so you may need to do some conversions on these coordinates before using them in your `drawXXX()` methods.

E.g. a `pointerPressed()` callback method can look like this:

```
public void pointerPressed(int x, int y) {  
    addPoint(x-getParent().getAbsoluteX(), y-getParent().getAbsoluteY());  
}
```

In this case we translated these points so that they would be relative to the origin of the parent component. This is because the `drawXXX()` methods for this component take coordinates relative to the parent component.

10.11. Images

Codename One has quite a few image types: loaded, RGB (builtin), RGB (Codename One), Mutable, EncodedImage, SVG, MultiImage, FontImage & Timeline. There are also URLImage, FileEncodedImage, FileEncodedImageAsync, `StorageEncodedImage`/Async that will be covered in the IO section.

All image types are mostly seamless to use and will just work with `drawImage` and various image related image API's for the most part with caveats on performance etc.



For animation images the code must invoke the `animate()` method on the image (this is done automatically by Codename One when placing the image as a background or as an icon!).

You only need to do it if you invoke `drawImage` in code rather than use a builtin component).

Performance and memory wise you should read the section below carefully and be aware of the image types you use. The Codename One designer tries to conserve memory and be "clever" by using only `EncodedImage`. While these are great for low memory you need to understand the complexities of image locking and be aware that you might pay a penalty if you don't.

Here are the pros/cons and logic behind every image type. This covers the logic of how it's created:

10.11.1. Loaded Image

This is the basic image you get when loading an image from the jar or network using `Image.createImage(String)` [\[https://www.codenameone.com/javadoc/com/codenname1/ui/Image.html#createImage-java.lang.String-\]](https://www.codenameone.com/javadoc/com/codenname1/ui/Image.html#createImage-java.lang.String-), `Image.createImage(InputStream)` [\[https://www.codenameone.com/javadoc/com/codename1/ui/Image.html#createImage-java.io.InputStream-\]](https://www.codenameone.com/javadoc/com/codename1/ui/Image.html#createImage-java.io.InputStream-) & `Image.createImage(byte array,int,int)`

[<https://www.codenameone.com/javadoc/com/codename1/ui/Image.html#createImage-byte:A-int-int->], ...



Some other API's might return this image type but those API's do so explicitly!

In some platforms calling `getGraphics()` on an image like this will throw an exception as it's immutable). This is true for almost all other images as well.

This restriction might not apply for all platforms.

The image is stored in RAM based on device logic and should be reasonably efficient in terms of drawing speed. However, it usually takes up a lot of RAM.

To calculate the amount of RAM taken by a loaded image we use the following formula:

$$\text{Image Width} * \text{Image Height} * 4 = \text{Size In RAM in Bytes}$$

E.g. a 50x100 image will take up 20,000 bytes of RAM.

The logic behind this is simple, every pixel contains 3 color channels and an alpha component hence 3 bytes for color and one for alpha.



This isn't the case for all images but it's very common and we prefer calculating for the worst case scenario. Even with JPEG's that don't include an alpha channel some OS's might require that additional byte.

10.11.2. The RGB Image's

There are two types of RGB constructed images that are very different from one another but since they are both technically "RGB image's" we are bundling them under the same subsection.

Internal

This is a close cousin of the loaded image. This image is created using the method `Image.createImage(int array, int, int)` [<https://www.codenameone.com/javadoc/com/codename1/ui/Image.html#createImage-int:A-int-int->] and receives the AARRGGBB data to form the image. It's more efficient than the Codename One RGB image but can't be modified, at least not on the pixel level.

The goal of this image type is to provide an easy way to render RGB data that isn't modified efficiently at platform native speeds. It's technically a [standard "Loaded Image"](#) internally.

RGBImage class

`RGBImage` [<https://www.codenameone.com/javadoc/com/codename1/ui/RGBImage.html>] is effectively an AARRGGBB array that can be drawn by Codename One.

On most platforms this is quite inefficient but for some pixel level manipulations there is just no other way.

An `RGBImage` is constructed with an `int` array (`int[]`) that includes `width*height` elements. You can

then modify the colors and alpha channel directly within the array and draw the image to any source using standard image drawing API's.



This is very inefficient in terms of rendering speed and memory overhead. Only use this technique if there is absolutely no other way!

10.11.3. EncodedImage

EncodedImage [<https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html>] is the workhorse of Codename One. Images returned from resource files are **EncodedImage** and many API's expect it.

The **EncodedImage** is effectively a **loaded image** that is "hidden" and extracted as needed to remove the memory overhead associated with loaded image. When creating an **EncodedImage** only the PNG (or JPEG etc.) is loaded to an array in RAM. Normally such images are very small (relatively) so they can be kept in memory without much overhead.

When image information is needed (pixels) the image is decoded into RAM and kept in a weak/soft reference. This allows the image to be cached for performance and allows the garbage collector to reclaim it when the memory becomes scarce.

Since the fully decoded image can be pretty big (**width X height X 4**) the ability to store just the encoded image can be pretty stark. E.g. taking our example above a 50x100 image will take up 20,000 bytes of RAM for a **loaded image** but an **EncodedImage** can reduce that to 1kb-2kb of RAM.



An **EncodedImage** might be more expensive than a **loaded image** as it will take up both the encoded size and the loaded size. So the cost might be slightly bigger in some cases. It's main value is its ability to shrink.

When drawing an **EncodedImage** it checks the weak reference cache and if the image is cached then it is shown otherwise the image is loaded the encoded image cache it then drawn.

EncodedImage is not final and can be derived to produce complex image fetching strategies e.g. the **URLImage** [<https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html>] class that can dynamically download its content from the web.

EncodedImage can be instantiated via the create methods in the **EncodedImage** class. Pretty much any image can be converted into an **EncodedImage** via the **createFromImage(Image, boolean)** [[https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html#createFromImage-com.codename1.ui.Image-boolean-\]](https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html#createFromImage-com.codename1.ui.Image-boolean-) method.

EncodedImage Locking

Naturally loading the image is more expensive so we want the images that are on the current form to remain in cache (otherwise GC will thrash a lot). That's where `lock()` kicks in, when `lock()` is active we keep a hard reference to the actual native image so it won't get GC'd. This significantly improves performance!

Internally this is invoked automatically for background images, icons etc. which results in a huge performance boost. This makes sense since these images are currently showing and they will be in RAM anyway. However, if you use a complex renderer or custom drawing UI you should `lock()` your images where possible!

To verify that locking might be a problem you can launch the performance monitor tool (accessible from the simulator menu), if you get log messages that indicate that an unlocked image was drawn you might have a problem.

10.11.4. MultiImage

Multi images don't physically exist as a concept within the Codename One API so there is no way to actually create them and they are in no way distinguishable from `EncodedImage`.

The only builtin support for multi images is in the resource file loading logic where a MultiImage is decoded and only the version that matches the current DPI is physically loaded. From that point on user code can treat it like any other `EncodedImage`.

9-image borders use multi images by default to keep their appearance more refined on the different DPI's.

10.11.5. FontImage & Material Design Icons

`FontImage` [<https://www.codenameone.com/javadoc/com/codename1/ui/FontImage.html>] allows using an icon font as if it was an image. You can specify the character, color and size and then treat the `FontImage` as if its a regular image. The huge benefits are that the font image can adapt to platform conventions in terms of color and easily scale to adapt to DPI.

You can generate icon fonts using free tools on the internet such as [this](http://fontello.com/) [<http://fontello.com/>]. Icon fonts are a remarkably simple and powerful technique to create a small, modern applications.

Icon fonts can be created in 2 basic ways the first is explicitly by defining all of the elements within the font.

```
Form hi = new Form("Icon Font");
Font materialFont = FontImage.getMaterialDesignFont();
int w = Display.getInstance().getDisplayWidth();
FontImage fntImage = FontImage.createFixed("\uE161", materialFont, 0xff0000, w, w);
hi.add(fntImage);
hi.show();
```

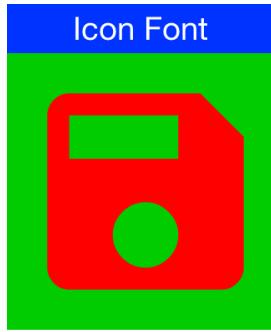


Figure 351. Icon font from material design icons created with the fixed size of display width



The samples use the builtin material design icon font. This is for convenience so the sample will work out of the box, for everyone. However you should be able to do this with any arbitrary icon font off the internet as long as its a valid TTF file.

A more common and arguably "correct" way to construct such an icon would be thru the `Style` [https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html] object. The `Style` object can provide the color, size and background information needed by `FontImage`.

There are two versions of this method, the first one expects the `Style` object to have the correct icon font set to its font attribute. The second accepts a `Font` object as an argument. The latter is useful for a case where you want to reuse the same `Style` object that you defined for a general UI element e.g. we can set an icon for a `Button` like this and it will take up the style of the `Button`:

```
Form hi = new Form("Icon Font");
Font materialFont = FontImage.getMaterialDesignFont();
int size = Display.getInstance().convertToPixels(6, true);
materialFont = materialFont.derive(size, Font.STYLE_PLAIN);
Button myButton = new Button("Save");
myButton.setIcon(FontImage.create("\uE161", myButton.getUnselectedStyle(), materialFont));
hi.add(myButton);
hi.show();
```



Figure 352. An image created from the `Style` object



Notice that for this specific version of the method the size of the font is used to determine the icon size. In the other methods for `FontImage` creation the size of the font is ignored!

Material Design Icons

There are many icon fonts in the web, the field is rather volatile and constantly changing. However, we wanted to have builtin icons that would allow us to create better looking demos and builtin components.

That's why we picked the material design icon font for inclusion in the Codename One distribution. It features a relatively stable core set of icons, that aren't IP encumbered.

You can use the builtin font directly as demonstrated above but there are far better ways to create a material design icon. To find the icon you want you can check out the [material design icon gallery](https://design.google.com/icons/) [<https://design.google.com/icons/>]. E.g. we used the save icon in the samples above.

To recreate the save icon from above we can do something like:

```
Form hi = new Form("Icon Font");
Button myButton = new Button("Save");
myButton.setIcon(FontImage.createMaterial(FontImage.MATERIAL_SAVE, myButton.getUnselectedStyle()));
hi.add(myButton);
```



Figure 353. Material save icon



Notice that the icon is smaller now as it's calculated based on the font size of the **Button** UIID.

We can even write the code in a more terse style using:

```
Form hi = new Form("Icon Font");
Button myButton = new Button("Save");
FontImage.setMaterialIcon(myButton, FontImage.MATERIAL_SAVE);
hi.add(myButton);
```

This will produce the same result for slightly shorter syntax.



FontImage can conflict with some complex API's that expect a "real" image underneath. Some odd issues can often be resolved by using the **toImage()** or **toEncodedImage()** methods to convert the scaled **FontImage** to a **loaded image**.

10.11.6. Timeline

Timeline's allow rudimentary animation and enable GIF importing using the Codename One Designer. Effectively a timeline is a set of images that can be moved rotated, scaled & blended to provide interesting animation effects. It can be created manually using the [Timeline](https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html>] class.

10.11.7. Image Masking

Image masking allows us to manipulate images by changing the opacity of an image according to a mask image. The mask image can be hardcoded or generated dynamically, it is then converted to a Mask object that can be applied to any image. Notice that the masking process is computationally intensive, it should be done once and cached/saved.

The code below can convert an image to a rounded image:

```
Toolbar.setGlobalToolbar(true);
Form hi = new Form("Rounder", new BorderLayout());
Label picture = new Label("", "Container");
hi.add(BorderLayout.CENTER, picture);
hi.getUnselectedStyle().setBgColor(0xff0000);
hi.getUnselectedStyle().setBgTransparency(255);
Style s = UIManager.getInstance().getComponentStyle("TitleCommand");
Image camera = FontImage.createMaterial(FontImage.MATERIAL_CAMERA, s);
hi.getToolbar().addCommandToRightBar("", camera, (ev) -> {
    try {
        int width = Display.getInstance().getDisplayWidth();
        Image capturedImage = Image.createImage(Capture.capturePhoto(width, -1));
        Image roundMask = Image.createImage(width, capturedImage.getHeight(), 0xff000000);
        Graphics gr = roundMask.getGraphics();
        gr.setColor(0xffffffff);
        gr.fillArc(0, 0, width, width, 0, 360);
        Object mask = roundMask.createMask();
        capturedImage = capturedImage.applyMask(mask);
        picture.setIcon(capturedImage);
        hi.revalidate();
    } catch(IOException err) {
        Log.e(err);
    }
});
});
```



Figure 354. Picture after the capture was complete and the resulted image was rounded. The background was set to red so the rounding effect will be more noticeable

Notice that this example is simplistic in order to be self contained. We often recommend that developers ship "ready made" mask images with their application which can allow very complex effects on the images.

10.11.8. URLImage

[URLImage](https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html) [https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html] is an image created with a URL, it implicitly downloads and adapts the image in the given URL while caching it locally. The typical adapt process scales the image or crops it to fit into the same size which is a hard restriction because of the way [URLImage](#) is implemented.

How Does URLImage Work?

The reason for the size restriction lies in the implementation of `URLImage`. `URLImage` is physically an animated image and so the UI thread tries to invoke its `animate()` method to refresh. The `URLImage` uses that call to check if the image was fetched and if not fetches it asynchronously.

Once the image was fetched the `animate()` method returns true to refresh the UI. During the loading process the placeholder is shown, the reason for the restriction in size is that image animations can't "grow" the image. They are assumed to be fixed so the placeholder must match the dimensions of the resulting image.

The simple use case is pretty trivial:

```
Image i = URLImage.createToStorage(placeholder, "fileNameInStorage", "http://xxx/myurl.jpg", URLImage.RESIZE_SCALE);
```

Alternatively you can use the similar `URLImage.createToFileSystem` method instead of the `Storage` [https://www.codenameone.com/javadoc/com/codename1/io/Storage.html] version.

This image can now be used anywhere a regular image will appear, it will initially show the placeholder image and then seamlessly replace it with the file after it was downloaded and stored. Notice that if you make changes to the image itself (e.g. the `scaled` method) it will generate a new image which won't be able to fetch the actual image.



Since `ImageIO` [https://www.codenameone.com/javadoc/com/codename1/ui/util/ImageIO.html] is used to perform the operations of the adapter interface its required that `ImageIO` will work. It is currently working in JavaSE, Android, iOS & Windows Phone. It doesn't work on J2ME/Blackberry devices so if you pass an adapter instance on those platforms it will probably fail to perform its task.

If the file in the URL contains an image that is too big it will scale it to match the size of the placeholder precisely!

There is also an option to fail if the sizes don't match. Notice that the image that will be saved is the scaled image, this means you will have very little overhead in downloading images that are the wrong size although you will get some artifacts.

The last argument is really quite powerful, its an interface called `URLImage.ImageAdapter` [https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.ImageAdapter.html] and you can implement it to adapt the downloaded image in any way you like. E.g. you can use an image mask to automatically create a rounded version of the downloaded image.

To do this you can just override:

```
public EncodedImage adaptImage(EncodedImage downloadedImage, Image placeholderImage)
```

In the adapter interface and just return the processed encoded image. If you do heavy processing

(e.g. rounded edge images) you would need to convert the processed image back to an encoded image so it can be saved. You would then also want to indicate that this operation should run asynchronously via the appropriate method in the class.

If you need to download the file instantly and not wait for the image to appear before download initiates you can explicitly invoke the `fetch()` method which will asynchronously fetch the image from the network. Notice that the downloading will still take time so the placeholder is still required.

Mask Adapter

A `URLImage` can be created with a mask adapter to apply an effect to an image. This allows us to round downloaded images or apply any sort of masking e.g. we can adapt the round mask code above as such:

```
Image roundMask = Image.createImage(placeholder.getWidth(), placeholder.getHeight(), 0xff000000);
Graphics gr = roundMask.getGraphics();
gr.setColor(0xffffffff);
gr.fillArc(0, 0, placeholder.getWidth(), placeholder.getHeight(), 0, 360);

URLImage.ImageAdapter ada = URLImage.createMaskAdapter(roundMask);
Image i = URLImage.createToStorage(placeholder, "fileNameInStorage", "http://xxx/myurl.jpg", ada);
```

URLImage In Lists

The biggest problem with image download service is with lists. We decided to attack this issue at the core by integrating `URLImage` [<https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html>] support directly into `GenericListCellRenderer` [<https://www.codenameone.com/javadoc/com/codename1/ui/list/GenericListCellRenderer.html>] which means it will work with `MultiList` [<https://www.codenameone.com/javadoc/com/codename1/ui/list/MultiList.html>], `List` [<https://www.codenameone.com/javadoc/java/util/List.html>] & `ContainerList` [<https://www.codenameone.com/javadoc/com/codename1/ui/list/ContainerList.html>]. To use this support just define the name of the component (name not UIID) to end with `_URLImage` and give it an icon to use as the placeholder. This is easy to do in the multilist by changing the name of icon to `icon_URLImage` then using this in the data:

```
map.put("icon_URLImage", urlToActualImage);
```

Make sure you also set a "real" icon to the entry in the GUI builder or in handcoded applications. This is important since the icon will be implicitly extracted and used as the placeholder value. Everything else should be handled automatically. You can use `setDefaultAdapter` & `setAdapter` on the generic list cell renderer to install adapters for the images. The default is a scale adapter although we might change that to scale fill in the future.

```

Style s = UIManager.getInstance().getComponentStyle("Button");
FontImage p = FontImage.createMaterial(FontImage.MATERIAL_PORTRAIT, s);
EncodedImage placeholder = EncodedImage.createFromImage(p.scaled(p.getWidth() * 3, p.getHeight() * 4), false);

Form hi = new Form("MultiList", new BorderLayout());

ArrayList<Map<String, Object>> data = new ArrayList<>();

data.add(createListEntry("A Game of Thrones", "1996", "http://www.georgerrmartin.com/wp-
content/uploads/2013/03/GOTMTI2.jpg"));
data.add(createListEntry("A Clash Of Kings", "1998", "http://www.georgerrmartin.com/wp-
content/uploads/2012/08/clashofkings.jpg"));
data.add(createListEntry("A Storm Of Swords", "2000", "http://www.georgerrmartin.com/wp-
content/uploads/2013/03/stormswordsMTI.jpg"));
data.add(createListEntry("A Feast For Crows", "2005", "http://www.georgerrmartin.com/wp-
content/uploads/2012/08/feastforcrows.jpg"));
data.add(createListEntry("A Dance With Dragons", "2011", "http://georgerrmartin.com/gallery/art/dragons05.jpg"));
data.add(createListEntry("The Winds of Winter", "2016 (please, please, please)", "http://www.georgerrmartin.com/wp-
content/uploads/2013/03/GOTMTI2.jpg"));
data.add(createListEntry("A Dream of Spring", "Ugh", "http://www.georgerrmartin.com/wp-
content/uploads/2013/03/GOTMTI2.jpg"));

DefaultListModel<Map<String, Object>> model = new DefaultListModel<>(data);
MultiList ml = new MultiList(model);
ml.getUnselectedButton().setIconName("icon_URLImage");
ml.getSelectedButton().setIconName("icon_URLImage");
ml.getUnselectedButton().setIcon(placeholder);
ml.getSelectedButton().setIcon(placeholder);
hi.add(BorderLayout.CENTER, ml);

```

The `createListEntry` method then looks like this:

```

private Map<String, Object> createListEntry(String name, String date, String coverURL) {
    Map<String, Object> entry = new HashMap<>();
    entry.put("Line1", name);
    entry.put("Line2", date);
    entry.put("icon_URLImage", coverURL);
    entry.put("icon_URLImageName", name);
    return entry;
}

```

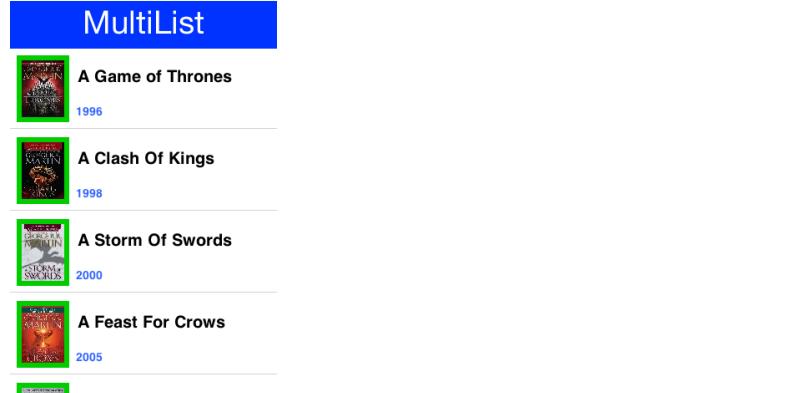


Figure 355. A URL image fetched dynamically into the list model

11. Events

Most events in Codename One are routed via the high level events (e.g. action listener) but sometimes we need access to low level events (e.g. when drawing via Graphics) that provide more fine grained access. Typically working with the higher level events is far more portable since it might map to different functionality on different devices.

11.1. High Level Events

High level events are broadcast using an `addListener/setListener` - publish/subscribe system. Most of them are channeled via the `EventDispatcher` [<https://www.codenameone.com/javadoc/com/codename1/ui/util/EventDispatcher.html>] class which further simplifies that and makes sending events correctly far easier.



All events are fired on the Event Dispatch Thread, the `EventDispatcher` makes sure of that.

11.1.1. Chain Of Events

Since all events fire on the EDT some complexities occur. E.g.:

We have two listeners monitoring the same event (or related events e.g. pointer event and button click event both of which will fire when the button is touched).

When the event occurs we can run into a scenario like this:

1. First event fires
2. It shows a blocking dialog or invokes an "AndWait" API
3. Second event fires only after the dialog was dismissed!

This happens because events are processed in-order per cycle. Since the old EDT cycle is stuck (because of the `Dialog`) the rest of the events within the cycle can't complete. New events are in a new EDT cycle so they can finish just fine!

A workaround to this issue is to wrap the code in a `callSerially`, you shouldn't do this universally as this can create a case of shifting the problem to the next EDT cycle. However, using `callSerially` will allow the current cycle to flush which should help.

Another workaround for the issue is avoiding blocking calls within an event chain.

11.1.2. Action Events

The most common high level event is the `ActionListener` [<https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html>] which allows binding a generic action event to pretty much anything. This is so ubiquitous in Codename One that it is even used for networking (as a base class) and for some of the low level event options.

E.g. we can bind an event callback for a `Button` [<https://www.codenameone.com/javadoc/com/codename1/ui/>]

Button.html] by using:

```
Button b = new Button("Click Me");
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        // button was clicked, you can do anything you want here...
    }
});
```

Or thanks to the Java 8 lambdas we can write it as:

```
Button b = new Button("Click Me");
b.addActionListener((ev) ->
    // button was clicked, you can do anything you want here...
);
```

Notice that the click will work whether the button was touched using a mouse, finger or keypad shortcut seamlessly with an action listener. Many components work with action events e.g. buttons, text components, slider etc.

There are quite a few types of high level event types that are more specific to requirements.

Types Of Action Events

When an action event is fired it is given a type, however this type might change as the event evolves e.g. a command triggered by a pointer event won't include details of the original pointer event.

You can get the event type from [getEventType\(\)](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getEventType-->], this also gives you a rather exhaustive list of the possible event types for the action event.

Source Of Event

[ActionEvent](#) has a source object, what that source is depends heavily on the event type. For most component based events this is the component but there are some nuances.

The [getComponent\(\)](#) method might not get the actual component. In case of a lead component such as [MultiButton](#) [<https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html>] the underlying [Button](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Button.html>] will be returned and not the [MultiButton](#) itself.

To get the component that you would logically think of as the source component use the [getActualComponent\(\)](#) method.

Event Consumption

An [ActionEvent](#) can be consumed, once consumed it will no longer proceed down the chain of event

processing. This is useful for some cases where we would like to block behavior from proceeding down the path.

E.g. the event dispatch thread allows us to listen to errors on the EDT using:

```
Display.getInstance().addEdtErrorHandler((e) -> {
    Exception err = (Exception)e.getSource();
    // ...
});
```

This will work great but you will still get the default error message from the EDT over that exception. To prevent the event from proceeding to the default error handling you can just do this:

```
Display.getInstance().addEdtErrorHandler((e) -> {
    e.consume();
    Exception err = (Exception)e.getSource();
    // ...
});
```

Notice that you can check if an event was already consumed using the `isConsumed()` method but it's pretty unlikely that you will receive a consumed event as the system will usually stop sending it.

NetworkEvent

[NetworkEvent](https://www.codenameone.com/javadoc/com/codename1/io/NetworkEvent.html) [https://www.codenameone.com/javadoc/com/codename1/io/NetworkEvent.html] is a subclass of [ActionEvent](#) that is passed to `actionPerformed` callbacks made in relation to generic network code. E.g.

```
NetworkManager.getInstance().addErrorListener(new ActionListener<NetworkEvent>() {
    public void actionPerformed(NetworkEvent ev) {
        // now we have access to the methods on NetworkEvent that provide more information about the network specific
        flags
    }
});
```

Or with Java 8 lambdas:

```
NetworkManager.getInstance().addErrorListener((ev) -> {
    // now we have access to the methods on NetworkEvent that provide more information about the network specific flags
});
```

The [NetworkEvent](#) allows the networking code to reuse the [EventDispatcher](#) infrastructure and to simplify event firing thru the EDT. But you should notice that some code might not be equivalent e.g. we could do this to read the input stream:

```

ConnectionRequest r = new ConnectionRequest() {
    @Override
    protected void readResponse(InputStream input) throws IOException {
        // read the input stream
    }
};

```

or we can do something similar using this code:

```

ConnectionRequest r = new ConnectionRequest();
r.addResponseListener((e) -> {
    byte[] data = (byte[])e.getMetaData();
    // work with the byte data
});

```

These seem very similar but they have one important distinction. The latter code is invoked on the EDT, so if **data** is big it might slow down processing significantly. The **ConnectionRequest** is invoked on the network thread and so can process any amount of data without slowing down the UI significantly.

11.1.3. DataChangeListener

The [DataChangedListener](#) [https://www.codenameone.com/javadoc/com/codename1/ui/events/DataChangedListener.html] is used in several places to indicate that the underlying model data has changed:

- [TextField](#) [https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html] - the text field provides an action listener but that only "fires" when the data input is complete. **DataChangeListener** fires with every key entered and thus allows functionality such as "auto complete" and is indeed used internally in the Codename One AutoCompleteTextField.
- [TableModel](#) [https://www.codenameone.com/javadoc/com/codename1/ui/table/TableModel.html] & [ListModel](#) [https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html] - the model for the [Table](#) [https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html] class notifies the view that its content has changed via this event, thus allowing the UI to refresh properly.

There is a very exhaustive example of search that is implemented using the **DataChangedListener** in the [Toolbar section](#) [https://www.codenameone.com/manual/components.html#Advanced-search-code].

11.1.4. FocusListener

The focus listener allows us to track the currently "selected" or focused component. It's not as useful as it used to be in feature phones.

You can bind a focus listener to the **Component** itself and receive an event when it gained focus, or you can bind the listener to the **Form** and receive events for every focus change event within the hierarchy.

11.1.5. ScrollListener

[ScrollListener](https://www.codenameone.com/javadoc/com/codename1/ui/events/ScrollListener.html) [https://www.codenameone.com/javadoc/com/codename1/ui/events/ScrollListener.html] allows tracking scroll events so UI elements can be adapted if necessary.

Normally scrolling is seamless and this event isn't necessary, however if developers wish to "shrink" or "fade" an element on scrolling this interface can be used to achieve that. Notice that you should bind the scroll listener to the actual scrollable component and not to an arbitrary component.

E.g. in this code from the [Flickr](#) demo the [Toolbar](#) [https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html] is faded based on scroll position:

```
public class CustomToolbar extends Toolbar implements ScrollListener {  
    private int alpha;  
  
    public CustomToolbar() {}  
  
    public void paintComponentBackground(Graphics g) {  
        int a = g.getAlpha();  
        g.setAlpha(alpha);  
        super.paintComponentBackground(g);  
        g.setAlpha(a);  
    }  
  
    public void scrollChanged(int scrollX, int scrollY, int oldscrollX, int oldscrollY) {  
        alpha = scrollY;  
        alpha = Math.max(alpha, 0);  
        alpha = Math.min(alpha, 255);  
    }  
}
```



There is a better way of implementing this exact effect using title animations [illustrated here](#) [https://www.codenameone.com/manual/components.html#title-animations-section].

11.1.6. SelectionListener

The [SelectionListener](#) [https://www.codenameone.com/javadoc/com/codename1/ui/events/SelectionListener.html] event is mostly used to broadcast list model selection changes to the list view. Since list supports the [ActionListener](#) event callback its usually the better option since it's more coarse grained.

[SelectionListener](#) gets fired too often for events and that might result in a performance penalty. When running on non-touch devices list selection could be changed with the keypad and only a specific fire button click would fire the action event, for those cases [SelectionListener](#) made a lot of sense. However, in touch devices this API isn't as useful.

11.1.7. StyleListener

[StyleListener](https://www.codenameone.com/javadoc/com/codename1/ui/events/StyleListener.html) [https://www.codenameone.com/javadoc/com/codename1/ui/events/StyleListener.html] allows components to track changes to the style objects. E.g. if the developer does something like:

```
cmp.getUnselectedStyle().setFgColor(0xffffffff);
```

This will trigger a style event that will eventually lead to the component being repainted. This is quite important for the component class but not a very important event for general user code. It is recommended that developers don't bind a style listener.

11.1.8. Event Dispatcher

When creating your own components and objects you sometimes want to broadcast your own events, for that purpose Codename One has the [EventDispatcher](https://www.codenameone.com/javadoc/com/codename1/ui/util/EventDispatcher.html) [https://www.codenameone.com/javadoc/com/codename1/ui/util/EventDispatcher.html] class which saves a lot of coding effort in this regard. E.g. if you wish to provide an [ActionListener](https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html) [https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html] event for components you can just add this to your class:

```
private final EventDispatcher listeners = new EventDispatcher();

public void addActionListener(ActionListener a) {
    listeners.addListener(a);
}

public void removeActionListener(ActionListener a) {
    listeners.removeListener(a);
}
```

Then when you need to broadcast the event just use:

```
private void fireEvent(ActionEvent ev) {
    listeners.fireActionEvent(ev);
}
```

11.2. Low Level Events

Low level events map to "system" events directly. Touch events are considered low level since they might expose platform specific nuances to your code.

E.g. one platform might send a very large number of events during drag while another might send only a few. Normally the high level event handling hides those complexities but some of them trickle down into the low level event handling.



Codename One tries to hide some of the complexities from the low level events as well. However, due to the nature of the event types it's a more challenging task.

Low level events can be bound in one of 3 ways:

- Use one of the add listener methods in [Form](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Form.html>] e.g. [addPointerPressedListener](#).
- Override one of the event callbacks on [Form](#)
- Override one of the event callbacks on a [Component](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/Component.html>].



When you override event callbacks on a [Component](#) the [Component](#) in question must be focusable and have focus at that point. This can be an advantage for some use cases as it will save you the need of handling unrelated events.

Each of those has advantages and disadvantages, specifically:

- 'Form' based events and callbacks deliver pointer events in the 'Form' coordinate space.
- 'Component' based events require focus
- 'Form' based events can block existing functionality from proceeding thru the event chain e.g. you can avoid calling super in a form event and thus block other events from happening (e.g. block a listener or component event from triggering).

Table 8. Event type map

	Listener	Override Form	Override Component
Coordinate System	Form	Form	Component
Block current functionality	Yes, just avoid super	Partially (event consume)	No

11.2.1. Low Level Event Types

There are two basic types of low level events: Key and Pointer.



Key events are only relevant to physical keys and will not trigger on virtual keyboard keys, to track those use a [TextField](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html>] with a [DataChangeListener](#) as mentioned above.

The pointer events (touch events) can be intercepted by overriding one or more of these methods in [Component](#) or [Form](#). Notice that unless you want to block functionality you should probably invoke [super](#) when overriding:

```

public void pointerDragged(int[] x, int[] y)
public void pointerDragged(final int x, final int y)
public void pointerPressed(int[] x, int[] y)
public void pointerPressed(int x, int y)
public void pointerReleased(int[] x, int[] y)
public void pointerReleased(int x, int y)
public void longPointerPress(int x, int y)
public void keyPressed(int keyCode)
public void keyReleased(int keyCode)
public void keyRepeated(int keyCode)

```

Notice that most pointer events have a version that accepts an array as an argument, this allows for multi-touch event handling by sending all the currently touched coordinates.

11.2.2. Drag Event Sanitation

Drag events are quite difficult to handle properly across devices. Some devices send a ridiculous number of events for even the lightest touch while others send too little. It seems like too many drag events wouldn't be a problem, however if we drag over a button then it might disable the buttons action event (since this might be the user trying to scroll).

Drag sensitivity is really about the component being dragged which is why we have the method `getDragRegionStatus` that allows us to "hint" to the drag API whether we are interested in drag events or not and if so in which directional bias.

E.g. if our component is a painting app where we are trying to draw using drag gestures we would use code such as:

```

public class MyComponent extends Component {
    protected int getDragRegionStatus(int x, int y) {
        return DRAG_REGIONLIKELY_DRAG_XY;
    }
}

```

This indicates that we want all drag events on both AXIS to be sent as soon as possible. Notice that this doesn't completely disable event sanitation.

11.3. BrowserNavigationCallback

The `BrowserNavigationCallback` [https://www.codenameone.com/javadoc/com/codename1/ui/events/BrowserNavigationCallback.html] isn't quite an "event" but there is no real "proper" classification for it.

 The callback method of this interface is invoked off the EDT! You must NEVER block this method and must not access UI or Codename One sensitive elements in this method!

The browser navigation callback is invoked directly from the native web component as it navigates

to a new page. Because of that it is invoked on the native OS thread and gives us a unique opportunity to handle the navigation ourselves as we see fit. That is why it MUST be invoked on the native thread, since the native browser is pending on our response to that method, spanning an invokeAndBlock/callSerially would be too slow and would bog down the browser.

You can use the browser navigation callback to change the UI or even to invoke Java code from JavaScript code e.g.:

```
bc.setBrowserNavigationCallback((url) -> {
    if(url.startsWith("http://click")) {
        Display.getInstance().callSerially(() -> bc.execute("fn('<p>You clicked!</p>')"));
        return false;
    }
    return true;
});
```


12. File System, Storage, Network & Parsing

12.1. Jar Resources

Resources that are packaged within the "JAR" don't really belong in this section of the developer guide but since they are often confused with [Storage/FileSystemStorage](#) this might be the best place to clarify what they are.

You can place arbitrary files within the `src` directory of a Codename One project. This file will get packaged into the final distribution. In standard Java SE you can usually do something like:

```
InputStream i = getClass().getResourceAsStream("/myfile");
```

This isn't guaranteed to work on all platforms and will probably fail on some. Instead you should use something such as:

```
InputStream i = Display.getInstance().getResourceAsStream(getClass(), "/myfile");
```

This isn't the only limitation though, you can use hierarchies so something like this would fail:

```
InputStream i = Display.getInstance().getResourceAsStream(getClass(), "/res/myfile");
```

You can't use relative paths either so this will fail as well (notice the lack of the first slash):

```
InputStream i = Display.getInstance().getResourceAsStream(getClass(), "myfile");
```

The reason for those limitations is portability, on iOS and Android resources behave quite differently so supporting the full Java SE semantics is unrealistic.



This is even worse in some regards. Because of the way iOS works with resources some unique file names might fail e.g. if you use the `@` character or have a `.` more than once (e.g. `file.ext.act`)

Notice that just like in Java SE, the entries within the "JAR" are read only and can't be modified. You can't gain access to the actual file, only to the stream!

12.2. Storage

[Storage](#) [<https://www.codenameone.com/javadoc/com/codename1/io/Storage.html>] is accessed via the [Storage](#) class. It is a flat filesystem like interface and contains the ability to list/delete and write to named storage entries.

The [Storage](#) API also provides convenient methods to write objects to [Storage](#) and read them from

Storage specifically `readObject` & `writeObject`.



The objects in Storage are usually deleted when an app is uninstalled but are retained between application updates. A notable exception is Android which, on some devices, keeps by default objects in Storage after app uninstalling. To force Android to remove them on app uninstalling, use the build hint `android.allowBackup=false`

The sample code below demonstrates listing the content of the storage, adding/viewing and deleting entries within the storage:

```

Toolbar.setGlobalToolbar(true);
Form hi = new Form("Storage", new BoxLayout(BoxLayout.Y_AXIS));
hi.getToolbar().addCommandToRightBar("+", null, e -> {
    TextField tf = new TextField("", "File Name", 20, TextField.ANY);
    TextArea body = new TextArea(5, 20);
    body.setHint("File Body");
    Command ok = new Command("OK");
    Command cancel = new Command("Cancel");
    Command result = Dialog.show("File Name", BorderLayout.north(tf).add(BorderLayout.CENTER, body), ok, cancel);
    if(ok == result) {
        try(OutputStream os = Storage.getInstance().createOutputStream(tf.getText())); {
            os.write(body.getText().getBytes("UTF-8"));
            createFileEntry(hi, tf.getText());
            hi.getContentPane().animateLayout(250);
        } catch(IOException err) {
            Log.e(err);
        }
    }
});
for(String file : Storage.getInstance().listEntries()) {
    createFileEntry(hi, file);
}
hi.show();

private void createFileEntry(Form hi, String file) {
    Label fileField = new Label(file);
    Button delete = new Button();
    Button view = new Button();
    FontImage.setMaterialIcon(delete, FontImage.MATERIAL_DELETE);
    FontImage.setMaterialIcon(view, FontImage.MATERIAL_OPEN_IN_NEW);
    Container content = BorderLayout.center(fileField);
    int size = Storage.getInstance().entrySize(file);
    content.add(BorderLayout.EAST, BoxLayout.encloseX(new Label(size + "bytes"), delete, view));
    delete.addActionListener(e -> {
        Storage.getInstance().deleteStorageFile(file);
        content.setY(hi.getWidth());
        hi.getContentPane().animateUnlayoutAndWait(150, 255);
        hi.removeComponent(content);
        hi.getContentPane().animateLayout(150);
    });
    view.addActionListener(e -> {
        try(InputStream is = Storage.getInstance().createInputStream(file);) {
            String s = Util.readToString(is, "UTF-8");
            Dialog.show(file, s, "OK", null);
        } catch(IOException err) {
            Log.e(err);
        }
    });
    hi.add(content);
}

```

Storage		
g1-TM1ATLyI0Y	1848bytes	[] []
g1-TM1ATLzITN5	2205bytes	[] []
g1-TNIATN3cDO+	2240bytes	[] []
g1-TNIATN5ID01	2158bytes	[] []
g1-TNIATN5ID02	2310bytes	[] []
g1-TNIATN5ID03	2215bytes	[] []
g1-TNIAzM0gTNy	2051bytes	[] []
g1-TNIAzM1gt05	2041bytes	[] []
g1-TNIAzM4KTM!	2001bytes	[] []
g1-TOTATMzijNM	1907bytes	[] []
g1-TOlUTM1ETN	2153bytes	[] []
Hi File	21bytes	[] []
list0	76690bytes	[] []

Figure 356. List of files within the storage

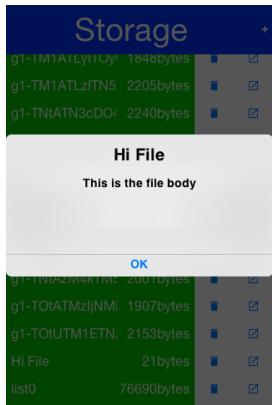


Figure 357. Content of a file added to the storage

12.2.1. The Preferences API

Storage also offers a very simple API in the form of the [Preferences](#) [<https://www.codenameone.com/javadoc/com/codename1/io/Preferences.html>] class. The [Preferences](#) class allows developers to store simple variables, strings, numbers, booleans etc. in storage without writing any storage code. This is a common use case within applications e.g. you have a server token that you need to store you can store it like this:

```
Preferences.set("token", myToken);
```

You can then read the token like this:

```
String token = Preferences.get("token", null);
```

This gets somewhat confusing with primitive numbers e.g. if you use `Preferences.set("primitiveLongValue", myLongNumber)` then invoke `Preferences.get("primitiveLongValue", 0)` you might get an exception!

This would happen because the value is physically a [Long](#) object but you are trying to get an [Integer](#). The workaround is to remain consistent and use code like this `Preferences.get("primitiveLongValue", (long)0)`.



12.3. File System

[FileSystemStorage](#) [https://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html] provides file system access. It maps to the underlying OS's file system API providing most of the common operations expected from a file API somewhat in the vain of `java.io.File` & `java.io.FileInputStream` e.g. opening, renaming, deleting etc.

Notice that the file system API is somewhat platform specific in its behavior. All paths used the API should be absolute otherwise they are not guaranteed to work.

The main reason `java.io.File` & `java.io.FileInputStream` weren't supported directly has a lot to do with the richness of those two API's. They effectively allow saving a file anywhere, however mobile devices are far more restrictive and don't allow apps to see/modify files that are owned by other apps.

12.3.1. File Paths & App Home

All paths in `FileSystemStorage` are absolute, this simplifies the issue of portability significantly since the concept of relativity and current working directory aren't very portable.

All URL's use the `/` as their path separator we try to enforce this behavior even in Windows.

Directories end with the `/` character and thus can be easily distinguished by their name.

The `FileSystemStorage` API provides a `getRoots()` call to list the root directories of the file system (you can then "dig in" via the `listFiles` API). However, this is confusing and unintuitive for developers.

To simplify the process of creating/reading files we added the `getAppHomePath()` method. This method allows us to obtain the path to a directory where files can be stored/read.

We can use this directory to place an image to share as we did in the [share sample](#) [https://www.codenameone.com/manual/components.html#sharebutton-section].



A common Android hack is to write files to the SDCard storage to share them among apps. Android 4.x disabled the ability to write to arbitrary directories on the SDCard even when the appropriate permission was requested.

A more advanced usage of the `FileSystemStorage` API can be a `FileSystemStorage Tree`:

```

Form hi = new Form("FileSystemTree", new BorderLayout());
TreeModel tm = new TreeModel() {
    @Override
    public Vector getChildren(Object parent) {
        String[] files;
        if(parent == null) {
            files = FileSystemStorage.getInstance().getRoots();
            return new Vector<Object>(Arrays.asList(files));
        } else {
            try {
                files = FileSystemStorage.getInstance().listFiles((String)parent);
            } catch(IOException err) {
                Log.e(err);
                files = new String[0];
            }
        }
        String p = (String)parent;
        Vector result = new Vector();
        for(String s : files) {
            result.add(p + s);
        }
        return result;
    }

    @Override
    public boolean isLeaf(Object node) {
        return !FileSystemStorage.getInstance().isDirectory((String)node);
    }
};

Tree t = new Tree(tm) {
    @Override
    protected String childToDisplayLabel(Object child) {
        String n = (String)child;
        int pos = n.lastIndexOf("/");
        if(pos < 0) {
            return n;
        }
        return n.substring(pos);
    }
};
hi.add(BorderLayout.CENTER, t);
hi.show();

```

FileSystemTree

```
/  
./DS_Store  
/400px-AGameOfThrones.jpg  
/A Clash Of Kings  
/A Clash Of Kings 1  
/A Dance With Dragons  
/A Dance With Dragons 4  
/A Dance With Dragons 4imageURLTMP  
/A Feast For Crows  
/A Feast For Crows 3  
/A Game of Thrones  
/A Game of Thrones 0  
/A Storm Of Swords
```

Figure 358. Simple sample of a tree for the `FileSystemStorage` API

12.3.2. Storage vs. File System

The question of storage vs. file system is often confusing for novice mobile developers. This embeds two separate questions:

- Why are there 2 API's where one would have worked?
- Which one should I pick?

The main reasons for the 2 API's are technical. Many OS's provide 2 ways of accessing data specific to the app and this is reflected within the API. E.g. on Android the `FileSystemStorage` maps to API's such as `java.io.FileInputStream` whereas the `Storage` maps to `Context.openFileInput()`.

The secondary reason for the two API's is conceptual. `FileSystemStorage` is more powerful and in a sense provides more ways to fail, this is compounded by the complex on-device behavior of the API. `Storage` is designed to be friendlier to the uninitiated and more portable.

You should pick `Storage` unless you have a specific requirement that prevents it. Some API's such as `Capture` expect a `FileSystemStorage` URI so in those cases this would also be a requirement.

Another case where `FileSystemStorage` is beneficial is the case of hierarchy or native API usage. If you need a directory structure or need to communicate with a native API the `FileSystemStorage` approach is usually easier.



In some OS's the `FileSystemStorage` API can find the content of the `Storage` API. As one is implemented on top of the other. This is undocumented behavior that can change at any moment!

To summarize the differences between the 3 file storage options:

Table 9. Compare Storage, FileSystem & Jar Resources

Option	Storage	File System	JAR Resource
Main Use Case	General application Data	Low level access	Ship data within the app
Initial State	Blank	Blank	As defined by developer

Option	Storage	File System	JAR Resource
Modifiable	Yes	Yes	No
Supports Hierarchies	No	Yes	No

12.4. SQL

Most new devices contain one version of sqlite or another; sqlite is a very lightweight SQL database designed for embedding into devices. For portability we recommend avoiding SQL altogether since it is both fragmented between devices (different sqlite versions) and isn't supported on all devices.

In general SQL seems overly complex for most embedded device programming tasks.

Portability Of SQLite

SQLite is supported on iOS, Android, RIM, Desktop & JavaScript builds. However, the JavaScript version of SQL has been deprecated and isn't supported on all platforms.

You will notice that at this time support is still missing from the Windows builds.

The biggest issue with SQLite portability is in iOS. The SQLite version for most platforms is threadsafe and as a result very stable. However, the iOS version is not!

This might not seem like a big deal normally, however if you forget to close a connection the GC might close it for you thus producing a crash. This is such a common occurrence that Codename One logs a warning when the GC collects a database resource on the simulator.

SQL is pretty powerful and very well suited for common tabular data. The Codename One SQL API is similar in spirit to JDBC but considerably simpler since many of the abstractions of JDBC designed for pluggable database architecture make no sense for a local database.

The [Database](https://www.codenameone.com/javadoc/com/codename1/db/Database.html) [https://www.codenameone.com/javadoc/com/codename1/db/Database.html] API is a high level abstraction that allows you to open an arbitrary database file using syntax such as:

```
Database db = Display.getInstance().openOrCreate("databaseName");
```

Some SQLite apps ship with a "ready made" database. We allow you to replace the DB file by using the code:

```
String path = Display.getInstance().getDatabasePath("databaseName");
```

You can then use the [FileSystemStorage](#) class to write the content of your DB file into the path. Notice that it must be a valid SQLite file!



`getDatabasePath()` is not supported in the Javascript port. It will always return null.

This is very useful for applications that need to synchronize with a central server or applications that ship with a large database as part of their core product.

Working with a database is pretty trivial, the application logic below can send arbitrary queries to the database and present the results in a **Table**. You can probably integrate this code into your app as a debugging tool:

```
Toolbar.setGlobalToolbar(true);
Style s = UIManager.getInstance().getComponentStyle("TitleCommand");
FontImage icon = FontImage.createMaterial(FontImage.MATERIAL_QUERY_BUILDER, s);
Form hi = new Form("SQL Explorer", new BorderLayout());
hi.getToolbar().addCommandToRightBar("", icon, (e) -> {
    TextArea query = new TextArea(3, 80);
    Command ok = new Command("Execute");
    Command cancel = new Command("Cancel");
    if(Dialog.show("Query", query, ok, cancel) == ok) {
        Database db = null;
        Cursor cur = null;
        try {
            db = Display.getInstance().openOrCreate("MyDB.db");
            if(query.getText().startsWith("select")) {
                cur = db.executeQuery(query.getText());
                int columns = cur.getColumnCount();
                hi.removeAll();
                if(columns > 0) {
                    boolean next = cur.next();
                    if(next) {
                        ArrayList<String[]> data = new ArrayList<>();
                        String[] columnNames = new String[columns];
                        for(int iter = 0 ; iter < columns ; iter++) {
                            columnNames[iter] = cur.getColumnName(iter);
                        }
                        while(next) {
                            Row currentRow = cur.getRow();
                            String[] currentRowArray = new String[columns];
                            for(int iter = 0 ; iter < columns ; iter++) {
                                currentRowArray[iter] = currentRow.getString(iter);
                            }
                            data.add(currentRowArray);
                            next = cur.next();
                        }
                        Object[][] arr = new Object[data.size()][];
                        data.toArray(arr);
                        hi.add(BorderLayout.CENTER, new Table(new DefaultTableModel(columnNames, arr)));
                    } else {
                        hi.add(BorderLayout.CENTER, "Query returned no results");
                    }
                } else {
                    hi.add(BorderLayout.CENTER, "Query returned no results");
                }
            } else {
                db.execute(query.getText());
                hi.add(BorderLayout.CENTER, "Query completed successfully");
            }
            hi.revalidate();
        } catch(IOException err) {
            Log.e(err);
        }
    }
})
```

```

        hi.removeAll();
        hi.add(BorderLayout.CENTER, "Error: " + err);
        hi.revalidate();
    } finally {
        Util.cleanup(db);
        Util.cleanup(cur);
    }
}
});
hi.show();

```

SQL Explorer		
id	name	num
1	Chen Fishbein	36.0
2	Shai Almog	36.0
3	Bill gates	56.0
4	Mark zuckerberg	28.0
5	Larry Page	39.0
6	Name	30.0
7	Name 2	40.0

Figure 359. Querying the temp demo generated by the SQLDemo application

SQL Explorer		
id	name	num
1	Chen Fishbein	36.0
2	Shai Almog	36.0
3	Bill gates	56.0

Query

Execute Cancel

Figure 360. Issuing a query

12.5. Network Manager & Connection Request

One of the more common problems in Network programming is spawning a new thread to handle the network operations. In Codename One this is done seamlessly and becomes unessential thanks to the [NetworkManager](http://www.codenameone.com/javadoc/com/codenname1/io/NetworkManager.html) [http://www.codenameone.com/javadoc/com/codenname1/io/NetworkManager.html].

[NetworkManager](#) effectively alleviates the need for managing network threads by managing the complexity of network threading. The connection request class can be used to facilitate web service requests when coupled with the JSON/XML parsing capabilities.

To open a connection one needs to use a [ConnectionRequest](http://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html) [http://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html] object, which has some similarities to the networking mechanism in JavaScript but is obviously somewhat more elaborate.

You can send a get request to a URL using something like:

```

ConnectionRequest request = new ConnectionRequest(url, false);
request.addResponseListener((e) -> {
    // process the response
});

// request will be handled asynchronously
NetworkManager.getInstance().addToQueue(request);

```

Notice that you can also implement the same thing and much more by avoiding the response listener code and instead overriding the methods of the `ConnectionRequest` class which offers multiple points to override e.g.

```

ConnectionRequest request = new ConnectionRequest(url, false) {
    protected void readResponse(InputStream input) {
        // just read from the response input stream
    }

    protected void postResponse() {
        // invoked on the EDT after processing is complete to allow the networking code
        // to update the UI
    }

    protected void buildRequestBody(OutputStream os) {
        // writes post data, by default this "just works" but if you want to write this
        // manually then override this
    }
};

NetworkManager.getInstance().addToQueue(request);

```



Notice that overriding `buildRequestBody(OutputStream)` will only work for `POST` requests and will replace writing the arguments



You don't need to close the output/input streams passed to the request methods. They are implicitly cleaned up.

`NetworkManager` also supports synchronous requests which work in a similar way to `Dialog` via the `invokeAndBlock` call and thus don't block the EDT^[9] illegally. E.g. you can do something like this:

```

ConnectionRequest request = new ConnectionRequest(url, false);
// request will be handled synchronously
NetworkManager.getInstance().addToQueueAndWait(request);
byte[] resultOfRequest = request.getData();

```

Notice that in this case the `addToQueueAndWait` method returned after the connection completed. Also notice that this was totally legal to do on the EDT!

12.5.1. Threading

By default the `NetworkManager` launches with a single network thread. This is sufficient for very simple applications that don't do too much networking but if you need to fetch many images concurrently and perform web services in parallel this might be an issue.



Once you increase the thread count there is no guarantee of order for your requests. Requests might not execute in the order with which you added them to the queue!

To update the number of threads use:

```
NetworkManager.getInstance().updateThreadCount(4);
```

All the callbacks in the `ConnectionRequest` occur on the network thread and **not on the EDT!**

There is one exception to this rule which is the `postResponse()` method designed to update the UI after the networking code completes.



Never change the UI from a `ConnectionRequest` callback. You can either use a listener on the `ConnectionRequest`, use `postResponse()` (which is the only exception to this rule) or wrap your UI code with `callSerially`.

12.5.2. Arguments, Headers & Methods

HTTP/S is a complex protocol that expects complex encoded data for its requests. Codename One tries to simplify and abstract most of these complexities behind common sense API's while still providing the full low level access you would expect from such an API.

Arguments

HTTP supports several "request methods", most commonly `GET` & `POST` but also a few others such as `HEAD`, `PUT`, `DELETE` etc.

Arguments in HTTP are passed differently between `GET` and `POST` methods. That is what the `setPost` method in Codename One determines, whether arguments added to the request should be placed using the `GET` semantics or the `POST` semantics.

So if we continue our example from above we can do something like this:

```
ConnectionRequest request = new ConnectionRequest(url, false);
request.addArgument("MyArgName", value);
```

This will implicitly add a get argument with the content of `value`. Notice that we don't really care what value is. It's implicitly HTTP encoded based on the get/post semantics. In this case it will use the get encoding since we passed `false` to the constructor.

A simpler implementation could do something like this:

```
ConnectionRequest request = new ConnectionRequest(url +  
    "MyArgName=" + Util.encodeUrl(value), false);
```

This would be almost identical but doesn't provide the convenience for switching back and forth between **GET/POST** and it isn't as fluent.

We can skip the encoding in complex cases where server code expects illegal HTTP characters (this happens) using the **addArgumentNoEncoding** method. We can also add multiple arguments with the same key using **addArgumentArray**.

Methods

As we explained above, the **setPost()** method allows us to manipulate the get/post semantics of a request. This implicitly changes the **POST** or **GET** method submitted to the server.

However, if you wish to have finer grained control over the submission process e.g. for making a **HEAD** request you can do this with code like:

```
ConnectionRequest request = new ConnectionRequest(url, false);  
request.setHttpMethod("HEAD");
```

Headers

When communicating with HTTP servers we often pass data within headers mostly for authentication/authorization but also to convey various properties.

Some headers are builtin as direct API's e.g. content type is directly exposed within the API since it's a pretty common use case. We can set the content type of a post request using:

```
ConnectionRequest request = new ConnectionRequest(url, true);  
request.setContentType("text/xml");
```

We can also add any arbitrary header type we want, e.g. a very common use case is basic authorization where the authorization header includes the Base64 encoded user/password combination as such:

```
String authCode = user + ":" + password;  
String authHeader = "Basic " + Base64.encode(authCode.getBytes());  
request.addRequestHeader("Authorization", authHeader);
```

This can be quite tedious to do if you want all requests from your app to use this header. For this use case you can just use:

```
String authCode = user + ":" + password;
String authHeader = "Basic " + Base64.encode(authCode.getBytes());
NetworkManager.getInstance().addDefaultHeader("Authorization", authHeader);
```

Server Headers

Server returned headers are a bit trickier to read. We need to subclass the connection request and override the `readHeaders` method e.g.:

```
ConnectionRequest request = new ConnectionRequest(url, false) {
    protected void readHeaders(Object connection) throws IOException {
        String[] headerNames = getHeaderFieldNames(connection);
        for(String headerName : headerNames) {
            String headerValue = getHeader(headerName);
            //....
        }
    }
    protected void readResponse(InputStream input) {
        // just read from the response input stream
    }
};
NetworkManager.getInstance().addToQueue(request);
```

Here we can extract the headers one by one to handle complex headers such as cookies, authentication etc.

Error Handling

As you noticed above practically all of the methods in the `ConnectionRequest` throw `IOException`. This allows you to avoid the `try/catch` semantics and just let the error propagate up the chain so it can be handled uniformly by the application.

There are two distinct places where you can handle a networking error:

- The `ConnectionRequest` - by overriding callback methods
- The `NetworkManager` error handler

Notice that the `NetworkManager` error handler takes precedence thus allowing you to define a global policy for network error handling by consuming errors.

E.g. if I would like to block all network errors from showing anything to the user I could do something like this:

```
NetworkManager.getInstance().addToQueue(request);
NetworkManager.getInstance().addErrorListener((e) -> e.consume());
```

The error listener is invoked first with the `NetworkEvent` [<https://www.codenameone.com/javadoc/com/>

codename1/io/NetworkEvent.html] matching the error. Consuming the event prevents it from propagating further down the chain into the `ConnectionRequest` callbacks.

We can also override the error callbacks of the various types in the request e.g. in the case of a server error code we can do:

```
ConnectionRequest request = new ConnectionRequest(url, false) {  
    protected void handleErrorResponseCode(int code, String message) {  
        if(code == 444) {  
            // do something  
        }  
    }  
    protected void readResponse(InputStream input) {  
        // just read from the response input stream  
    }  
};  
NetworkManager.getInstance().addToQueue(request);
```



The error callback callback is triggered in the network thread!

As a result it can't access the UI to show a `Dialog` or anything like that.

Another approach is to use the `setFailSilently(true)` method on the `ConnectionRequest`. This will prevent the `ConnectionRequest` from displaying any errors to the user. It's a very powerful strategy if you use the synchronous version of the API's e.g.:

```
ConnectionRequest request = new ConnectionRequest(url, false);  
request.setFailSilently(true);  
NetworkManager.getInstance().addToQueueAndWait(request);  
if(request.getResponseCode() != 200) {  
    // probably an error...  
}
```



This code will only work with the synchronous "AndWait" version of the method since the response code will take a while to return for the non-wait version.

Error Stream

When we get an error code that isn't 200/300 we ignore the result. This is problematic as the result might contain information we need. E.g. many webservices provide further XML/JSON based details describing the reason for the error code.

Calling `setReadResponseForErrors(true)` will trigger a mode where even errors will receive the `readResponse` callback with the error stream. This also means that API's like `getData` and the listener API's will also work correctly in case of error.

12.5.3. GZIP

Gzip is a very common compression format based on the lz algorithm, it's used by web servers around the world to compress data.

Codename One supports [GZipInputStream](https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZInputStream.html) [<https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZInputStream.html>] and [GZipOutputStream](https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZOutputStream.html) [<https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZOutputStream.html>], which allow you to compress data seamlessly into a stream and extract compressed data from a stream. This is very useful and can be applied to every arbitrary stream.

Codename One also features a [GZConnectionRequest](https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZConnectionRequest.html) [<https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZConnectionRequest.html>], which will automatically unzip an HTTP response if it is indeed gzipped. Notice that some devices (iOS) always request gzip'ed data and always decompress it for us, however in the case of iOS it doesn't remove the gziped header. The [GZConnectionRequest](#) is aware of such behaviors so it's better to use that when connecting to the network (if applicable).

By default [GZConnectionRequest](#) doesn't request gzipped data (only unzips it when its received) but its pretty easy to do so just add the HTTP header `Accept-Encoding: gzip` e.g.:

```
GZConnectionRequest con = new GZConnectionRequest();
con.addRequestHeader("Accept-Encoding", "gzip");
```

Do the rest as usual and you should have smaller responses from the servers.

12.5.4. File Upload

[MultipartRequest](https://www.codenameone.com/javadoc/com/codename1/io/MultipartRequest.html) [<https://www.codenameone.com/javadoc/com/codename1/io/MultipartRequest.html>] tries to simplify the process of uploading a file from the local device to a remote server.

You can always submit data in the [buildRequestBody](#) but this is flaky and has some limitations in terms of devices/size allowed. HTTP standardized file upload capabilities thru the multipart request protocol, this is implemented by countless servers and is well documented. Codename One supports this out of the box:

```
MultipartRequest request = new MultipartRequest();
request.setUrl(url);
request.addData("myFileName", fullPathToFile, "text/plain")
NetworkManager.getInstance().addToQueue(request);
```



[MultipartRequest](#) is a [ConnectionRequest](#) most stuff you expect from there should work. Even `addArgument` etc.

Since we assume most developers reading this will be familiar with Java here is the way to implement the multipart upload in the servlet API:

```

@WebServlet(name = "UploadServlet", urlPatterns = {"/upload"})
@MultipartConfig(fileSizeThreshold = 1024 * 1024 * 100, // 10 MB
                 maxFileSize = 1024 * 1024 * 150, // 50 MB
                 maxRequestSize = 1024 * 1024 * 200)      // 100 MB
public class UploadServlet extends HttpServlet {

    @Override
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        Collection<Part> parts = req.getParts();
        Part data = parts.iterator().next();
        try(InputStream is = data.getInputStream();) {}
        // store or do something with the input stream
    }
}

```

12.5.5. Parsing

Codename One has several built in parsers for JSON, XML, CSV & Properties formats. You can use those parsers to read data from the Internet or data that is shipping with your product. E.g. use the CSV data to setup default values for your application.

All our parsers are designed with simplicity and small distribution size; they don't validate and will fail in odd ways when faced with broken data. The main logic behind this is that validation takes up CPU time on the device where CPU is a precious resource.

Parsing CSV

CSV is probably the easiest to use, the "Comma Separated Values" format is just a list of values separated by commas (or some other character) with new lines to indicate another row in the table. These usually map well to an Excel spreadsheet or database table and are supported by default in all spreadsheets.

To parse a CSV just use the [CSVParser](https://www.codenameone.com/javadoc/com/codenname1/io/CSVParser.html) [https://www.codenameone.com/javadoc/com/codenname1/io/CSVParser.html] class as such:

```

Form hi = new Form("CSV Parsing", new BorderLayout());
CSVParser parser = new CSVParser();
try(Reader r = new CharArrayReader("1997,Ford,E350,\"Super, \"luxurious\" truck\"".toCharArray());) {
    String[][] data = parser.parse(r);
    String[] columnNames = new String[data[0].length];
    for(int iter= 0 ; iter < columnNames.length ; iter++) {
        columnNames[iter] = "Col " + (iter + 1);
    }
    TableModel tm = new DefaultTableModel(columnNames, data);
    hi.add(BorderLayout.CENTER, new Table(tm));
} catch(IOException err) {
    Log.e(err);
}
hi.show();

```

CSV Parsing

Col 1	Col 2	Col 3	Col 4
1997	Ford	E350	Super, "luxurious" truck

Figure 361. CSV parsing results, notice the properly escaped parentheses and comma

The data contains a two dimensional array of the CSV content. You can change the delimiter character by using the `CSVParser` constructor that accepts a character.

 Notice that we used `CharArrayReader` from the `com.codename1.io` package for this sample. Normally you would want to use `java.util.InputStreamReader` for real world data.

JSON

The JSON ("Java Script Object Notation") format is popular on the web for passing values to/from webservices since it works so well with JavaScript. Parsing JSON is just as easy but has two different variations. You can use the `JSONParser` [<https://www.codenameone.com/javadoc/com/codename1/io/JSONParser.html>] class to build a tree of the JSON data as such:

```

JSONParser parser = new JSONParser();
Hashtable response = parser.parse(reader);

```

The response is a `Map` containing a nested hierarchy of `Collection (java.util.List)`, Strings and numbers to represent the content of the submitted JSON. To extract the data from a specific path just iterate the `Map` keys and recurs into it.

The sample below uses results from [an API of ice and fire](#) [<https://anapioficeandfire.com/>] that queries structured data about the "Song Of Ice & Fire" book series. Here is a sample result returned from the API for the query <http://www.anapioficeandfire.com/api/characters?page=5&pageSize=3> :

```

[
{
  "url": "http://www.anapioficeandfire.com/api/characters/13",

```

```
"name": "Chayle",
"culture": "",
"born": "",
"died": "In 299 AC, at Winterfell",
"titles": [
    "Septon"
],
"aliases": [],
"father": "",
"mother": "",
"spouse": "",
"allegiances": [],
"books": [
    "http://www.anapioficeandfire.com/api/books/1",
    "http://www.anapioficeandfire.com/api/books/2",
    "http://www.anapioficeandfire.com/api/books/3"
],
"povBooks": [],
"tvSeries": [],
"playedBy": []
},
{
"url": "http://www.anapioficeandfire.com/api/characters/14",
"name": "Gillam",
"culture": "",
"born": "",
"died": "",
"titles": [
    "Brother"
],
"aliases": [],
"father": "",
"mother": "",
"spouse": "",
"allegiances": [],
"books": [
    "http://www.anapioficeandfire.com/api/books/5"
],
"povBooks": [],
"tvSeries": [],
"playedBy": []
},
{
"url": "http://www.anapioficeandfire.com/api/characters/15",
"name": "High Septon",
"culture": "",
"born": "",
"died": "",
"titles": [
    "High Septon",
    "His High Holiness",

```

```

    "Father of the Faithful",
    "Voice of the Seven on Earth"
],
"aliases": [
    "The High Sparrow"
],
"father": "",
"mother": "",
"spouse": "",
"allegiances": [],
"books": [
    "http://www.anapioficeandfire.com/api/books/5",
    "http://www.anapioficeandfire.com/api/books/8"
],
"povBooks": [],
"tvSeries": [
    "Season 5"
],
"playedBy": [
    "Jonathan Pryce"
]
}
]

```

We will place that into a file named "anapioficeandfire.json" in the src directory to make the next sample simpler:

```

Form hi = new Form("JSON Parsing", new BoxLayout(BoxLayout.Y_AXIS));
JSONParser json = new JSONParser();
try(Reader r = new InputStreamReader(Display.getInstance().getResourceAsStream(getClass(), "/anapioficeandfire.json"),
"UTF-8")) {
    Map<String, Object> data = json.parseJSON(r);
    java.util.List<Map<String, Object>> content = (java.util.List<Map<String, Object>>)data.get("root"); ①
    for(Map<String, Object> obj : content) { ②
        String url = (String)obj.get("url");
        String name = (String)obj.get("name");
        java.util.List<String> titles = (java.util.List<String>)obj.get("titles"); ③
        if(name == null || name.length() == 0) {
            java.util.List<String> aliases = (java.util.List<String>)obj.get("aliases");
            if(aliases != null && aliases.size() > 0) {
                name = aliases.get(0);
            }
        }
        MultiButton mb = new MultiButton(name);
        if(titles != null && titles.size() > 0) {
            mb.setTextLine2(titles.get(0));
        }
        mb.addActionListener((e) -> Display.getInstance().execute(url));
        hi.add(mb);
    }
} catch(IOException err) {
    Log.e(err);
}
hi.show();

```

① The `JSONParser` returns a `Map` which is great if the root object is a `Map` but in some cases its a list of

elements (as is the case above). In this case a special case "root" element is created to contain the actual list of elements.

- ② We rely that the entries are all maps, this might not be the case for every API type.
- ③ Notice that the "titles" and "aliases" entries are both lists of elements. We use `java.util.List` to avoid a clash with `com.codename1.ui.List`.

JSON Parsing

Chayle
Septon
Gillam
Brother
High Septon
High Septon

Figure 362. Parsed JSON result, clicking the elements opens the URL from the JSON



The structure of the returned map is sometimes unintuitive when looking at the raw JSON. The easiest thing to do is set a breakpoint on the method and use the inspect variables capability of your IDE to inspect the returned element hierarchy while writing the code to extract that data

An alternative approach is to use the static `data parse()` method of the `JSONParser` class and implement a callback parser e.g.:

```
JSONParser.parse(reader, callback);
```

Notice that a static version of the method is used! The `callback` object is an instance of the `JSONParseCallback` interface, which includes multiple methods. These methods are invoked by the parser to indicate internal parser states, this is similar to the way traditional XML SAX event parsers work.

XML Parsing

The `XMLParser` [<https://www.codenameone.com/javadoc/com/codename1/xml/XMLParser.html>] started its life as an HTML parser built for displaying mobile HTML. That usage has since been deprecated but the parser can still parse many HTML pages and is very "loose" in terms of verification. This is both good and bad as the parser will work with invalid data without complaining.

The simplest usage of `XMLParser` looks a bit like this:

```
XMLParser parser = new XMLParser();
Element elem = parser.parse(reader);
```

The `Element` [<https://www.codenameone.com/javadoc/com/codename1/xml/Element.html>] contains children and attributes. It represents a tag within the XML document and even the root document itself. You can iterate over the XML tree to extract the data from within the XML file.

We've had a great sample of working with **XMLParser** in the Tree Section [<https://www.codenameone.com/manual/components.html#tree-section>] of this guide.

XMLParser has the complimentary **XMLWriter** [<https://www.codenameone.com/javadoc/com/codename1/xml/XMLWriter.html>] class which can generate XML from the **Element** hierarchy. This allows a developer to mutate (modify) the elements and save them to a writer stream.

XPath Processing

The **Result** [<https://www.codenameone.com/javadoc/com/codename1/processing/Result.html>] class provides a subset of **XPath** [https://www.w3schools.com/xml/xpath_intro.asp], but it is not limited to just XML documents, it can also work with JSON documents, and even with raw **Map** objects.

Lets start by demonstrating how to process a response from the **Google Reverse Geocoder API** [<https://developers.google.com/maps/documentation/geocoding/>]. Lets start with this XML snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
    <status>OK</status>
    <result> <!-- (irrelevant content removed) -->
        <address_component>
            <long_name>London</long_name>
            <short_name>London</short_name>
            <type>locality</type>
            <type>political</type>
        </address_component>
        <!-- (irrelevant content removed) -->
        <address_component>
            <long_name>Ontario</long_name>
            <short_name>ON</short_name>
            <type>administrative_area_level_1</type>
            <type>political</type>
        </address_component>
        <address_component>
            <long_name>Canada</long_name>
            <short_name>CA</short_name>
            <type>country</type>
            <type>political</type>
        </address_component>
    </result>
</GeocodeResponse>
```

We want to extract some of the data above into simpler string results. We can do this using:

```
Result result = Result.fromContent(input, Result.XML);
String country = result.getAsString("/result/address_component[type='country']/long_name");
String region = result.getAsString("/result/address_component[type='administrative_area_level_1']/long_name");
String city = result.getAsString("/result/address_component[type='locality']/long_name");
```

If you are at all familiar with processing responses from webservices, you will notice that what would normally require several lines of code of selecting and testing nodes in regular java can now be done in a single line using the new path expressions.

In the code above, input can be any of:

- `InputStream` directly from `ConnectionRequest.readResponse(java.io.InputStream)`.
- XML or JSON document in the form of a {@code String}
- XML DOM `Element` [<https://www.codenameone.com/javadoc/com/codename1/xml/Element.html>] returned from `XMLParser` [<https://www.codenameone.com/javadoc/com/codename1/xml/XMLParser.html>]
- JSON DOM `Map` returned from `JSONParser` [<https://www.codenameone.com/javadoc/com/codename1/io/JSONParser.html>]

To use the expression processor when calling a webservice, you could use something like the following to parse JSON (notice this is interchangeable between JSON and XML):

```
Form hi = new Form("Location", new BoxLayout(BoxLayout.Y_AXIS));
hi.add("Pinpointing Location");
Display.getInstance().callSerially(() -> {
    Location l = Display.getInstance().getLocationManager().getCurrentLocationSync();
    ConnectionRequest request = new ConnectionRequest("http://maps.googleapis.com/maps/api/geocode/json", false) {
        private String country;
        private String region;
        private String city;
        private String json;

        @Override
        protected void readResponse(InputStream input) throws IOException {
            Result result = Result.fromContent(input, Result.JSON);
            country = result.getAsString("/results/address_components[types='country']/long_name");
            region =
result.getAsString("/results/address_components[types='administrative_area_level_1']/long_name");
            city = result.getAsString("/results/address_components[types='locality']/long_name");
            json = result.toString();
        }
    };

    @Override
    protected void postResponse() {
        hi.removeAll();
        hi.add(country);
        hi.add(region);
        hi.add(city);
        hi.add(new SpanLabel(json));
        hi.revalidate();
    }
};

request.setContentType("application/json");
request.addRequestHeader("Accept", "application/json");
request.addArgument("sensor", "true");
request.addArgument("latlng", l.getLatitude() + "," + l.getLongitude());

NetworkManager.getInstance().addToQueue(request);
});
hi.show();
[source,java]
```

The returned JSON looks something like this (notice it's snipped because the data is too long):

```
{  
  "status": "OK",  
  "results": [  
    {  
      "place_id": "ChIJJ5T9-iFawokRTPGaOginE04",  
      "formatted_address": "280 Broadway, New York, NY 10007, USA",  
      "address_components": [  
        {  
          "short_name": "280",  
          "types": ["street_number"],  
          "long_name": "280"  
        },  
        {  
          "short_name": "Broadway",  
          "types": ["route"],  
          "long_name": "Broadway"  
        },  
        {  
          "short_name": "Lower Manhattan",  
          "types": [  
            "neighborhood",  
            "political"  
          ],  
          "long_name": "Lower Manhattan"  
        },  
        {  
          "short_name": "Manhattan",  
          "types": [  
            "sublocality_level_1",  
            "sublocality",  
            "political"  
          ],  
          "long_name": "Manhattan"  
        },  
        {  
          "short_name": "New York",  
          "types": [  
            "locality",  
            "political"  
          ],  
          "long_name": "New York"  
        },  
        {  
          "short_name": "New York County",  
          "types": [  
            "administrative_area_level_2",  
            "political"  
          ],  
          "long_name": "New York County"  
        },  
      ]  
    }  
  ]  
}
```

```
{  
    "short_name": "NY",  
    "types": [  
        "administrative_area_level_1",  
        "political"  
    ],  
    "long_name": "New York"  
},  
{  
    "short_name": "US",  
    "types": [  
        "country",  
        "political"  
    ],  
    "long_name": "United States"  
},  
{  
    "short_name": "10007",  
    "types": ["postal_code"],  
    "long_name": "10007"  
},  
{  
    "short_name": "1868",  
    "types": ["postal_code_suffix"],  
    "long_name": "1868"  
}  
,  
]  
,  
"types": ["street_address"],  
"geometry": {  
    "viewport": {  
        "northeast": {  
            "lng": -74.0044642197085,  
            "lat": 40.7156470802915  
        },  
        "southwest": {  
            "lng": -74.0071621802915,  
            "lat": 40.7129491197085  
        }  
    },  
    "location_type": "ROOFTOP",  
    "location": {  
        "lng": -74.0058131999999,  
        "lat": 40.7142981  
    }  
}  
}  
/* SNIPED the rest */  
]  
}
```

Location
United States
New York
New York
{
"status": "OK",
"results": [
{
"place_id":
"ChIJ5T9-iFawokRTPGaOginEO4",
"formatted_address": "280 Broadway,
New York, NY 10007, USA",
"address_components": [
{
"short_name": "280",
"types": ["street_number"],
"long_name": "280"
},

Figure 363. Running the geocode sample above in the simulator

The XML processor currently handles global selections by using a double slash anywhere within the expression, for example:

```
// get all address_component names anywhere in the document with a type "political"
String array[] = result.getAsStringArray("//address_component[type='political']/long_name");

// get all types anywhere under the second result (dimension is 0-based)
String array[] = result.getAsStringArray("/result[1]//type");
```



Notice that Google's JSON webservice uses plural form for each of the node names in that API (ie. results, address_components, and types) where they don't in the XML services (ie result, address_component etc.)

Second Example

It also possible to do some more complex expressions. We'll use the following XML fragment for the next batch of examples:

```

<rankings type="aus" gender="male" date="2011-12-31">
    <player id="1036" coretennisid="6752" rank="1"
        delta="0" singlespoints="485000" doublespoints="675"
        deductedpoints="0" totalpoints="485675">
        <firstname>Bernard</firstname>
        <lastname>Tomic</lastname>
        <town>SOUTHPORT</town>
        <state>QLD</state>
        <dob>1992-10-21</dob>
    </player>
    <player id="2585" coretennisid="1500" rank="2"
        delta="0" singlespoints="313500" doublespoints="12630"
        deductedpoints="0" totalpoints="326130">
        <firstname>Mathew</firstname>
        <lastname>Ebden</lastname>
        <town>CHURCHLANDS</town>
        <state>WA</state>
        <dob>1987-11-26</dob>
    </player>
    <player id="6457" coretennisid="287" rank="3"
        delta="0" singlespoints="132500" doublespoints="1500"
        deductedpoints="0" totalpoints="134000">
        <firstname>Lleyton</firstname>
        <lastname>Hewitt</lastname>
        <town>EXETER</town>
        <state>SA</state>
        <dob>1981-02-24</dob>
    </player>
    <!-- ... etc ... -->
</rankings>

```

Above, if you want to select the IDs of all players that are ranked in the top 2, you can use an expression like:

```
int top2[] = result.getAsIntegerArray("//player[@rank < 3]/@id");
```



Notice above that the expression is using an attribute for selecting both rank and id. In JSON documents, if you attempt to select an attribute, it will look for a child node under the attribute name you ask for)

If a document is ordered, you might want to select nodes by their position, for example:

```
String first2[] = result.getAsStringArray("//player[position() < 3]/firstname");
String secondLast = result.getString("//player[last() - 1]/firstName");
```

It is also possible to select parent nodes, by using the '..' expression. For example:

```
int id = result.getAsInteger("//lastname[text()='Hewitt']/../@id");
```

Above, we globally find a lastname element with a value of ‘Hewitt’, then grab the parent node of lastname which happens to be the player node, then grab the id attribute from the player node. Alternatively, you could get the same result from the following simpler statement:

```
int id = result.getAsInteger("//player[lastname='Hewitt']/@id");
```

It is also possible to nest expressions, for example:

```
String id=result.getAsInteger("//player[//address[country/isocode='CA']]//@id");
```

In the above example, if the player node had an address object, we’d be selecting all players from Canada. This is a simple example of a nested expression, but they can get much more complex, which will be required as the documents themselves get more complex.

Moving on, to select a node based on the existence of an attribute:

```
int id[] = result.getAsIntegerArray("//player[@rank]//@id");
```

Above, we selected the IDs of all ranked players. Conversely, we can select the non-ranked players like this:

```
int id[] = result.getAsIntegerArray("//player[@rank=null]//@id");
```



Logical not (!) operators currently are not implemented)

You can also select by the existence of a child node

```
int id[] = result.getAsIntegerArray("//player[middlename]//@id");
```

Above, we selected all players that have a middle name.
 Keep in mind that the Codename One path expression language is not a full implementation of XPath 1.0, but does already handle many of the most useful features of the specification.

Properties Files

[Properties](https://www.codenameone.com/javadoc/com/codename1/io/Properties.html) [https://www.codenameone.com/javadoc/com/codename1/io/Properties.html] files are standard key/value pairs encoded into a text file. This file format is very familiar to Java developers and the Codename One specific version tries to be as close as possible to the original Java implementation.

Notice that properties file both in Java proper and in Codename One don’t support non-ascii characters. In order to encode unicode values into the properties file format you should use the [native2ascii](#) tool that ships with the JDK.

One major difference between standard Java properties and the ones in Codename One is that Codename One sorts properties alphabetically when saving. Java uses random order based on the **Hashtable** natural ordering.

This was done to provide consistency for saved files.

12.6. Debugging Network Connections



Figure 364. Debugging Network Connections

Codename One includes a Network Monitor tool which you can access via the simulator menu option. This tool reflects all the requests made through the connection requests and displays them in the left pane. This allows you to track issues in your code/web service and see everything that is "going through the wire".

This is a remarkably useful tool for optimizing and for figuring out what exactly is happening with your server connection logic.

12.6.1. Simpler Downloads

A very common task is file download to storage or filesystem.

The [Util](#) [<https://www.codenameone.com/javadoc/com/codename1/io/Util.html>] class has simple utility methods:

`downloadUrlToFileSystemInBackground`, `downloadUrlToStorageInBackground`, `downloadUrlToFile` & `downloadUrlToStorage`.

These all delegate to a feature in [ConnectionRequest](#) [<https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html>]:

```
ConnectionRequest.setDestinationStorage(fileName)  
ConnectionRequest.setDestinationFile(fileName);
```

Both of which simplify the whole process of downloading a file.

Downloading Images

Codename One has multiple ways to download an image and the general recommendation is the [URLImage](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html>]. However, the [URLImage](#) assumes that you know the size of the image in advance or that you are willing to resize it. In that regard it works great for some use cases but not so much for others.

The download methods mentioned above are great alternatives but they are a bit verbose when working with images and don't provide fine grained control over the [ConnectionRequest](#) e.g. making

a **POST** request to get an image.



Adding global headers is another use case but you can use `addDefaultHeader` [

To make this process simpler there is a set of helper methods in [ConnectionRequest that downloads images directly](#) [<https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html#downloadImageToStorage-java.lang.String-com.codename1.util.SuccessCallback->].

These methods complement the [Util](#) methods but go a bit further and feature very terse syntax e.g. you can just download a [ConnectionRequest](#) to [Storage](#) using code like this:

```
request.downloadImageToStorage(url, (img) -> theImageIsHereDoSomethingWithIt(img));
```

This effectively maps the [ConnectionRequest](#) directly to a [SuccessCallback](#) [<https://www.codenameone.com/javadoc/com/codename1/util/SuccessCallback.html>] for further processing.

URLImage Caching

[URLImage](#) is great. It changed the way we do some things in Codename One.

However, when we introduced it we didn't have support for the cache filesystem or for the JavaScript port. The cache filesystem is probably the best place for images of [URLImage](#) so supporting that as a target is a "no brainer" but JavaScript seems to work so why would it need a special case?

JavaScript already knows how to download and cache images from the web. [URLImage](#) is actually a step back from the things a good browser can do so why not use the native abilities of the browser when we are running there and fallback to using the cache filesystem if it's available and as a last resort go to storage...

That's exactly what the new method of [URLImage](#) does:

```
public static Image createCachedImage(String imageName, String url, Image placeholder, int resizeMode);
```

There are a few important things you need to notice about this method:

- It returns **Image** and not **URLImage**. This is crucial. Down casting to `URLImage` will work on the simulator but might fail in some platforms (e.g. JavaScript) so don't do that!
Since this is implemented natively in JavaScript we need a different abstraction for that platform.
- It doesn't support image adapters and instead uses a simplified resize rule. Image adapters work on [URLImage](#) since we have a lot of control in that class. However, in the browser our control is limited and so an adapter won't work.

If you do use this approach it would be far more efficient when running in the JavaScript port and will make better use of caching in most OS's.

12.7. Rest API

The **Rest** API makes it easy to invoke a restfull webservice without many of the nuances of **ConnectionRequest**. You can use it to define the HTTP method and start building based on that. So if I want to get a parsed JSON result from a URL you could do:

```
Map<String, Object> jsonData = Rest.get(myUrl).getAsJsonMap();
```

For a lot of REST requests this will fail because we need to add an HTTP header indicating that we accept JSON results. We have a special case support for that:

```
Map<String, Object> jsonData = Rest.get(myUrl).acceptJson().getAsJsonMap();
```

We can also do POST requests just as easily:

```
Map<String, Object> jsonData = Rest.post(myUrl).body(bodyValueAsString).getAsJsonMap();
```

Notice the usage of post and the body builder method. There are MANY methods in the builder class that cover pretty much everything you would expect and then some when it comes to the needs of rest services.

I changed the code in the kitchen sink webservice sample to use this API. I was able to make it shorter and more readable without sacrificing anything.

12.7.1. Rest in Practice - Twilio

The best way to explain the usage of this API is via a concrete "real world" example. Twilio provides many great telephony oriented webservices to developers. One of those is an SMS sending webservice which can be useful for things such as "device activation".

To get started you would need to signup to [Twilio](http://twilio.com/) [http://twilio.com/] and have the following 3 variable values:

```
String accountSID = "-----";  
String authToken = "-----";  
String fromPhone = "your Twilio phone number here";
```



You can open a trial Twilio account and it just tags all of your SMS's. Notice you would need to use a US based number if you don't want to pay

We can now send hello world as an SMS to the end user. Once this is in place sending an SMS via REST is just a matter of using the **Rest** API:

```
Response<Map> result = Rest.post("https://api.twilio.com/2010-04-01/Accounts/" + accountSID + "/Messages.json").  
    queryParam("To", destinationPhone).  
    queryParam("From", fromPhone).  
    queryParam("Body", "Hello World").  
    basicAuth(accountSID, authToken)).  
    getAsJsonMap();
```

Notice that this is equivalent of this "curl" command:

```
curl 'https://api.twilio.com/2010-04-01/Accounts/[accountSID]/Messages.json' -X POST \  
--data-urlencode 'To=[destinationPhone]' \  
--data-urlencode 'From=[fromPhone]' \  
--data-urlencode 'Body=Hello World' \  
-u [accountSID]:[AuthToken]
```

That's pretty cool as the curl command maps almost directly to the [Rest API call!](#)

What we do here is actually pretty trivial, we open a connection to the api messages URL. We add arguments to the body of the post request and define the basic authentication data.

The result is in JSON form we mostly ignore it since it isn't that important but it might be useful for error handling. This is a sample response (redacted keys):

```
{  
    "sid": "[sid value]",  
    "date_created": "Sat, 09 Sep 2017 19:47:30 +0000",  
    "date_updated": "Sat, 09 Sep 2017 19:47:30 +0000",  
    "date_sent": null,  
    "account_sid": "[sid value]",  
    "to": "[to phone number]",  
    "from": "[from phone number]",  
    "messaging_service_sid": null,  
    "body": "Sent from your Twilio trial account - Hello World",  
    "status": "queued",  
    "num_segments": "1",  
    "num_media": "0",  
    "direction": "outbound-api",  
    "api_version": "2010-04-01",  
    "price": null,  
    "price_unit": "USD",  
    "error_code": null,  
    "error_message": null,  
    "uri": "/2010-04-01/Accounts/[sid value]/Messages/SMe802d86b9f2246989c7c66e74b2d84ef.json",  
    "subresource_uris": {  
        "media": "/2010-04-01/Accounts/[sid value]/Messages/[message value]/Media.json"  
    }  
}
```

Notice the error message entry which is null meaning there was no error, if there was an error we'd have a message there or an error code that isn't in the 200-210 range.

This should display an error message to the user if there was a problem sending the SMS:

```
if(result.getResponseData() != null) {  
    String error = (String)result.getResponseData().get("error_message");  
    if(error != null) {  
        ToastBar.showErrorMessage(error);  
    }  
} else {  
    ToastBar.showErrorMessage("Error sending SMS: " + result.getResponseCode());  
}
```

12.8. Webservice Wizard

The Webservice Wizard can be invoked directly from the plugin. It generates stubs for the client side that allow performing simple method invocations on the server. It also generates a servlet that can be installed on any servlet container to intercept client side calls.

There are limits to the types of values that can be passed via the webservice wizard protocol but it is highly efficient since it is a binary protocol and very extensible thru object externalization. All methods are provided both as asynchronous and synchronous calls for the convenience of the developer.

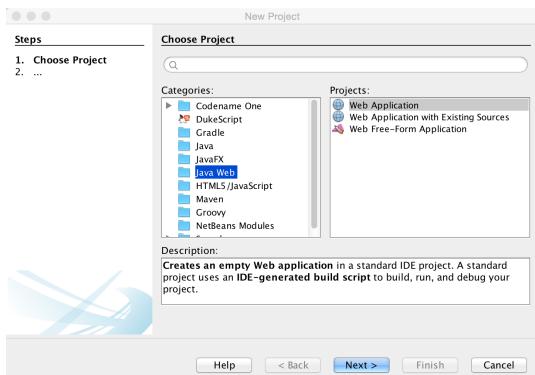


Figure 365. The first step in creating a client/server connection using the webservice wizard is to create a web application

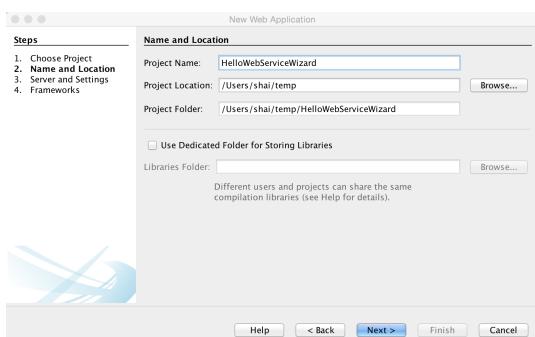


Figure 366. Any name will do

Normally you should have a server setup locally. I use Tomcat since it's really trivial and I don't really need much but there are many great Java web servers out there and this should work with all of them!

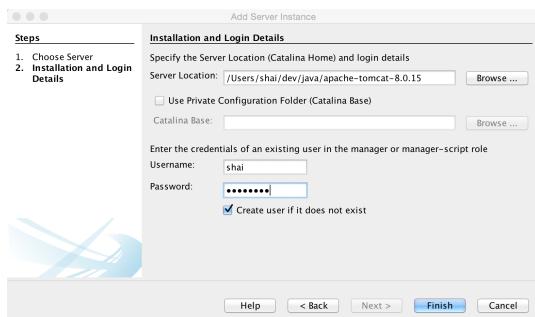


Figure 367. Setup your webserver in the IDE

Figure 368. Configure the application server to the newly created app, notice the application context value which we will use later

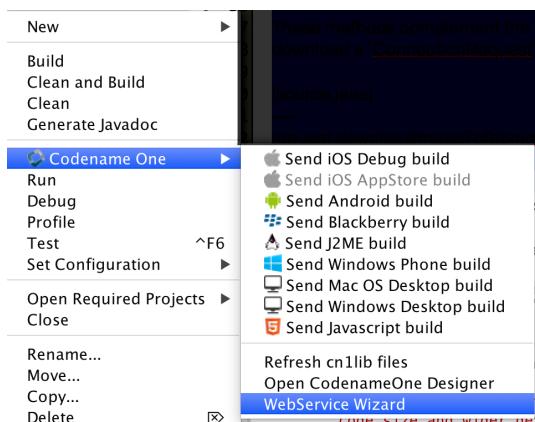


Figure 369. In the main Codename One project right click and select the WebService Wizard option

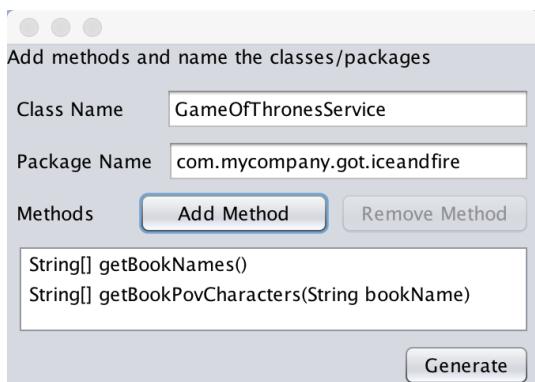


Figure 370. Set the package and class name for the webservice abstraction (notice this isn't your main class name) and then add the methods you want in the webservice



Figure 371. Add the methods and their arguments/return types. Once you finished adding all of those press the "Generate" button



The types of arguments are pretty limited however you can pass an arbitrary **Externalizable** object which can be "anything"

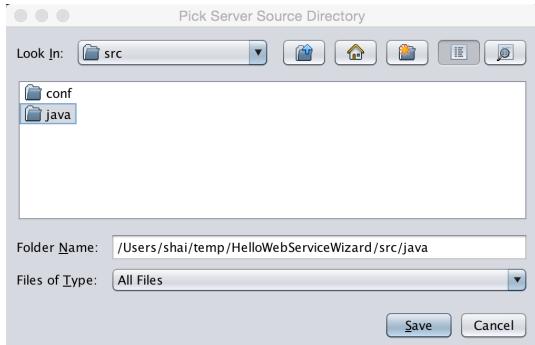


Figure 372. Pick the directory in the server project to which the source files will be generated by default this is the src/java directory under the project we created in the first step

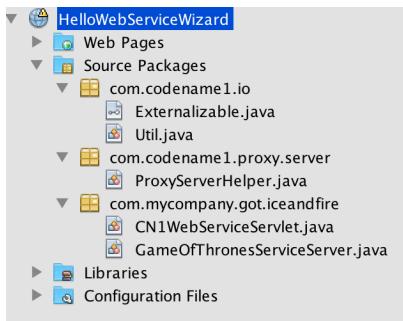


Figure 373. If you saved to the right location the server project directory should look like this

We can now open the **GameOfThronesServiceServer.java** file in the server and it looks like this:

```
public class GameOfThronesServiceServer {  
    public static String[] getBookNames() {  
        // your code goes here...  
        return null;  
    }  
  
    public static String[] getBookPovCharacters(String bookName) {  
        // your code goes here...  
        return null;  
    }  
}
```

All we need to do is fill in the code, for this example we'll only implement the first method for simplicity:

```

public class GameOfThronesServiceServer {
    public static String[] getBookNames() {
        return new String[] {
            "A Game of Thrones", "A Clash Of Kings", "A Storm Of Swords", "A Feast For Crows",
            "A Dance With Dragons", "The Winds of Winter", "A Dream of Spring"
        };
    }

    public static String[] getBookPovCharacters(String bookName) {
        // your code goes here...
        return null;
    }
}

```

Now lets open the client side code, in the `GameOfThronesService.java` file we see this

```

public class GameOfThronesService {
    private static final String DESTINATION_URL = "http://localhost:8080/cn1proxy";

    //...
}

```

The destination URL needs to point at the actual server which you will recall from the new project creation should include "HelloWebServiceWizard". So we can fix the URL to:

```
private static final String DESTINATION_URL = "http://localhost:8080/HelloWebServiceWizard/cn1proxy";
```

You would naturally need to update the host name of the server for running on a device otherwise the device would need to reside within your internal network and point to your IP address.

It is now time to write the actual client code that calls this. Every method we defined above is now defined as a static method within the `GameOfThronesService` class with two permutations. One is a synchronous permutation that behaves exactly as expected. It blocks the calling thread while calling the server and might throw an `IOException` if something failed.

This type of method (synchronous method) is very easy to work with since it's completely legal to call it from the event dispatch thread and it's very easy to map it to application logic flow.

The second type of method uses the async JavaScript style callbacks and accepts the callback interface. It returns immediately and doesn't throw any exception. It will call `onSuccess`/`onError` based on the server result.

You can pick either one of these approaches based on your personal preferences. Here we demonstrate both uses with the server API:

```

Form hi = new Form("WebService Wizard", new BoxLayout(BoxLayout.Y_AXIS));
Button getNamesSync = new Button("Get Names - Sync");
Button getNamesASync = new Button("Get Names - ASync");
hi.add(getNamesSync).add(getNamesASync);

getNamesSync.addActionListener((e) -> {
    try {
        String[] books = GameOfThronesService.getBookNames();
        hi.add("--- SYNC");
        for(String b : books) {
            hi.add(b);
        }
        hi.revalidate();
    } catch(IOException err) {
        Log.e(err);
    }
});

getNamesASync.addActionListener((e) -> {
    GameOfThronesService.getBookNamesAsync(new Callback<String[]>() {
        @Override
        public void onSucess(String[] value) {
            hi.add("--- ASYNC");
            for(String b : value) {
                hi.add(b);
            }
            hi.revalidate();
        }

        @Override
        public void onError(Object sender, Throwable err, int errorCode, String errorMessage) {
            Log.e(err);
        }
    });
});

```



Figure 374. The final result of the WebService Wizard code

12.9. Connection Request Caching

Caching server data locally is a huge part of the advantage a native app has over a web app.

Normally this is non-trivial as it requires a delicate balance especially if you want to test the server resource for changes.

HTTP provides two ways to do that the [ETag](https://en.wikipedia.org/wiki/HTTP_ETag) [https://en.wikipedia.org/wiki/HTTP_ETag] and [Last-Modified](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Last-Modified) [https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Last-Modified]. While both are great they are non-trivial to use and by no definition seamless.

We just added an experimental feature to connection request that allows you to set the caching mode to one of 4 states either globally or per connection request:

- **OFF** is the default meaning no caching.
- **SMART** means all get requests are cached intelligently and caching is "mostly" seamless
- **MANUAL** means that the developer is responsible for the actual caching but the system will not do a request on a resource that's already "fresh"
- **OFFLINE** will fetch data from the cache and won't try to go to the server. It will generate a 404 error if data isn't available

You can toggle these in the specific request by using `setCacheMode(CachingMode)` and set the global default using `setDefaultCacheMode(CachingMode)`.



Caching only applies to `GET` operations, it will not work for `POST` or other methods

There are several methods of interest to keep an eye for:

```
protected InputStream getCachedData() throws IOException;
protected void cacheUnmodified() throws IOException;
public void purgeCache();
public static void purgeCacheDirectory() throws IOException;
```

12.9.1. `getCachedData()`

This returns the cached data. This is invoked to implement `readResponse(InputStream)` when running offline or when we detect that the local cache isn't stale.

The smart mode implements this properly and will fetch the right data. However, the manual mode doesn't store the data and relies on you to do so. In that case you need to return the data you stored at this point and must implement this method for manual mode.

12.9.2. `cacheUnmodified()`

This is a callback that's invoked to indicate a cache hit, meaning that we already have the data.

The default implementation still tries to call all the pieces for compatibility (e.g. `readResponse`). However, if this is unnecessary you can override that method with a custom implementation or even a blank implementation to block such a case.

12.9.3. purgeCache & purgeCacheDirectory

These methods are pretty self explanatory. Notice one caveat though...

When you download a file or a storage element we don't cache them and rely on the file/storage element to be present and serve as "cache". When purging we won't delete a file or storage element you downloaded and thus these might remain.

However, we do remove the **ETag** and **Last-Modified** data so the files might get refreshed the next time around.

12.10. Cached Data Service

The [CachedDataService](https://www.codenameone.com/javadoc/com/codename1/io/services/CachedDataService.html) [https://www.codenameone.com/javadoc/com/codename1/io/services/CachedDataService.html] allows caching data and only updating it if the data changed on the server. Normally the download API's won't check for update if there is a local cache of the data (e.g. **URLImage** always uses the local copy). This isn't a bad thing, it's pretty efficient.

However, it might be important to update the image if it changed but we still want caching.

The **CachedDataService** will fetch data if it isn't cached locally and cache it. When you "refresh" it will send a special HTTP request that will only send back the data if it has been updated since the last refresh:

```
CachedDataService.register();
CachedData d = (CachedData)Storage.getInstance().readObject("LocallyCachedData");

if(d == null) {
    d = new CachedData();
    d.setUrl("http://....");
}

// check if there is a new version of this on the server
CachedDataService.updateData(d, new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        // invoked when/if the data arrives, we now have a fresh cache
        Storage.getInstance().writeObject("LocallyCachedData", d);
    }
});
```

12.11. Externalizable Objects

Codename One provides the [Externalizable](https://www.codenameone.com/javadoc/com/codename1/io/Externalizable.html) [https://www.codenameone.com/javadoc/com/codename1/io/Externalizable.html] interface, which is similar to the Java SE **Externalizable** interface. This interface allows an object to declare itself as **Externalizable** for serialization (so an object can be stored in a file/storage or sent over the network). However, due to the lack of reflection and use of obfuscation these objects must be registered with the [Util](https://www.codenameone.com/javadoc/com/codename1/io/Util.html) [https://www.codenameone.com/javadoc/com/codename1/io/Util.html] class.

Codename One doesn't support the Java SE Serialization API due to the size issues and complexities related to obfuscation.

The major objects that are supported by default in the Codename One `Externalizable` are: `String`, `Collection`, `Map`, `ArrayList`, `HashMap`, `Vector`, `Hashtable`, `Integer`, `Double`, `Float`, `Byte`, `Short`, `Long`, `Character`, `Boolean`, `Object[]`, `byte[]`, `int[]`, `float[]`, `long[]`, `double[]`.

Externalizing an object such as `h` below should work just fine:

```
Map<String, Object> h = new HashMap<>();
h.put("Hi", "World");
h.put("data", new byte[] {(byte)1});
Storage.getInstance().writeObject("Test", h);
```

However, notice that some things aren't polymorphic e.g. if we will externalize a `String[]` we will get back an `Object[]` since `String` arrays aren't detected by the implementation.



The externalization process caches objects so the app will seem to work and only fail on restart!

Implementing the `Externalizable` interface is only important when we want to store a proprietary object. In this case we must register the object with the `com.codename1.io.Util` class so the externalization algorithm will be able to recognize it by name by invoking:

```
Util.register("MyClass", MyClass.class);
```



You should do this early on in the app e.g. in the `init(Object)` but you shouldn't do it in a static initializer within the object as that might never be invoked!

An `Externalizable` object **must** have a **default public constructor** and must implement the following 4 methods:

```
public int getVersion();
public void externalize(DataOutputStream out) throws IOException;
public void internalize(int version, DataInputStream in) throws IOException;
public String getObjectId();
```

The `getVersion()` method returns the current version of the object allowing the stored data to change its structure in the future (the version is then passed when internalizing the object). The object id is a `String` uniquely representing the object; it usually corresponds to the class name (in the example above the Unique Name should be `MyClass`).



It's a common mistake to use `getClass().getName()` to implement `get ObjectId()` and it would **seem to work** in the simulator. This isn't the case though!

Since devices obfuscate the class names this becomes a problem as data is stored in a random name that changes with every release.

Developers need to write the data of the object in the externalize method using the methods in the data output stream and read the data of the object in the internalize method e.g.:

```
public void externalize(DataOutputStream out) throws IOException {
    out.writeUTF(name);
    if(value != null) {
        out.writeBoolean(true);
        out.writeUTF(value);
    } else {
        out.writeBoolean(false);
    }
    if(domain != null) {
        out.writeBoolean(true);
        out.writeUTF(domain);
    } else {
        out.writeBoolean(false);
    }
    out.writeLong(expires);
}

public void internalize(int version, DataInputStream in) throws IOException {
    name = in.readUTF();
    if(in.readBoolean()) {
        value = in.readUTF();
    }
    if(in.readBoolean()) {
        domain = in.readUTF();
    }
    expires = in.readLong();
}
```

Since strings might be null sometimes we also included convenience methods to implement such externalization. This effectively writes a boolean before writing the UTF to indicate whether the string is null:

```
public void externalize(DataOutputStream out) throws IOException {
    Util.writeUTF(name, out);
    Util.writeUTF(value, out);
    Util.writeUTF(domain, out);
    out.writeLong(expires);
}

public void internalize(int version, DataInputStream in) throws IOException {
    name = Util.readUTF(in);
    value = Util.readUTF(in);
    domain = Util.readUTF(in);
    expires = in.readLong();
}
```

Assuming we added a new date field to the object we can do the following. Notice that a **Date** is really a **long** value in Java that can be null. For completeness the full class is presented below:

```

public class MyClass implements Externalizable {
    private static final int VERSION = 2;
    private String name;
    private String value;
    private String domain;
    private Date date;
    private long expires;

    public MyClass() {}

    public int getVersion() {
        return VERSION;
    }

    public String getObjectId() {
        return "MyClass";
    }

    public void externalize(DataOutputStream out) throws IOException {
        Util.writeUTF(name, out);
        Util.writeUTF(value, out);
        Util.writeUTF(domain, out);
        if(date != null) {
            out.writeBoolean(true);
            out.writeLong(date.getTime());
        } else {
            out.writeBoolean(false);
        }
        out.writeLong(expires);
    }

    public void internalize(int version, DataInputStream in) throws IOException {
        name = Util.readUTF(in);
        value = Util.readUTF(in);
        domain = Util.readUTF(in);
        if(version > 1) {
            boolean hasDate = in.readBoolean();
            if(hasDate) {
                date = new Date(in.readLong());
            }
        }
        expires = in.readLong();
    }
}

```

Notice that we only need to check for compatibility during the reading process as the writing process always writes the latest version of the data.

12.12. UI Bindings & Utilities

Codename One provides several tools to simplify the path between networking/IO & GUI. A common task of showing a wait dialog or progress indication while fetching network data can be simplified by using the [InfiniteProgress](https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html) [https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html] class e.g.:

```
InfiniteProgress ip = new InfiniteProgress();
Dialog dlg = ip.showInifiniteBlocking();
request.setDisposeOnCompletion(dlg);
```

The process of showing a progress bar for a long IO operation such as downloading is automatically mapped to the IO stream in Codename One using the [SliderBridge](https://www.codenameone.com/javadoc/com/codename1/components/SliderBridge.html) [https://www.codenameone.com/javadoc/com/codename1/components/SliderBridge.html] class.



You can simulate network delays and disconnected network in the [Simulator](#) menu

The [SliderBridge](#) class can bind a [ConnectionRequest](#) to a [Slider](#) and effectively indicate the progress of the download. E.g.:

```
Form hi = new Form("Download Progress", new BorderLayout());
Slider progress = new Slider();
Button download = new Button("Download");
download.addActionListener((e) -> {
    ConnectionRequest cr = new ConnectionRequest("https://www.codenameone.com/img/blog/new_icon.png", false);
    SliderBridge.bindProgress(cr, progress);
    NetworkManager.getInstance().addToQueueAndWait(cr);
    if(cr.getResponseCode() == 200) {
        hi.add(BorderLayout.CENTER, new ScaleImageLabel(EncodedImage.create(cr.getResponseData())));
        hi.revalidate();
    }
});
hi.add(BorderLayout.SOUTH, progress).add(BorderLayout.NORTH, download);
hi.show();
```

Download Progre

Download

Figure 375. SliderBridge progress for downloading the image in the slow network mode

12.13. Logging & Crash Protection

Codename One includes a [Log](https://www.codenameone.com/javadoc/com/codename1/io/Log.html) [https://www.codenameone.com/javadoc/com/codename1/io/Log.html] API that allows developers to just invoke `Log.p(String)` or `Log.e(Throwable)` to log information to storage.

As part of the premium cloud features it is possible to invoke `Log.sendLog()` in order to email a log directly to the developer account. Codename One can do that seamlessly based on changes printed into the log or based on exceptions that are uncaught or logged e.g.:

```
Log.setReportingLevel(Log.REPORTING_DEBUG);
DefaultCrashReporter.init(true, 2);
```

This code will send a log every 2 minutes to your email if anything was changed. You can place it within the `init(Object)` method of your application.

For a production application you can use `Log.REPORTING_PRODUCTION` which will only email the log on exception.

12.14. Sockets

At this moment Codename One only supports TCP sockets. Server socket (`listen/accept`) is only available on Android and the simulator but not on iOS.

You can check if Sockets are supported using the `Socket.isSupported()`. You can test for server socket support using `Socket.isServerSocketSupported()`.

To use sockets you can use the `Socket.connect(String host, int port, SocketConnection eventCallback)` method.

To listen on sockets you can use the `Socket.listen(int port, Class scClass)` method which will instantiate a `SocketConnection` instance (scClass is expected to be a subclass of `SocketConnection`) for every incoming connection.

This simple example allows you to create a server and a client assuming the device supports both:

```
public class MyApplication {
    private Form current;

    public void init(Object context) {
        try {
            Resources theme = Resources.openLayered("/theme");
            UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()[0]));
        } catch(IOException e){
            e.printStackTrace();
        }
    }

    public void start() {
        if(current != null){
            current.show();
        }
    }
}
```

```

        return;
    }
    final Form soc = new Form("Socket Test");
    Button btn = new Button("Create Server");
    Button connect = new Button("Connect");
    final TextField host = new TextField("127.0.0.1");
    btn.addActionListener((evt) -> {
        soc.addComponent(new Label("Listening: " + Socket.getIP()));
        soc.revalidate();
        Socket.listen(5557, SocketListenerCallback.class);
    });
    connect.addActionListener((evt) -> {
        Socket.connect(host.getText(), 5557, new SocketConnection() {
            @Override
            public void connectionError(int errorCode, String message) {
                System.out.println("Error");
            }

            @Override
            public void connectionEstablished(InputStream is, OutputStream os) {
                try {
                    int counter = 1;
                    while(isConnected()) {
                        os.write(("Hi: " + counter).getBytes());
                        counter++;
                        Thread.sleep(2000);
                    }
                } catch(Exception err) {
                    err.printStackTrace();
                }
            }
        });
    });
    soc.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
    soc.addComponent(btn);
    soc.addComponent(connect);
    soc.addComponent(host);
    soc.show();
}

public static class SocketListenerCallback extends SocketConnection {
    private Label connectionLabel;

    @Override
    public void connectionError(int errorCode, String message) {
        System.out.println("Error");
    }

    private void updateLabel(final String t) {
        Display.getInstance().callSerially(new Runnable() {
            public void run() {
                if(connectionLabel == null) {
                    connectionLabel = new Label(t);
                    Display.getInstance().getCurrent().addComponent(connectionLabel);
                } else {
                    connectionLabel.setText(t);
                }
            }
        });
    }
}

```

```

        Display.getInstance().getCurrent().revalidate();
    }
}
}

@Override
public void connectionEstablished(InputStream is, OutputStream os) {
    try {
        byte[] buffer = new byte[8192];
        while(isConnected()) {
            int pending = is.available();
            if(pending > 0) {
                int size = is.read(buffer, 0, 8192);
                if(size == -1) {
                    return;
                }
                if(size > 0) {
                    updateLabel(new String(buffer, 0, size));
                }
            } else {
                Thread.sleep(50);
            }
        }
    } catch(Exception err) {
        err.printStackTrace();
    }
}

public void stop() {
    current = Display.getInstance().getCurrent();
}

public void destroy() {
}
}

```

12.15. Properties

In standard Java we usually have a POJO (Plain Old Java Object) which has getters/setters e.g. we can have a simple **Meeting** class like this:

```

public class Meeting {
    private Date when;
    private String subject;
    private int attendance;

    public Date getWhen() {
        return when;
    }

    public String getSubject() {
        return subject;
    }

    public int getAttendance() {
        return attendance;
    }

    public void setWhen(Date when) {
        this.when = when;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public void setAttendance(int attendance) {
        this.attendance = attendance;
    }
}

```

That's a classic POJO and it is the force that underlies JavaBeans and quite a few tools in Java.

The properties are effectively the getters/setters e.g. `subject`, `when` etc. but properties have several features that are crucial:

- They can be manipulated in runtime by a tool that had no knowledge of them during compile time
- They are observable - a tool can monitor changes to a value of a property
- They can have meta-data associated with them

These features are crucial since properties allow us all kinds of magic e.g. hibernate/ORM uses properties to bind Java objects to a database representation, jaxb does it to parse XML directly into Java objects and GUI builders use them to let us customize UI's visually.

POJO's don't support most of that so pretty much all Java based tools use a lot of reflection & bytecode manipulation. This works but has a lot of downsides e.g. say I want to map an object both to the Database and to XML/JSON.

Would the bytecode manipulation collide?

Would it result in duplicate efforts?

And how do I write custom generic code that uses such abilities? Do I need to manipulate the VM?

12.15.1. Properties in Java

These are all very abstract ideas, lets look at how we think properties should look in Java and how we can benefit from this moving forward.

This is the same class as the one above written with properties:

```
public class Meeting implements PropertyBusinessObject {  
    public final Property<Date,Meeting> when = new Property<>("when");  
    public final Property<String,Meeting> subject = new Property<>("subject");  
    public final Property<Integer,Meeting> attendance = new Property<>("attendance");  
    private final PropertyIndex idx = new PropertyIndex(this, "Meeting", when, subject, attendance);  
  
    @Override  
    public PropertyIndex getPropertyIndex() {  
        return idx;  
    }  
}
```

This looks a bit like a handful so let's start with usage which might clarify a few things then dig into the class itself.

When we used a POJO we did this:

```
Meeting meet = new Meeting();  
meet.setSubject("My Subject");  
Log.p(meet.getSubject());
```

With properties we do this:

```
Meeting meet = new Meeting();  
meet.subject.set("My Subject");  
Log.p(meet.subject.get());
```

Encapsulation

At first glance it looks like we just created public fields (which we did) but if you will look closely at the declaration you will notice the **final** keyword:

```
public final Property<String,Meeting> subject = new Property<>("subject");
```

This means that this code will not compile:

```
meet.subject = otherValue;
```

So all setting/getting must happen thru the set/get methods and they can be replaced. E.g. this is valid syntax that prevents setting the property to null and defaults it to an empty string:

```
public final Property<String,Meeting> subject = new Property<>("subject", "") {  
    public Meeting set(String value) {  
        if(value == null) {  
            return Meeting.this;  
        }  
        return super.set(value);  
    }  
};
```



We'll discuss the reason for returning the `Meeting` instance below

Introspection & Observability

Since `Property` is a common class it's pretty easy for introspective code to manipulate properties. However, it can't detect properties in an object without reflection.

That's why we have the `index` object and the `PropertyBusinessObject` interface (which defines `getPropertyIndex`).

The `PropertyIndex` class provides meta data for the surrounding class including the list of the properties within. It allows enumerating the properties and iterating over them making them accessible to all tools.

Furthermore all properties are observable with the property change listener. I can just write this to instantly print out any change made to the property:

```
meet.subject.addChangeListener((p) -> Log.p("New property value is: " + p.get()));
```

12.15.2. The Cool Stuff

That's the simple stuff that can be done with properties, but they can do **much** more!

For starters all the common methods of `Object` can be implemented with almost no code:

```

public class Meeting implements PropertyBusinessObject {
    public final Property<Date,Meeting> when = new Property<>("when");
    public final Property<String,Meeting> subject = new Property<>("subject");
    public final Property<Integer,Meeting> attendance = new Property<>("attendance");
    private final PropertyIndex idx = new PropertyIndex(this, "Meeting", when, subject, attendance);

    @Override
    public PropertyIndex getPropertyIndex() {
        return idx;
    }

    public String toString() {
        return idx.toString();
    }

    @Override
    public boolean equals(Object obj) {
        return obj.getClass() == getClass() && idx.equals(((TodoTask)obj).getPropertyIndex());
    }

    @Override
    public int hashCode() {
        return idx.hashCode();
    }
}

```

This is easy thanks to introspection...

We already have some simple code that can convert an object to/from JSON Maps e.g. this can fill the property values from parsed JSON:

```
meet.getPropertyIndex().populateFromMap(jsonParsedData);
```

And visa versa:

```
String jsonString = meet.toJSON();
```

We also have a very simple ORM solution that maps values to table columns and can create tables. It's no hibernate but sqlite isn't exactly big iron so it might be good enough.

Constructors

One of the problematic issues with constructors is that any change starts propagating everywhere. If I have fields in the constructor and I add a new field later I need to keep the old constructor for compatibility.

So we added a new syntax:

```
Meeting meet = new Meeting();
subject.set("My Subject");
when.set(new Date());
```

That is why every property in the definition needed the **Meeting** generic and the set method returns the **Meeting** instance...

We are pretty conflicted on this feature and are thinking about removing it.

Without this feature the code would look like this:

```
Meeting meet = new Meeting();
meet.subject.set("My Subject");
meet.when.set(new Date());
```

Seamless Serialization

Lets assume I have an object called **Contacts** which includes contact information of contact e.g.:

```
public class Contact implements PropertyBusinessObject {
    public final IntProperty<Contact> id = new IntProperty<>("id");
    public final Property<String, Contact> name = new Property<>("name");
    public final Property<String, Contact> email = new Property<>("email");
    public final Property<String, Contact> phone = new Property<>("phone");
    public final Property<Date, Contact> dateOfBirth = new Property<>("dateOfBirth", Date.class);
    public final Property<String, Contact> gender = new Property<>("gender");
    public final IntProperty<Contact> rank = new IntProperty<>("rank");
    public final PropertyIndex idx = new PropertyIndex(this, "Contact", id, name, email, phone, dateOfBirth, gender,
    rank);

    @Override
    public PropertyIndex getPropertyIndex() {
        return idx;
    }

    public Contact() {
        name.setLabel("Name");
        email.setLabel("E-Mail");
        phone.setLabel("Phone");
        dateOfBirth.setLabel("Date Of Birth");
        gender.setLabel("Gender");
        rank.setLabel("Rank");
    }
}
```

Standard Java Objects can be serialized in Codename One by implementing the Codename One **Externalizable** interface. You also need to register the **Externalizable** object so the implementation will be aware of it. Codename One business objects are seamlessly **Externalizable** and you just need to register them.

E.g. you can do something like this in your **init(Object)** method:

```
new Contact().getPropertyIndex().registerExternalizable();
```

After you do that once you can write/read contacts from storage if you so desire:

```
Storage.getInstance().writeObject("MyContact", contact);  
Contact readContact = (Contact)Storage.getInstance().readObject("MyContact");
```

This will obviously also work for things like `List<Contact>` etc...

Seamless SQL Storage

Writing SQL code can be tedious. Which is why `SQLMap` is such an important API for some of us. `SQLMap` allows CRUD (Create, Read, Update, Delete) operations on the builtin SQLite database using property objects.

If we continue the example from above to show persistence to the SQL database we can just do something like this:

```
private Database db;  
private SQLMap sm;  
public void init(Object context) {  
    theme = UIManager.initFirstTheme("/theme");  
    Toolbar.setGlobalToolbar(true);  
    Log.bindCrashProtection(true);  
  
    try {  
        Contact c = new Contact();  
        db = Display.getInstance().openOrCreate("propertiesdemo.db"); ①  
        sm = SQLMap.create(db); ②  
        sm.setPrimaryKeyAutoIncrement(c, c.id); ③  
        sm.createTable(c); ④  
    } catch(IOException err) {  
        Log.e(err);  
    }  
}
```

In the above code we do the following:

- ① Create or open an SQLite database using the standard syntax
- ② Create a properties binding instance
- ③ Define the primary key for contact as `id` and set it to `auto increment` which will give it a unique value from the database
- ④ Call SQL's `createTable` if the table doesn't exist yet!



Notice that at this time altering a created table isn't possible so if you add a new property you might need to detect that and do an `alter` call manually

We can then add entries to the contact table using:

```
sm.insert(myContact);
```

We can update an entry using:

```
sm.update(myContact);
```

And delete an entry using:

```
sm.delete(myContact);
```

Listing the entries is more interesting:

```
List<PropertyBusinessObject> contacts = sm.select(c, c.name, true, 1000, 0);

for(PropertyBusinessObject cc : contacts) {
    Contact currentContact = (Contact)cc;

    // ...
}
```

The arguments for the `select` method are:

- The object type
- The attribute by which we want to sort the result (can be null)
- Whether sort is ascending
- Number of elements to fetch
- Page to start with - in this case if we have more than 1000 elements we can fetch the next page using `sm.select(c, c.name, true, 1000, 1)`

There are many additional configurations where we can fine tune how a specific property maps to a column etc.

What's Still Missing

The `SQLMap` API is very simplistic and doesn't try to be Hibernate/JPA for mobile. So basic things aren't available at this time and just won't work. This isn't necessarily a problem as mobile databases don't need to be as powerful as server databases.

Relational Mappings/JOIN

Right now we can't map an object to another object in the database with the typical one-many, one-one etc. relationships that would could do with JPA. The [SQLMap](#) API is really simplistic and isn't suited for that level of mapping at this time.

If there is demand for this it's something we might add moving forward but our goal isn't to re-invent hibernate.

Threading

SQLite is sensitive to threading issues especially on iOS. We mostly ignored the issue of threading and issue all calls in process. This can be a problem for larger data sets as the calls would usually go on the EDT.

This is something we might want to fix for the generic SQLite API so low level SQL queries will work with our mapping in a sensible way.

Alter

Right now we don't support table altering to support updated schemas. This is doable and shouldn't be too hard to implement correctly so if there is demand for doing it we'll probably add support for this.

Complex SQL/Transactions

We ignored functions, joins, transactions and a lot of other SQL capabilities.

You can use SQL directly to use all of these capabilities e.g. if you begin a transaction before inserting/updating or deleting this will work as advertised however if a rollback occurs our mapping will be unaware of that so you will need to re-fetch the data.

You will notice we mapped auto-increment so we will generally try to map things that make sense for various use cases, if you have such a use case we'd appreciate pull requests and feedback on the implementation.

Caching/Collision

As mentioned above, we don't cache anything and there might be a collision if you select the same object twice you will get two separate instances that might collide if you update both (one will "win").

That means you need to pay attention to the way you cache objects to avoid a case of a modified version of an object kept with an older version.

Preferences Binding

Some objects make sense as global objects, we can just use the [Preferences](#) API to store that data directly but then we don't have the type safety that property objects bring to the table. That's where the binding of property objects to preferences makes sense. E.g. say we have a global [Settings](#) property object we can just bind it to preferences using:

```
PreferencesObject.create(settingsInstance).bind();
```

So if settings has a property called `companyName` it would bind into `Preferences` under the `Settings.companyName` entry.

We can do some more elaborate bindings such as:

```
PreferencesObject.create(settingsInstance).
    setPrefix("MySettings-").
    setName(settingsInstance.companyName, "company").
    bind();
```

This would customize all entry keys to start with `MySettings-` instead of `Settings..`. This would also set the company name entry to `company` so in this case instead of `Settings.companyName` we'd have `MySettings-company`.

UI Binding

One of the bigger features of properties are their ability to bind UI to a property. E.g. if we continue the sample above with the `Contact` class let's say we have a text field on the form and we want the property (which we mapped to the database) to have the value of the text field. We could do something like this:

```
myNameTextField.setText(myNameTextField.getText());
myNameTextField.addActionListener(e -> myContact.name.set(myNameTextField.getText()));
```

That would work nicely but what if we changed the property value, that wouldn't be reflected back into the text field?

Also that works nicely for text field but what about other types e.g. numbers, check boxes, pickers etc. this becomes a bit more tedious with those.

Binding makes this all seamless. E.g. the code above can be written as:

```
UiBinding uib = new UiBinding();
uib.bind(myNameTextField, myContact.name);
```

The cool thing is that this works with multiple component types and property types almost magically. Binding works by using an adapter class to convert the data to/from the component. The adapter itself works with a generic converter e.g. this code:

```
uib.bind(myRankTextField, myContact.rank);
```

Seems similar to the one above but it takes a String that is returned by the text field and seamlessly

converts it to the integer needed by rank. This also works in the other direction...

We can easily build a UI that would allow us to edit the `Contact` property in memory:

```
Container resp = new Container(BoxLayout.y());
UiBinding uib = new UiBinding();

TextField nameTf = new TextField();
uib.bind(c.name, nameTf);
resp.add(c.name.getLabel()). ①
    add(nameTf);

TextField emailTf = new TextField();
emailTf.setConstraint(TextField.EMAILADDR);
uib.bind(c.email, emailTf);
resp.add(c.email.getLabel()). ②
    add(emailTf);

TextField phoneTf = new TextField();
phoneTf.setConstraint(TextField.PHONENUMBER);
uib.bind(c.phone, phoneTf);
resp.add(c.phone.getLabel()). ③
    add(phoneTf);

Picker dateOfBirth = new Picker();
dateOfBirth.setType(Display.PICKER_TYPE_DATE); ④
uib.bind(c.dateOfBirth, dateOfBirth);
resp.add(c.dateOfBirth.getLabel()). ⑤
    add(dateOfBirth);

ButtonGroup genderGroup = new ButtonGroup();
RadioButton male = RadioButton.createToggle("Male", genderGroup);
RadioButton female = RadioButton.createToggle("Female", genderGroup);
RadioButton undefined = RadioButton.createToggle("Undefined", genderGroup);
uib.bindGroup(c.gender, new String[] {"M", "F", "U"}, male, female, undefined); ⑥
resp.add(c.gender.getLabel()). ⑦
    add(GridLayout.encloseIn(3, male, female, undefined));

TextField rankTf = new TextField();
rankTf.setConstraint(TextField.NUMERIC);
uib.bind(c.rank, rankTf); ⑧
resp.add(c.rank.getLabel()). ⑨
    add(rankTf);
```

① Notice I use the label of the property which allows better encapsulation

② We can bind picker seamlessly

③ We can bind multiple radio buttons to a single property to allow the user to select the gender, notice that labels and values can be different e.g. "Male" selection will translate to "M" as the value

④ Numeric bindings "just work"

The screenshot shows a user interface for a contact properties form. At the top are two small blue navigation icons. Below them is a table with rows for Name, E-Mail, Phone, Date Of Birth, Gender, and Rank. Each row has a label on the left and a corresponding input field on the right. The Gender row contains three radio buttons labeled 'Male', 'Female', and 'Undefined'. The Rank row contains a single input field.

Figure 376. Properties form for the contact

Binding Object & Auto Commit

We skipped a couple of fact about the `bind()` method. It has an additional version that accepts a `ComponentAdapter` which allows you to adapt the binding to any custom 3rd party component. That's a bit advanced for now but I might discuss this later.

However, the big thing I "skipped" was the return value... `bind` returns a `UiBinding.Binding` object when performing the bind. This object allows us to manipulate aspects of the binding specifically unbind a component and also manipulate auto commit for a specific binding.

Auto commit determines if a property is changed instantly or on `commit`. This is useful for a case where we have an "OK" button and want the changes to the UI to update the properties only when "OK" is pressed (this might not matter if you keep different instances of the object). When auto-commit is on (the default which you can change via `setAutoCommit` in the `UiBinding`) changes reflect instantly, when it's off you need to explicitly call `commit()` or `rollback()` on the `Binding` class.

`commit()` applies the changes in the UI to the properties, `rollback()` restores the UI to the values from the properties object (useful for a "reset changes" button).

Binding also includes the ability to "unbind" this is important if you have a global object that's bound to a UI that's discarded. Binding might hold a hard reference to the UI and the property object might create a memory leak.

By using the `disconnect()` method in `Binding` we can separate the UI from the object and allow the GC to cleanup.

UI Generation

Up until now this was pretty cool but if you looked at the UI construction code above you would see that it's pretty full of boilerplate code. The thing about boilerplate is that it shows where automation can be applied, that's the exact idea behind the magical "InstantUI" class. This means that the UI above can be generated using this code:

```

InstantUI iui = new InstantUI();
iui.excludeProperty(myContact.id); ①
iui.setMultiChoiceLabels(myContact.gender, "Male", "Female", "Undefined"); ②
iui.setMultiChoiceValues(myContact.gender, "M", "F", "U");
Container cnt = iui.createEditUI(myContact, true); ③

```

- ① The id property is useful for database storage but we want to exclude it from the UI
- ② This implements the `gender` toggle button selection, we provide a hint to the UI so labels and values differ
- ③ We create the UI from the screenshot above with one line and it's seamlessly bound to the properties of myContact. The second argument indicates the "auto commit" status.

This still carries most of the flexibilities of the regular binding e.g. I can still get a binding object using:

```
UiBinding.Binding b = iui.getBindings(cnt);
```

You might not have noticed this but in the previous vbose code we had lines like:

```
emailTf.setConstraint(TextField.EMAILADDR);
```

You might be surprised to know that this will still work seamlessly without doing anything, as would the picker component used to pick a date...

The picker component implicitly works for date type properties, numeric constraints and numbers are implicitly used for number properties and check boxes are used for booleans.

But how do we know to use an email constraint for the email property?

We have some special case defaults for some common property names, so if your property is named email it will use an email constraint by default. If it's named url or password etc. it will do the "right thing" unless you explicitly state otherwise. You can customize the constraint for a specific property using something like:

```
iui.setTextFieldConstraint(contact.email, TextArea.ANY);
```

This will override the defaults we have in place. The goal of this tool is to have sensible "magical" defaults that "just work".

[\[9\] Event Dispatch Thread](#)

13. Push Notifications

This chapter discusses push support in Codename One applications. It covers how to set up push on the various platforms, how to respond to push notifications in your app, and how to send push notifications to your app.



Push notifications require a Codename One Pro account or higher. You must register your app to receive push notification using `Display.getInstance().registerPush();`



For a quick reference on setting up Push notifications, check out the [Push Cheatsheet](#) [<https://www.codenameone.com/files/push-cheatsheet.pdf>].

13.1. Understanding Push Notifications

Push notifications provide a way to inform users that something of interest has taken place. It is one of the few mechanisms available to interact with the user **even when the app isn't running**. If your app is registered to receive push notifications, then you can send messages to the user's device via REST API either from another device, or, as is usually the case, from a web server.

Messages may contain a short title and text body that will be displayed to the user in their device's messages stream. They may also specify a badge to display on the app's icon (e.g. that red circle on your mail app icon that indicates how many unread messages you have), a sound to play when the message arrives, an image attachment, and a set of "actions" that the user can perform directly in the push notification.

In addition to messages that the user **sees**, a push notification can contain non-visual information that is **silently** sent to your app.

13.2. Implementing Push Support

Enabling push support in your application is just a matter of implementing the [PushCallback](#) interface in your application's main class (i.e. the class that includes your `start()`, `init()`, etc... methods.

```

public class MyApplication implements PushCallback {

    // ....

    /**
     * Invoked when the push notification occurs
     *
     * @param value the value of the push notification
     */
    public void push(String value) {
        System.out.println("Received push message: "+value);
    }

    /**
     * Invoked when push registration is complete to pass the device ID to the application.
     *
     * @param deviceId OS native push id you should not use this value and instead use <code>Push.getPushKey()</code>
     * @see Push#getPushKey()
     */
    public void registeredForPush(String deviceId) {
        System.out.println("The Push ID for this device is "+Push.getPushKey());
    }

    /**
     * Invoked to indicate an error occurred during registration for push notification
     * @param error descriptive error string
     * @param errorCode an error code
     */
    public void pushRegistrationError(String error, int errorCode) {
        System.out.println("An error occurred during push registration.");
    }
}

```

There will be additional steps required to deploy to each platform (e.g. iOS requires you to generate push certificates, Android needs you to register your app ID with their cloud messaging platform, etc...), but, fundamentally, this is all that is required to enable push support in your app.

13.3. The Push Lifecycle

Let's take minute to go over the three callbacks in our application.

13.3.1. Registration

When your application first opens, it needs to register with the platform's central cloud messaging infrastructure. On Android this involves a call to their GCM/FCM server; on iOS, the call will be to the APNS server, on Windows (UWP) the call will be to the WNS server. And so on. That server will return a unique device ID that can be used to send push notifications to the device. This device ID will then be passed to your `registeredForPush()` method as the `deviceId` parameter so that you can save it somewhere. Typically you would send this value to your own web service so that you can use it to send notifications to the device later on. The device ID will generally not change unless you uninstall and reinstall the app, but you will receive the callback every time the app starts.



The `deviceId` parameter cannot be used directly when sending push messages via the Codename One push service. It needs to be prefixed with a platform identifier so the that push server knows which messaging service to route the message through. You can obtain the complete device ID, including the platform prefix, by calling `Push.getPushKey()`

If the registration failed for some reason, the the `pushRegistrationError()` callback will be fired instead.

Notice that all of this happens seamlessly behind the scenes when your app loads. You don't need to initiate any of this workflow.

13.3.2. Sending a Push Notification

Codename One provides a secure REST API for sending push notifications. As an HTTP API, it is language agnostic. You can send push notifications to your app using Java, PHP, Python, Ruby, or even by hand using something like curl. Each HTTP request can contain a push message and a list of device IDs to which the message should be sent. You don't need to worry about whether your app is running on Android, iOS, Windows, or the web. A single HTTP request can send a message to many devices at once.

13.3.3. Receiving a Push Notification

There are two different scenarios to be aware of when it comes to receiving push notifications. If your app is running in the foreground when the message arrives, then it will be passed directly to your `push()` callback. If your app is either in the background, or not running, then the notification will be displayed in your device's notifications. If the user then taps the notification, it will open your app, and the `push()` callback will be run with the contents of the message.

Some push message types include hidden content that will not be displayed in your device's notifications. These hidden messages (or portions of messages) are passed directly to the `push()` callback of your app for processing.



On iOS, hidden push messages (push type 2) will not be delivered when the app is in the background.



You can set the `android.background_push_handling` build hint to "true" to deliver push messages on Android when the app is minimized (running in the background). There is no equivalent setting on other platforms currently.

13.4. Testing Push Support

The easiest way to test push notifications in your app is to use the push simulation feature of the Codename One Simulator.

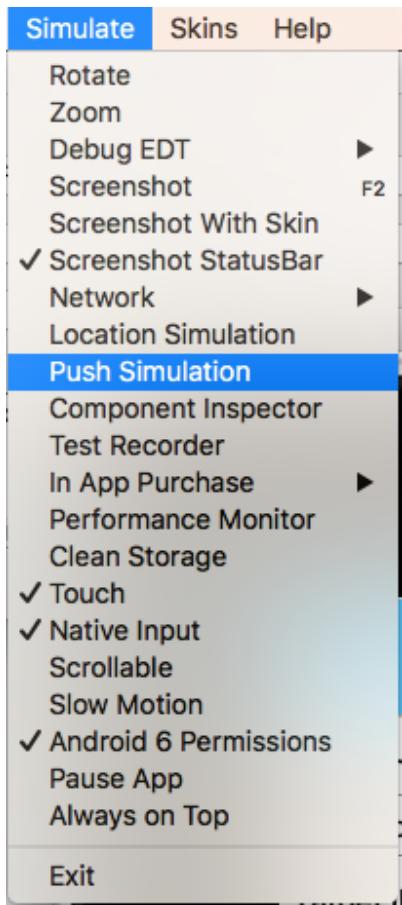


Figure 377. The "Push Simulation" menu item in the simulator opens the push simulator tool.

The "Registered Successfully" button will trigger your app's `registeredForPush()` method, and the "Registration Error" button will trigger your app's `pushRegistrationError()` method.

The "Send" button will trigger the `push()` callback with the message body that you place in the "Send Message" field.

The "Push Type" drop-down allows you to select the "type" of the push message. This dictates how the message body (i.e. the contents of the "Send Message" field) is interpreted. Some push types simply pass the message to the device verbatim, while others assume that the message contains structure that is meant to be parsed by the client to extract such things as badges, sounds, images, and actions that are associated with the message. We'll go over the available push types in a moment, but for now, we'll keep it simple by just using a push type of "1" - which just sends the message verbatim.

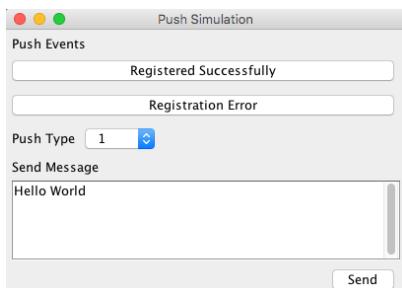


Figure 378. Sending a basic hello world push from the push simulator

Let's try a simple "hello world" push message. Select "1" from the "Push Type" drop-down menu, and enter "Hello World" into the "Send Message" field as shown above. Then press "send".

Assuming your `push()` method looks like:

```
public void push(String value) {  
    System.out.println("Received push message: "+value);  
}
```

You should see the following output in your console:

```
Received push message: Hello World
```

This experiment simulated a push notification while the app is running in the foreground. Now let's simulate the case where the app is not running, or running in the background. We can simulate this by pausing the app. Return to the Codename One simulator window, and select "Pause App" from the "Simulate" menu as shown below.

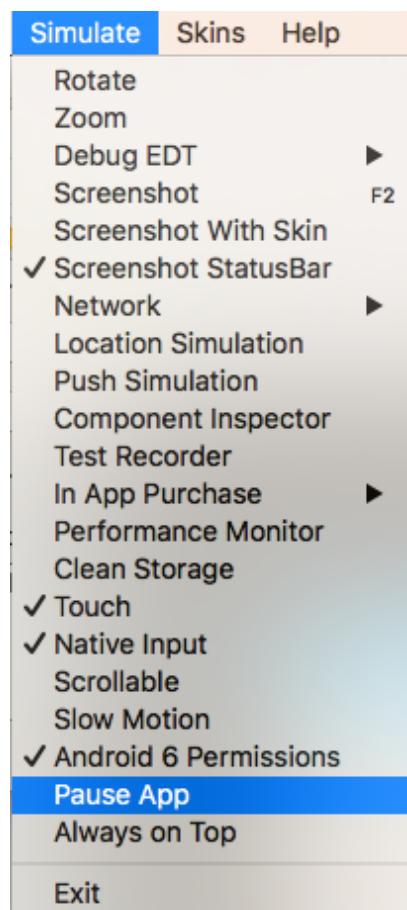


Figure 379. Pausing the app in the simulator so we can simulate push notifications while app is in the background.

When the app is paused it will simply display a white screen in the simulator with the text "Paused" in the middle.

Now return to the push simulator again, and press "Send" again with same values in the other fields (Push type 1, and Message "Hello World"). Rather than running the `push()` callback this time, it will display a popup dialog outside the app, as shown below.

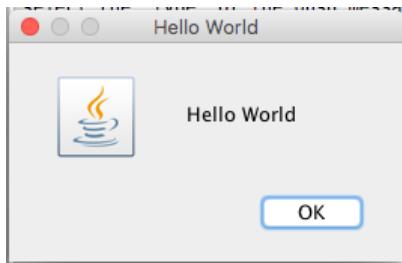


Figure 380. Push message causes a popup dialog in the simulator when the app is paused.

While this popup dialog doesn't replicate what a push notification will look like in a device's notifications stream when the app is closed, it does simulate the conceptual workflow. The process whereby the user is notified of the message outside of the app, and the app is not notified until/unless the user taps on the notification.

If you monitor the console for your app, you should notice that the `push()` callback hasn't been called yet for this notification, but if you click "OK" in the dialog, your `push()` callback will be run. Clicking OK is analogous to the user tapping on the notification. If you simply close the dialog box (by clicking the "x" in the corner), this would be analogous to the user dismissing the notification. In this case the `push()` callback would not be called at all.

13.5. Push Types and Message Structure

The example above sends a simple message to be displayed to the user. Push notifications can include more data than just an alert message, though. When the selected "push type" is 0 or 1, the push server will interpret the message body a simple alert string. Selecting a different push type will affect how the message body is interpreted. The following push types are supported:

- 0, 1 - The default push types, they work everywhere and present the string as the push alert to the user
- 2 - hidden, non-visual push. This won't show any visual indicator on any OS! In Android this will trigger the `push(String)` call with the message body. In iOS this will only happen if the application is in the foreground otherwise the push will be lost
- 3 - 1 + 2 = 3 allows combining a visual push with a non-visual portion. Expects a message in the form: `This is what the user won't see;This is something he will see`. E.g. you can bundle a special ID or even a JSON string in the hidden part while including a friendly message in the visual part.
When active this will trigger the `push(String)` method twice, once with the visual and once with the hidden data.
- 4 - Allows splitting a visual push request based on the format `title;body` to provide better visual representation in some OS's.
- 5 - Sends a regular push message but doesn't play a sound when the push arrives
- 99 - The message body is expected to be XML, where the root element contains at least `type` and `body` attributes which correspond to one of the other push push types and message body respectively. This push type supports additional information such as image attachments and push actions. E.g. `<push type="1" body="Hello World"/>`
- 100 - Applicable only to iOS and Windows. Allows setting the numeric badge on the icon to the

given number. The body of the message must be a number e.g. unread count.

- 101 - identical to 100 with an added message payload separated with a space. E.g. 30 You have 30 unread messages will set the badge to "30" and present the push notification text of "You have 30 unread messages". Supported on Android, iOS, and Windows.

The following sections will show examples of the various kinds of pushes. You can try them out yourself by opening the push simulator.

13.5.1. Example Push Type 1

Push Type 1; Message Body: "Hello World"

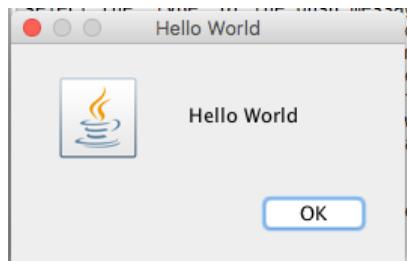


Figure 381. Push type 1 "Hello World" message in simulator.



Figure 382. Push type 1 "Hello World" message in Android when app is in background.

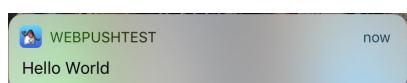


Figure 383. Push type 1 "Hello World" message in iOS when app is in background.

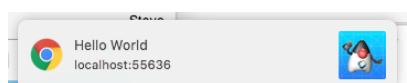


Figure 384. Push type 1 "Hello World" message in Chrome desktop.

In all cases, if the user taps/clicks the notification, it will bring the app to the foreground and call the `push()` callback with "Hello World" as the argument.

13.5.2. Example Push Type 2

Push Type 2; Message Body: "Hello World"

Push type 2 is a hidden push so it will behave differently on different platforms. On Android, the `push()` callback will be fired even if the app is in the background. On iOS, it will simply be ignored if the app is in the background.

If the app is in the foreground, this will trigger the `push()` callback with "Hello World" as the argument.



You can determine the type of push that has been received in your `push()` callback by calling `Display.getInstance().getProperty("pushType")`. This will return a String of the push type. E.g. in this case `Display.getInstance().getProperty("pushType")` will return "2".

13.5.3. Example Push Type 3

Push Type 3; Message Body `Hello World;{"from":"Jim", "content":"Hello World"}`

Push type 3 combines an alert message with some hidden content that the user won't see. In the example above, the alert message is "Hello World" and the hidden content is a JSON string that will be passed to our app to be parsed.

If the app is in the background, then the alert message will be posted to the user's notifications. See "Example Push Type 1" above as this message will be identical.

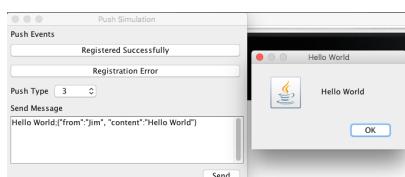


Figure 385. Push type 3 shows only the alert message (the portion before the first ";").

This push will result in our `push()` callback being fired twice; once with the alert message, and once with the hidden content. When it is fired with the alert message, `Display.getInstance().getProperty("pushType")` will report a type of "1". When it is fired with the JSON hidden content, it will report a push type of "2".

13.5.4. Example Push Type 4

Push Type 4; Message Body "Hello World;I'm just saying hello"

Push type 4 specifies a title and a message body. In this example, alert title will be "Hello World", and the body will be "I'm just saying hello".

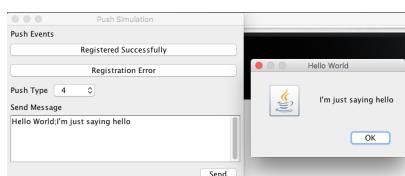


Figure 386. Push type 4 "Hello World" message in simulator.

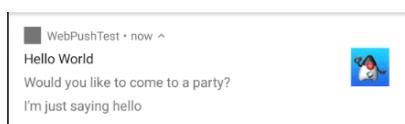


Figure 387. Push type 4 "Hello World" message in Android when app is in background.

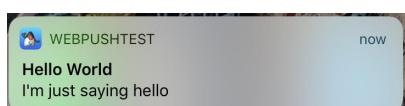


Figure 388. Push type 4 "Hello World" message in iOS when app is in background.

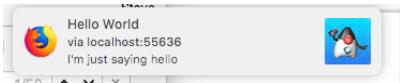


Figure 389. Push type 4 "Hello World" message in Firefox desktop.

With this push type, the `push()` callback will be fired only if the user taps/opens the notification, and the argument will contain the entire message ("Title;Body").



On some platforms, the argument of the `push()` callback will only include the "body" portion of the payload, and in other platforms it will include the full "Title;Body" payload.

13.5.5. Example Push Type 5

Push Type 5; Message Body "Hello World"

Push type 5 will behave identically to push type 1, except that the notification won't make any sound on the device. On some platforms, `Display.getInstance().getProperty("pushType")` will report a push type of "1", when it receives a push of type 5.

13.5.6. Example Push Type 100

Push Type 100; Message Body "5"

Push type 100 just expects an integer in the message body. This is interpreted as the badge that should be set on the app. This is currently only supported on Windows and iOS.



Figure 390. Push type 100 on iOS, setting the badge to "5"

Push type 100 should not trigger the `push()` callback.

13.5.7. Example Push Type 101

Push Type 101; Message Body "5 Hello World"

Push type 101 combines a badge with an alert message. The badge number should be the first thing in the payload, followed by a space, and the remainder is the alert message.

On platforms that do not support badges, Push type 101 will behave exactly as push type 1, except that the badge prefix will be stripped from the message.

The `push()` callback will be called only if the user taps the notification. `Display.getInstance().getProperty("pushType")` will return "1" for this type.

Badging on iOS

The badge number can be set thru code as well, this is useful if you want the badge to represent the unread count within your application.

To do this we have two methods in the `Display` [https://www.codenameone.com/javadoc/com/codename1/ui/Display.html] class: `isBadgingSupported()` and `setBadgeNumber(int)`. Notice that even if `isBadgingSupported` will return `true`, it will not work unless you activate push support!

To truly utilize this you might need to disable the clearing of the badges on startup which you can do with the build hint `ios.enableBadgeClear=false`.

13.6. Rich Push Notifications

Rich push notifications refer to push notifications that include functionality above and beyond simply displaying an alert message to the user. Codename One's support for rich notifications includes image attachments and push actions.

13.6.1. Image Attachment Support

When you attach an image to a push notification, it will appear as a large image in the push notification on the user's device if that device supports it. iOS supports image attachments in iOS 10, Android supports them in API 26. The Javascript port, and Windows (UWP) port do not currently support image attachments. If a platform that doesn't support image attachments receives a push notification with an image attachment, it will just ignore it.

Push type "99" is used to send rich push notifications. It is sort of a "meta" push type, or a "container", as it can be used to send any of the other push types, but to attach additional content, such as image attachments.

The message body should be an XML string. A minimal example of a push type 99 that **actually** sends a push type 1, which message "Hello World", but with an attached image is:

```
<push type="1" body="Hello World"></push>
```



The image URL **must** be a secure URL (i.e. start with "https:" and not "http:"), otherwise, iOS will simply ignore it.

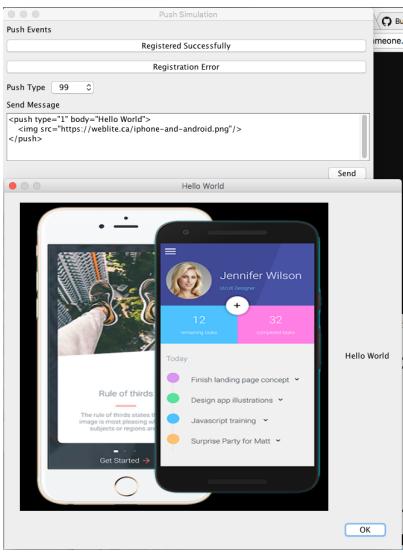


Figure 391. Push type 99 with attached image in simulator.

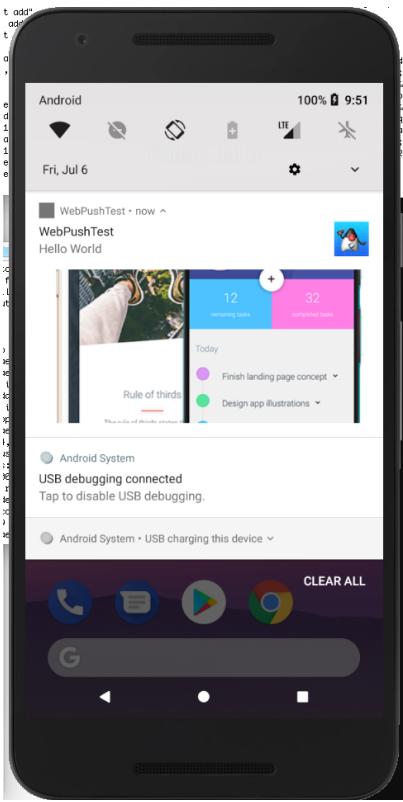


Figure 392. Push type 99 with attached image in Android when app is in background.

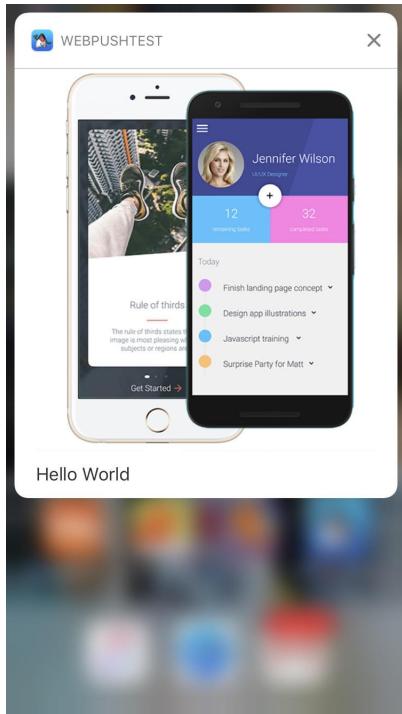


Figure 393. Push type 99 with attached image in iOS when app is in background.



The image will only be shown if you press and pull down on the notification. When the notification initially appears in the user's notifications it will appear like a normal alert - but possibly with the image shown as a small thumbnail.

The `push()` callback will receive "Hello World" as its argument and `Display.getInstance().getProperty("pushType")` will return "1" in this example.

You can access additional information about the push content using the `com.codename1.push.PushContent` class, as follows:

```
public void push(String message) {
    PushContent content = PushContent.get();
    if (content != null) {
        String imageUrl = content.getImageUrl();
        // The image attachment URL in the push notification
        // or 'null' if there was no image attachment.
    }
}
```



Make sure to only call `PushContent.get()` once inside your `push()` callback, and store the return value. `PushContent.get()` works like a queue of size=1, and it pops off the item from the front of the queue when it is called. If you call it twice, the second time will return `null`.

13.6.2. Notification Actions

When you include actions in a push notification, the user will be presented with buttons as part of the notification on supported platforms. E.g. if the notification is intended to invite the user to an

event, you might want to include buttons/actions like "Attending", "Not Attending", "Maybe", so that the user can respond quickly to the notification and not necessarily have to open your app.

You can determine whether the user has pressed a particular button on the notification using the `PushContent.getActionId()` method inside your `push()` callback.

How it works

Your app defines which action categories it supports, and associates a set of actions with each category. If a push notification includes a "category" attribute, then the notification will be presented with the associated actions manifested as buttons.

Defining the Categories and Actions

You can specify the available categories and actions for your app by implementing the `com.codename1.push.PushActionsProvider` interface in your app's main class.

E.g.

```
import com.codename1.push.PushAction;
import com.codename1.push.PushActionCategory;
import com.codename1.push.PushActionsProvider;
...

public class MyApplication implements PushCallback, PushActionsProvider {

    ...

    @Override
    public PushActionCategory[] getPushActionCategories() {
        return new PushActionCategory[]{
            new PushActionCategory("invite", new PushAction[]{
                new PushAction("yes", "Yes"),
                new PushAction("no", "No"),
                new PushAction("maybe", "Maybe")
            })
        };
    }
}
```

In the above example, we create only a single category, "invite" that has actions "yes", "no", and "maybe".

Sending a Push Notification with "invite" Category

Now we can test our new category. In the push simulator, you can select Push Type "99", with the message body:

```
<push type="1" body="Would you like to come to a party?" category="invite"/>
```

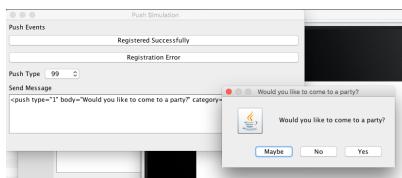


Figure 394. Push notification with "invite" category on the simulator will show dialog with buttons to select between the actions defined in the "invite" category.

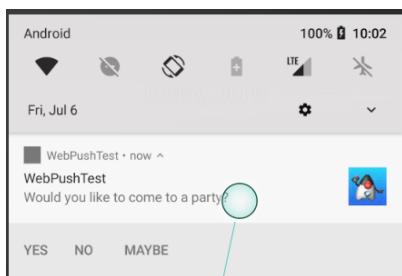


Figure 395. Push notification with "invite" category on the android will show dialog with buttons to select between the actions defined in the "invite" category.

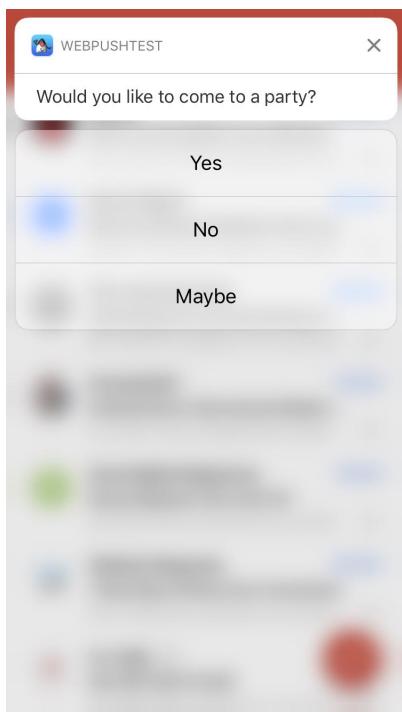


Figure 396. Push notification with "invite" category on the android will show dialog with buttons to select between the actions defined in the "invite" category.

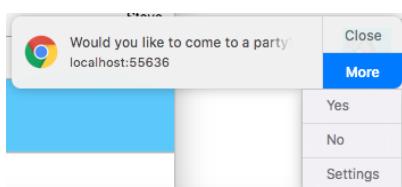


Figure 397. Push notification with "invite" category on the Chrome desktop includes a "More" dropdown where user can select the action.

The `push()` callback will be fired after the user taps on the notification, or one of its actions. If the

user taps the notification itself, and not one of the actions, then your `PushContent.getActionId()` will return `null`. If they selected one of the actions, then the action ID of that action can be obtained from `getActionId()`.

The `category` of the notification will be made available via the `getCategory()` method of `PushContent`.

E.g.

```
public void push(String message) {  
    PushContent content = PushContent.get();  
    if (content != null) {  
        if ("invite".equals(content.getCategory())) {  
            if (content.getActionId() != null) {  
                System.out.println("The user selected the "+content.getActionid()+" action");  
            } else {  
                System.out.println("The user clicked on the invite notification, but didn't select an action.");  
            }  
        }  
    }  
}
```

13.7. Deploying Push-Enabled Apps to Device

So, you've implemented the Push callback in your app, and tested it in the push simulator and it works. If you try to deploy your app to a device, you may be disappointed to discover that your app doesn't seem to be receiving push notifications when installed on the device. This is because each platform has its own bureaucracy and hoops that you have to jump through before they will deliver notifications to your app. Read on to find out how to satisfy their requirements.

13.7.1. The Push Bureaucracy - Android



To set the push icon place a 24x24 icon named `ic_stat_notify.png` under the `native/android` folder of the app. The icon can be white with transparency areas

Android Push goes thru Google servers and to do that we need to register with Google to get keys for server usage. Google uses Firebase for its cloud messaging, so we'll begin by creating a Firebase project.

Go to <https://console.firebaseio.google.com/> and click `Add project`:

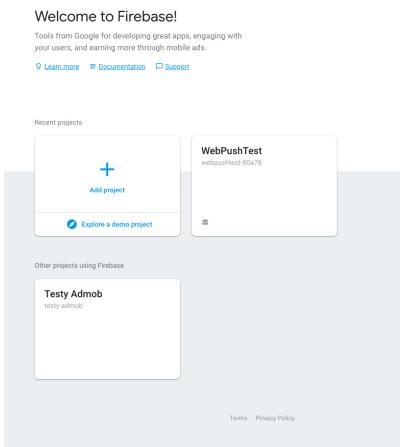


Figure 398. Click "Add project"

This will open a form as shown here:

A modal dialog titled "Add a project". It has a "Project name" input field containing "My awesome project" with a dropdown arrow. To the right of the input is a tip: "Tip: Projects span apps across platforms". Below this are fields for "Project ID" (set to "my-awesome-project-id") and "Country/region" (set to "United States"). There is a checkbox checked that says "Use the default settings for sharing Google Analytics for Firebase data". Below this is a list of four items: "Share your Analytics data with Google to improve Google Products and Services", "Share your Analytics data with Google to enable technical support", "Share your Analytics data with Google to enable Benchmarking", and "Share your Analytics data with Google Account Specialists". At the bottom, there is a checkbox for accepting terms, which is unchecked, followed by the text "I accept the controller-controller terms. This is required when sharing Analytics data to improve Google Products and Services. Learn more.", a "Cancel" button, and a "Create project" button.

Figure 399. Enter project name

Enter the project name, select your country, read/accept their terms, and press "Create Project".

Once the project has been created (should take only a few seconds), you'll be sent to your new project's dashboard.

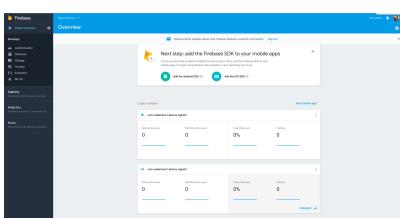


Figure 400. Firebase Project Dashboard

Expand the "Grow" section of the left menu bar, then click on the "Cloud Messaging" link.

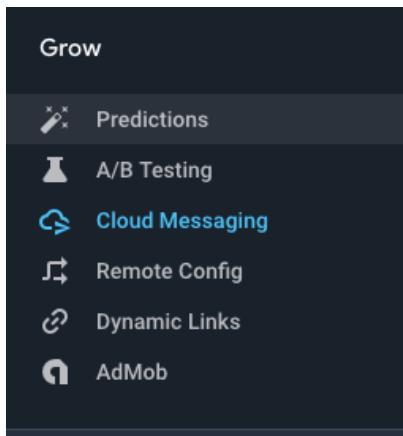


Figure 401. Expand "Grow" Section and select "Cloud Messaging"

On the next screen, click on the Android icon where it says "Add an app to get started".

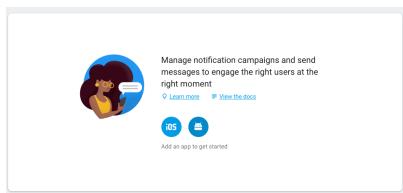


Figure 402. Click on the "Android" icon to add an Android App to the project

This will bring us to the "Add Application Form", which visually shows us the remainder of the steps.

Fill in the Android package name with the package name of your project, and the app nickname with your app's name.

The Debug signing certificate SHA-1 is optional, but you can paste the SHA-1 from your app's certificate here if you like.

A screenshot of the "Add Firebase to your Android app" registration form. The form is titled "Register app". It contains fields for "Android package name" (with the value "com.codename1.test.myapp") and "App nickname (optional)" (with the value "testmyapp"). Below these fields is a placeholder for "Debug signing certificate SHA-1 (optional)". At the bottom of the form is a large blue "Register app" button. Below the button, there is a list of three steps: "Download config file", "Add Firebase SDK", and "Run your app to verify installation".

Figure 403. Fill in the package name

Press "Register app" once you have filled in the required fields.

This will expand "Step 2" of this form: "Download config file".

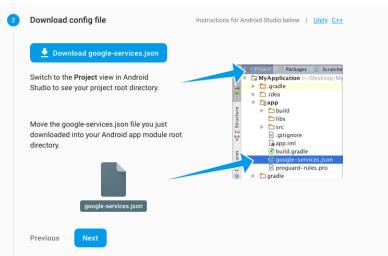


Figure 404. Download the google-services.json file

All we need to do here is press the "Download google-services.json" file, then copy the file into your project's native/android directory.



Firebase console directs you to copy the google-services.json file into the "app" directory of your project. Ignore this direction as it only applies for Android studio projects. For Codename One, this file goes into the **native/android** directory of your project.

There is one last piece of information that we need so that we can **send** push notifications to our app: The **FCM_SERVER_API_KEY** value.

Go to your project dashboard in Firebase console. Then click the "Settings" menu (the "Gear" icon next to "Project Overview" in the upper left):

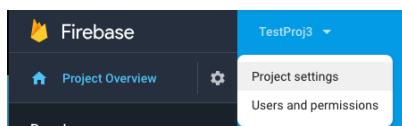


Figure 405. Select "Project settings"

Then select the "Cloud Messaging" tab.

The "Server Key" displayed here is the **FCM_SERVER_API_KEY** that we refer to throughout this document. It will be used to send push notifications to your app from a server, or from another device. You can copy and paste this value now, or you can retrieve it later by logging into the Firebase console.

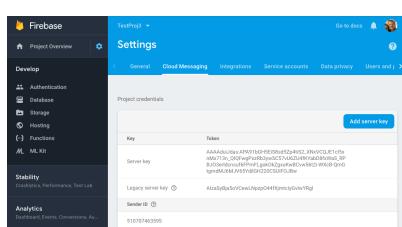


Figure 406. Save the Server key for later use.



The Sender ID shown in the above is not required for our Android app, however it is helpful/required to support Push notifications in Javascript builds (in Chrome). This value is referred to elsewhere in this document as **GCM_SENDER_ID**.

13.7.2. The Push Bureaucracy - iOS

Push on iOS is much harder to handle than the Android version, however we simplified this

significantly with the certificate wizard.

iOS push needs two additional P12 certificates.



Please **notice** that these are **NOT** the signing certificates!
They are **completely different certificates** that have nothing to do with the build process!

The certificate wizard can generate these additional push certificates and do quite a few other things if you just check this flag in the end of the wizard:

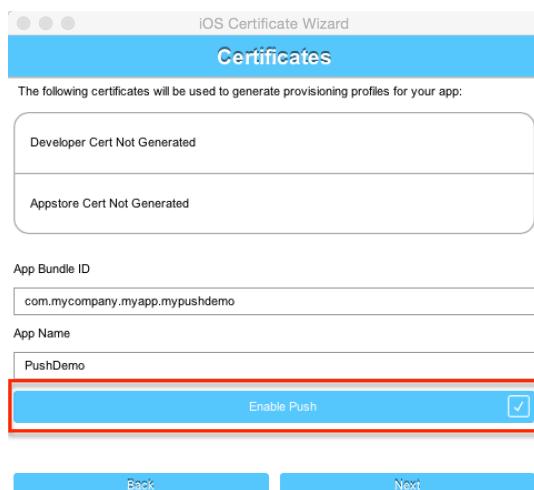


Figure 407. Enable Push in the certificate wizard

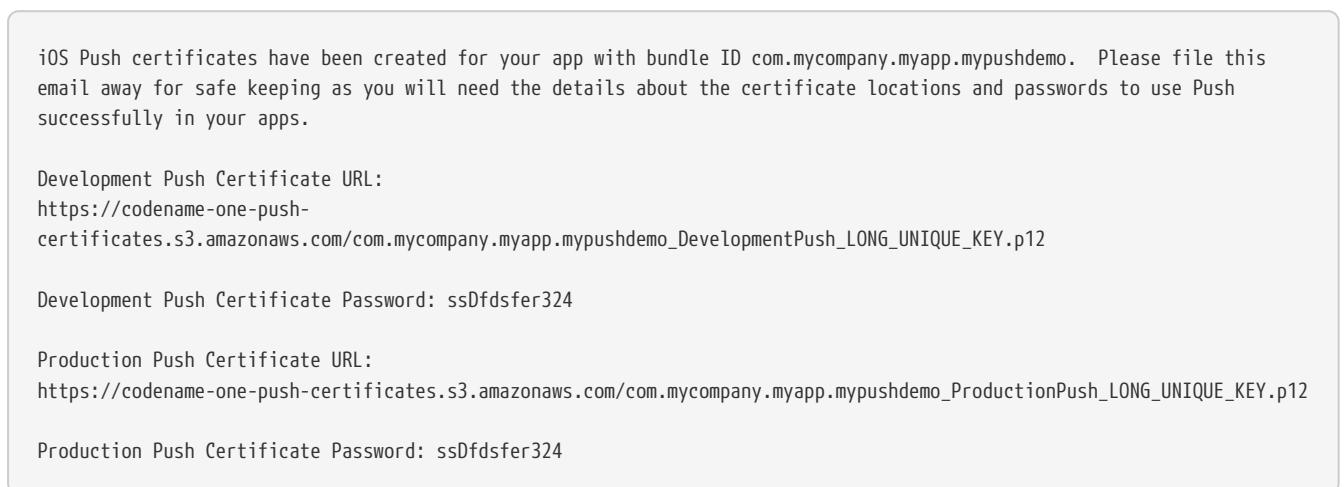


If you already have signing certificates defined for your app just skip the certificate generation phase (answer no) the rest will work as usual.

You can then install the push certificates locally and use them later on but there is an easier way.

You need to host the push certificates in the cloud so they will be reachable by the push servers, Codename One that for you seamlessly.

Once you go thru the wizard you should get an automated email containing information about push and the location of the push certificates, it should start like this:



The URL's and passwords are everything that you will need later on to get push working on iOS.

Notice that the wizard also performs a couple of other tasks specifically it sets the `ios.includePush` build hint to true & adds push to the provisioning profile etc.

You can read more about the certificate wizard in the [signing section](#) [<https://www.codenameone.com/manual/signing.html>].

13.7.3. The Push Bureaucracy - UWP (Windows 10)

Push on UWP requires only that you register your app in the Windows Store Dashboard. You will then be provided with credentials (Package Security Identifier (SID) and a secret key) that you can use to send push notifications to your app. To begin this process, go to the [Windows Dev Center](#) [<https://developer.microsoft.com/en-us/windows>] and select "Dashboard".

You can read more about the registering your app in the Windows store [here](#) [<https://www.codenameone.com/manual/appendix-uwp.html#building-for-the-windows-store>].

Once you have registered your app in the Windows Store, and completed the corresponding setup in Codename One settings (e.g. generated a certificate), you should proceed to configure your app for push notifications.

Navigate to the App overview page for your app inside the Windows store dashboard. Under the "Services" menu (left side), select "Push notifications".

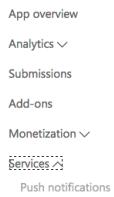


Figure 408. Push notifications menu item in Windows Store dashboard

Then, select the "WNS/MPNS" option that appears in the left menu under "Push notifications"



Figure 409. WNS menu item in Windows Store dashboard

This will bring you to a page with information about WNS push notifications. You'll be interested in the paragraph shown here:

WNS requires logging in with the Microsoft account that owns the app. If you don't have access, contact your Azure AD global administrator. If you have an existing WNS solution or to update your current client secret, visit the Live Services site.

Figure 410. Live services link

Click on the "Live Services Site" link.

You'll be prompted to log in using your Windows Store account. Then you'll be taken to a page that contains the push credentials that you can use for sending push messages to your app. You'll be interested in two values:

Package SID. (It will be of the form "ms-app://XXXXXXXXXXXX...")



Figure 411. WNS Package SID

Client Secret. This will be listed in a section called "Application Secrets"



Figure 412. WNS Client secret

You will use these two values for sending push notifications to your app.

Microsoft provides full instructions on setting up WNS notifications [here](https://msdn.microsoft.com/library/windows/apps/hh465407) [https://msdn.microsoft.com/library/windows/apps/hh465407] but much of this is not relevant for Codename One apps. For Codename One apps, one need only obtain Package Security ID and client secret values for the app.

13.7.4. The Push Bureaucracy - Javascript

Codename One apps support push in browsers that implement the Web Push API. At time of writing, this list includes:

- Firefox (Version 50)
- Chrome (Version 49)
- Opera (Version 42)
- Chrome for Android (Version 56)
- MS Edge

Firefox doesn't require any special setup for Push. If your main class implements the [PushCallback](#) interface, it should **just work**.

Chrome uses FCM for its push notifications - the same system that Android uses. The directions for setting up a FCM account are the same as provided [here](#), and you can reuse the same [GCM_SENDER_ID](#) and [FCM_API_SERVER_KEY](#) values. For Chrome push support you will need to add the [GCM_SENDER_ID](#) in the [gcm.sender_id](#) build hint so that the GCM_SENDER_ID will be added to the app's manifest file:

```
gcm.sender_id=GCM_SENDER_ID
```

Where [GCM_SENDER_ID](#) is your GCM_SENDER_ID.



Push support requires that your app be served over https with a valid SSL certificate. It will not work with the "preview" version of your app. You'll need to download the .zip or .war file and host the file on your own site - with a valid SSL certificate.

13.8. Sending Push Messages

You can send a push message in many ways e.g. from another device or any type of server but there are some values that you will need regardless of the way in which you send the push message.

```
private static final String PUSH_TOKEN = "*****_****_***_***_*****";
```

The push token is a unique "key" that you can use to send push thru your Codename One account. It allows you to send push messages without placing your Codename One email or password into your source files.

You can get it by going to the Codename One build server dashboard at <https://www.codenameone.com/build-server.html> and selecting the **Account** tab.

The token should appear at the bottom as such:

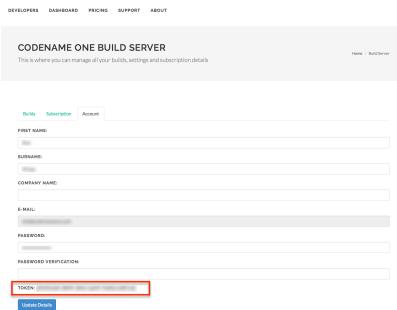


Figure 413. Push Token from the build server

The instructions for extracting the API key for Google are [listed above](#).

```
private static final String FCM_SERVER_API_KEY = "*****_*****_*****_*****";
```

The instructions for extracting the SID and Client Secret for Windows are [listed above](#).

```
private static final String WNS_SID = "ms-app://*****";  
private static final String WNS_CLIENT_SECRET = "*****";
```

When sending push to iOS devices we have two modes: - Production - Distribution

This allows you to debug the push related functionality without risking the possibility of sending a push into a production app. Its important to send the values to the right server during development/production.

```
private static final boolean ITUNES_PRODUCTION_PUSH = false;
```

iOS needs a certificate in order to send a push, this allows you to prove to Apples push servers that you are who you claim to be (the author of the app).



These are not the signing certificates and are completely separate from them!

You can obtain these two certificates (for development/appstore) via the certificate wizard as explained above.

```
private static final String ITUNES_PRODUCTION_PUSH_CERT = "https://domain.com/linkToP12Prod.p12";
private static final String ITUNES_PRODUCTION_PUSH_CERT_PASSWORD = "ProdPassword";
private static final String ITUNES_DEVELOPMENT_PUSH_CERT = "https://domain.com/linkToP12Dev.p12";
private static final String ITUNES_DEVELOPMENT_PUSH_CERT_PASSWORD = "DevPassword";
```

13.8.1. Sending a Push Message From Codename One

While normally sending a push message to a device should involve a server code there might be cases (e.g. instant messaging/social) where initiating a push from one client to another makes sense.

To simplify these use cases we added the [Push](#) [<https://www.codenameone.com/javadoc/com/codename1/push/Push.html>] API. To use the [Push](#) API you need the device key of the destination device to which you want to send the message. You can get that value from the [Push.getPushKey\(\)](#) method. Notice that you need that value from the **destination device** and not the local device!

To send a message to another device just use:

```
String cert = ITUNES_DEVELOPMENT_PUSH_CERT;
String pass = ITUNES_DEVELOPMENT_PUSH_CERT_PASSWORD;
if(ITUNES_PRODUCTION_PUSH) {
    cert = ITUNES_PRODUCTION_PUSH_CERT;
    pass = ITUNES_PRODUCTION_PUSH_CERT_PASSWORD;
}
new Push(PUSH_TOKEN, "Hello World", deviceKey)
    .apnsAuth(cert, pass, ITUNES_PRODUCTION_PUSH)
    .gcmAuth(FCM_SERVER_API_KEY)
    .wnsAuth(WNS_SID, WNS_CLIENT_SECRET)
    .send();
```

The "builder" style API used in the above sample was added post Codename One 3.6 to facilitate the addition of new Push services. If you are building against Codename one 3.6 or earlier, you should use the static [Push.sendPushMessage\(\)](#) instead as shown below:

```
Push.sendPushMessage(PUSH_TOKEN, "Hello World",
    ITUNES_PRODUCTION_PUSH, FCM_SERVER_API_KEY, cert, pass, 1, deviceKey));
```

This will send the push message "Hello World" to the device with the key [deviceKey](#). The **1** argument represents the standard push message type, which we discussed [previously](#).

13.8.2. Sending Push Message From A Java or Generic Server

Sending a push message from the server is a more elaborate affair and might require sending push messages to many devices in a single batch.

We can send a push message as an HTTP **POST** request to <https://push.codenameone.com/push/push> [<https://push.codenameone.com/push/push>]. That URL accepts the following arguments:

- **token** - your developer token to identify the account sending the push - **PUSH_TOKEN**
- **device** - one or more device keys to send the push to. You can send push to up to 500 devices with a single request -
- **type** - the message type identical to the old set of supported types in the old push servers
- **body** - the body of the message
- **auth** - the Google push auth key - **FCM_SERVER_API_KEY** (also used for sending to Chrome Javascript Apps)
- **production** - **true/false** whether to push to production or sandbox environment in iOS - **ITUNES_PRODUCTION_PUSH**
- **certPassword** - password for the push certificate in iOS push - **ITUNES_DEVELOPMENT_PUSH_CERT_PASSWORD** or **ITUNES_PRODUCTION_PUSH_CERT_PASSWORD**
- **cert** - http or https URL containing the push certificate for an iOS push - **ITUNES_DEVELOPMENT_PUSH_CERT** or **ITUNES_PRODUCTION_PUSH_CERT**
- **sid** - The package security ID (SID) for UWP apps.
- **client_secret** - The client secret for UWP apps.

We can thus send a push from Java EE using code like this:

```

HttpURLConnection connection = (HttpURLConnection) new URL("https://push.codenameone.com/push/push").openConnection();
connection.setDoOutput(true);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");
String cert = ITUNES_DEVELOPMENT_PUSH_CERT;
String pass = ITUNES_DEVELOPMENT_PUSH_CERT_PASSWORD;
if(ITUNES_PRODUCTION_PUSH) {
    cert = ITUNES_PRODUCTION_PUSH_CERT;
    pass = ITUNES_PRODUCTION_PUSH_CERT_PASSWORD;
}
String query = "token=" + PUSH_TOKEN +
    "&device=" + URLEncoder.encode(deviceId1, "UTF-8") +
    "&device=" + URLEncoder.encode(deviceId2, "UTF-8") +
    "&device=" + URLEncoder.encode(deviceId3, "UTF-8") +
    "&type=1" +
    "&auth=" + URLEncoder.encode(FCM_SERVER_API_KEY, "UTF-8") +
    "&certPassword=" + URLEncoder.encode(pass, "UTF-8") +
    "&cert=" + URLEncoder.encode(cert, "UTF-8") +
    "&body=" + URLEncoder.encode(MESSAGE_BODY, "UTF-8") +
    "&production=" + ITUNES_PRODUCTION_PUSH +
    "&sid=" + URLEncoder.encode(WNS_SID, "UTF-8") +
    "&client_secret=" + URLEncoder.encode(WNS_CLIENT_SECRET, "UTF-8");
try (OutputStream output = connection.getOutputStream()) {
    output.write(query.getBytes("UTF-8"));
}
int c = connection.getResponseCode();
// read response JSON

```

Notice that you can send a push to 500 devices. To send in larger batches you need to split the push requests into 500 device batches.

Server JSON Responses

The push servers send responses in JSON form. It's crucial to parse and manage those as they might contain important information.

If there is an error that isn't fatal such as quota exceeded etc. you will get an error message like this:

```
{"error":"Error message"}
```

A normal response, will be an array with results:

```
[
    {"id":"deviceId", "status":"error", "message":"Invalid Device ID"},
    {"id":"cn1-gcm-nativegcmkey", "status":"updateId", "newId":"cn1-gcm-newgcmkey"},
    {"id":"cn1-gcm-okgcmkey", "status":"OK"},
    {"id":"cn1-gcm-errorkey", "status":"error", "message":"Server error message"},
    {"id":"cn1-ios-iphonekey", "status":"inactive"},
]
```

There are several things to notice in the responses above:

- If the response contains `status=updateId` it means that the GCM server wants you to update the device id to a new device id. You should do that in the database and avoid sending pushes to the old key
- iOS doesn't acknowledge device receipt but it does send a `status=inactive` result which you should use to remove the device from the list of devices



APNS (Apple's push service) returns uppercase key results. This means that code for managing the keys in your database must be case insensitive



Apple doesn't always send back a result for a device being inactive and might fail silently

14. Miscellaneous Features

14.1. Phone Functions

Most of the low level phone functionality is accessible in the [Display](#) [https://www.codenameone.com/javadoc/com/codename1/ui/Display.html] class. Think of it as a global central class covering your access to the "system".

14.1.1. SMS

Codename One supports sending SMS messages but not receiving them as this functionality isn't portable. You can send an SMS using:

```
Display.getInstance().setSMS("+999999999", "My SMS Message");
```

Android/Blackberry support sending SMS's in the background without showing the user anything. iOS & Windows Phone just don't have that ability, the best they can offer is to launch the native SMS app with your message already in that app. Android supports that capability as well (launching the OS native SMS app).

The default [sendSMS](#) API ignores that difference and simply works interactively on iOS/Windows Phone while sending in the background for the other platforms.

The [getSMSSupport](#) API returns one of the following options:

- `SMS_NOT_SUPPORTED` - for desktop, tablet etc.
- `SMS_SEAMLESS` - [sendSMS](#) will not show a UI and will just send in the background
- `SMS_INTERACTIVE` - [sendSMS](#) will show an SMS sending UI
- `SMS_BOTH` - [sendSMS](#) can support both seamless and interactive mode, this currently only works on Android

The [sendSMS](#) can accept an interactive argument: `sendSMS(String phoneNumber, String message, boolean interactive)`

The last argument will be ignored unless `SMS_BOTH` is returned from [getSMSSupport](#) at which point you would be able to choose one way or the other. The default behavior (when not using that flag) is the background sending which is the current behavior on Android.

A typical use of this API would be something like this:

```

switch(Display.getInstance().getSMSsupport()) {
    case Display.SMS_NOT_SUPPORTED:
        return;
    case Display.SMS_SEAMLESS:
        showUIDialogToEditMessageData();
        Display.getInstance().sendSMS(phone, data);
        return;
    default:
        Display.getInstance().sendSMS(phone, data);
        return;
}

```

14.1.2. Dialing

Dialog the phone is pretty trivial, this should open the dialer UI without physically dialing the phone as that is discouraged by device vendors.

You can dial the phone by using:

```
Display.getInstance().dial("+9999999999");
```

14.1.3. E-Mail

You can send an email via the platforms native email client with code such as this:

```

Message m = new Message("Body of message");
Display.getInstance().sendMessage(new String[] {"someone@gmail.com"}, "Subject of message", m);

```

You can add one attachment by using `setAttachment` and `setAttachmentMimeType`.



You need to use files from `FileSystemStorage` and **NOT** `Storage` files!

You can add more than one attachment by putting them directly into the attachment map e.g.:

```

Message m = new Message("Body of message");
m.getAttachments().put(textAttachmentUri, "text/plain");
m.getAttachments().put(imageAttachmentUri, "image/png");
Display.getInstance().sendMessage(new String[] {"someone@gmail.com"}, "Subject of message", m);

```



Some features such as attachments etc. don't work correctly in the simulator but should work on iOS/Android

The email messaging API has an additional ability within the `Message` [<https://www.codenameone.com/javadoc/com/codename1/messaging/Message.html>] class. The `sendMessageViaCloud` method allows you to use the Codename One cloud to send an email without end user interaction. This feature is available to

pro users only since it makes use of the Codename One cloud:

```
Message m = new Message("<html><body>Check out <a href=\"https://www.codenameone.com/\">Codename One</a></body></html>");
m.setMimeType(Message.MIME_HTML);

// notice that we provide a plain text alternative as well in the send method
boolean success = m.sendMessageViaCloudSync("Codename One", "destination@domain.com", "Name Of User", "Message Subject",
    "Check out Codename One at https://www.codenameone.com/");
```

14.2. Contacts API

The contacts API provides us with the means to query the phone's address book, delete elements from it and create new entries into it. To get the platform specific list of contacts you can use `String[] contacts = ContactsManager.getAllContacts();`

Notice that on some platforms this will prompt the user for permissions and the user might choose not to grant that permission. To detect whether this is the case you can invoke `isContactsPermissionGranted()` after invoking `getAllContacts()`. This can help you adapt your error message to the user.

Once you have a `Contact` [https://www.codenameone.com/javadoc/com/codename1/contacts/Contact.html] you can use the `getContactById` method, however the default method is a bit slow if you want to pull a large batch of contacts. The solution for this is to only extract the data that you need via

```
getContactById(String id, boolean includesFullName,
    boolean includesPicture, boolean includesNumbers, boolean includesEmail,
    boolean includeAddress)
```

Here you can specify true only for the attributes that actually matter to you.

Another capability of the contacts API is the ability to extract all of the contacts very quickly. This isn't supported on all platforms but platforms such as Android can really get a boost from this API as extracting the contacts one by one is remarkably slow on Android.

You can check if a platform supports the extraction of all the contacts quickly thru `ContactsManager.isGetAllContactsFast()`.



When retrieving all the contacts, notice that you should probably not retrieve all the data and should set some fields to false to perform a more efficient query

You can then extract all the contacts using code that looks a bit like this, notice that we use a thread so the UI won't be blocked!

```

Form hi = new Form("Contacts", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(new InfiniteProgress());
Display.getInstance().scheduleBackgroundTask(() -> {
    Contact[] contacts = ContactsManager.getAllContacts(true, true, false, true, false, false);
    Display.getInstance().callSerially(() -> {
        hi.removeAll();
        for(Contact c : contacts) {
            MultiButton mb = new MultiButton(c.getDisplayName());
            mb.setTextLine2(c.getPrimaryPhoneNumber());
            hi.add(mb);
            mb.putClientProperty("id", c.getId());
        }
        hi.getContentPane().animateLayout(150);
    });
});
hi.show();

```

Contacts

Kenny McCormick

+111111

Kyle Broflovski

+111111

Eric Cartman

+111111

Shai Almog

+111111

Chen Fishbein

+111111

Figure 414. List of contacts

Notice that we didn't fetch the image of the contact as the performance of loading these images might be prohibitive. We can enhance the code above to include images by using slightly more complex code such as this:



The `scheduleBackgroundTask` method is similar to `new Thread()` in some regards. It places elements in a queue instead of opening too many threads so it can be good for non-urgent tasks

```

Form hi = new Form("Contacts", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(new InfiniteProgress());
int size = Display.getInstance().convertToPixels(5, true);
FontImage fi = FontImage.createFixed("") + FontImage.MATERIAL_PERSON, FontImage.getMaterialDesignFont(), 0xff, size,
size);

Display.getInstance().scheduleBackgroundTask(() -> {
    Contact[] contacts = ContactsManager.getContacts(true, true, false, true, false, false);
    Display.getInstance().callSerially(() -> {
        hi.removeAll();
        for(Contact c : contacts) {
            MultiButton mb = new MultiButton(c.getDisplayName());
            mb.setIcon(fi);
            mb.setTextLine2(c.getPrimaryPhoneNumber());
            hi.add(mb);
            mb.putClientProperty("id", c.getId());
            Display.getInstance().scheduleBackgroundTask(() -> {
                Contact cc = ContactsManager.getContactById(c.getId(), false, true, false, false, false);
                Display.getInstance().callSerially(() -> {
                    Image photo = cc.getPhoto();
                    if(photo != null) {
                        mb.setIcon(photo.fill(size, size));
                        mb.revalidate();
                    }
                });
            });
        }
        hi.getContentPane().animateLayout(150);
    });
});

```

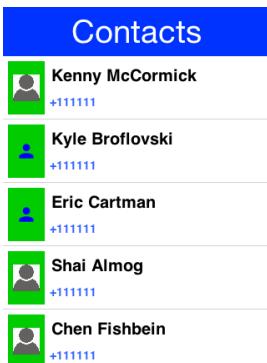


Figure 415. Contacts with the default photos on the simulator, on device these will use actual user photos when available



Notice that the code above uses `callSerially` & `scheduleBackgroundTask` in a liberal nested way. This is important to avoid an EDT violation

You can use `createContact(String firstName, String familyName, String officePhone, String homePhone, String cellPhone, String email)` to add a new contact and `deleteContact(String id)` to delete a contact.

14.3. Localization & Internationalization (L10N & I18N)

Localization (l10n) means adapting to a locale which is more than just translating to a specific language but also to a specific language within environment e.g. `en_US` != `en_UK`. Internationalization (i18n) is the process of creating one application that adapts to all locales and regional requirements.

Codename One supports automatic localization and seamless internationalization of an application using the Codename One design tool.



Although localization is performed in the design tool most features apply to hand coded applications as well. The only exception is the tool that automatically extracts localizable strings from the GUI.

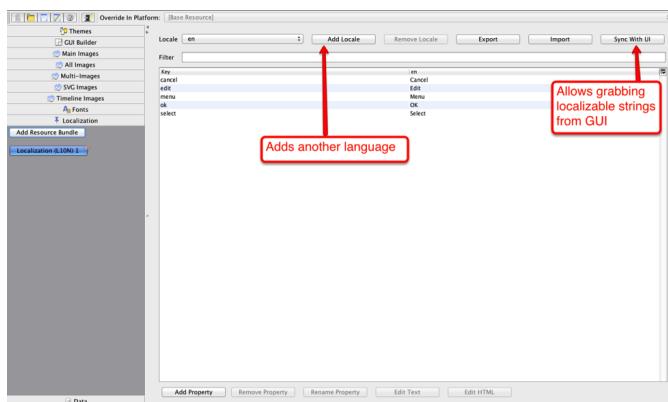


Figure 416. Localization tool in the Designer

To translate an application you need to use the localization section of the Codename One Designer. This section features a handy tool to extract localization called Sync With UI, it's a great tool to get you started assuming you used the old GUI builder.

Some fields on some components (e.g. `Commands`) are not added when using "Sync With UI" button. But you can add them manually on the localization bundle and they will be automatically localized. You can just use the `Property Key` used in the localization bundle in the Command name of the form.

You can add additional languages by pressing the `Add Locale` button.

This generates “bundles” in the resource file which are really just key/value pairs mapping a string in one language to another language. You can install the bundle using code like this:

```
UIManager.getInstance().setBundle(res.getL10N("l10n", local));
```

The device language (as an ISO 639 two letter code) could be retrieved with this:

```
String local = L10NManager.getInstance().getLanguage();
```

Once installed a resource bundle takes over the UI and every string set to a label (and label like

components) will be automatically localized based on the bundle. You can also use the localize method of [UIManager](https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html>] to perform localization on your own:

```
UIManager.getInstance().localize( "KeyInBundle", "DefaultValue");
```

The list of available languages in the resource bundle could be retrieved like this. Notice that this a list that was set by you and doesn't need to conform to the ISO language code standards:

```
Resources res = fetchResourceFile();
Enumeration locales = res.listL10NLocales( "l10n" );
```

An exception for localization is the [TextField/TextArea](#) components both of which contain user data, in those cases the text will not be localized to avoid accidental localization of user input.

You can preview localization in the theme mode within the Codename One designer by selecting [Advanced](#), picking your locale then clicking the theme again.



You can export and import resource bundles as standard Java properties files, CSV and XML. The formats are pretty standard for most localization shops, the XML format Codename One supports is the one used by Android's string bundles which means most localization specialists should easily localize it

The resource bundle is just a map between keys and values e.g. the code below displays "[This Label is localized](#)" on the [Label](#) with the hardcoded resource bundle. It would work the same with a resource bundle loaded from a resource file:

```
Form hi = new Form("L10N", new BoxLayout(BoxLayout.Y_AXIS));
HashMap<String, String> resourceBudle = new HashMap<String, String>();
resourceBudle.put("Localize", "This Label is localized");
UIManager.getInstance().setBundle(resourceBudle);
hi.add(new Label("Localize"));
hi.show();
```



Figure 417. Localized label

14.3.1. Localization Manager

The [L10NManager](https://www.codenameone.com/javadoc/com/codename1/l10n/L10NManager.html) [<https://www.codenameone.com/javadoc/com/codename1/l10n/L10NManager.html>] class includes a multitude of features useful for common localization tasks.

It allows formatting numbers/dates & time based on platform locale. It also provides a great deal of the information you need such as the language/locale information you need to pick the proper

resource bundle.

```
Form hi = new Form("L10N", new TableLayout(16, 2));
L10NManager l10n = L10NManager.getInstance();
hi.add("format(double)").add(l10n.format(11.11)).
    add("format(int)").add(l10n.format(33)).
    add("formatCurrency").add(l10n.formatCurrency(53.267)).
    add("formatDateLongStyle").add(l10n.formatDateLongStyle(new Date())).
    add("formatDateShortStyle").add(l10n.formatDateShortStyle(new Date())).
    add("formatDateTime").add(l10n.formatDateTime(new Date())).
    add("formatDateTimeMedium").add(l10n.formatDateTimeMedium(new Date())).
    add("formatDateTimeShort").add(l10n.formatDateTimeShort(new Date())).
    add("getCurrencySymbol").add(l10n.getCurrencySymbol()).
    add("getLanguage").add(l10n.getLanguage()).
    add("getLocale").add(l10n.getLocale()).
    add("isRTLLocale").add("'" + l10n.isRTLLocale()).
    add("parseCurrency").add(l10n.formatCurrency(l10n.parseCurrency("33.77$"))).
    add("parseDouble").add(l10n.format(l10n.parseDouble("34.35"))).
    add("parseInt").add(l10n.format(l10n.parseInt("56"))).
    add("parseLong").add("'" + l10n.parseLong("4444444"));
hi.show();
```

L10N	
format(double)	11.11
format(int)	33
formatCurrency	ILS53.27
formatDateLongStyle	March 4, 2016
formatDateShortStyle	3/4/16
formatDateTime	Mar 4, 2016 2:39
formatDateTimeMedium	Mar 4, 2016 2:39
formatDateTimeShort	3/4/16 2:39 PM
getCurrencySymbol	ILS
getLanguage	en
getLocale	US
isRTLLocale	false
parseCurrency	ILS33.77

Figure 418. Localization formatting/parsing and information

14.3.2. RTL/Bidi

RTL stands for right to left, in the world of internationalization it refers to languages that are written from right to left (Arabic, Hebrew, Syriac, Thaana).

Most western languages are written from left to right (LTR), however some languages are written from right to left (RTL) speakers of these languages expect the UI to flow in the opposite direction otherwise it seems weird just like reading this word would be to most English speakers: "drieW".

The problem posed by RTL languages is known as BiDi (Bi-directional) and not as RTL since the "true" problem isn't the reversal of the writing/UI but rather the mixing of RTL and LTR together. E.g. numbers are always written from left to right (just like in English) so in an RTL language the direction is from right to left and once we reach a number or English text embedded in the middle of the sentence (such as a name) the direction switches for a duration and is later restored.

The main issue in the Codename One world is in the layouts, which need to reverse on the fly. Codename One supports this via an RTL flag on all components that is derived from the global **RTL** flag in [UIManager](https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html>].

Resource bundles can also include special case constant `@rtl`, which indicates if a language is written from right to left. This allows everything to automatically reverse.

When in **RTL** mode the UI will be the exact mirror so **WEST** will become **EAST**, **RIGHT** will become **LEFT** and this would be true for paddings/margins as well.

If you have a special case where you don't want this behavior you will need to wrap it with an `isRTL` check. You can also use `setRTL` on a per [Component](#) basis to disable RTL behavior for a specific [Component](#).



Most UI API's have special cases for BiDi instead of applying it globally e.g. AWT introduced constants such as **LEADING** instead of making **WEST** mean the opposite direction. We think that was a mistake since the cases where you wouldn't want the behavior of automatic reversal are quite rare.

Codename One's support for bidi includes the following components:

- **Bidi algorithm** - allows converting between logical to visual representation for rendering
- **Global RTL flag** - default flag for the entire application indicating the UI should flow from right to left
- **Individual RTL flag** - flag indicating that the specific component/container should be presented as an RTL/LTR component (e.g. for displaying English elements within a RTL UI).
- **RTL text field input**

Most of Codename One's RTL support is under the hood, the [LookAndFeel](#) [<https://www.codenameone.com/javadoc/com/codename1/ui/plaf/LookAndFeel.html>] global RTL flag can be enabled using:

```
UIManager.getInstance().getLookAndFeel().setRTL(true);
```

Once RTL is activated all positions in Codename One become reversed and the UI becomes a mirror of itself. E.g. Adding a [Toolbar](#) command to the left will actually make it appear on the right. Padding on the left becomes padding on the right. The scroll moves to the left etc.

This applies to the layout managers (except for group layout) and most components. Bidi is mostly seamless in Codename One but a developer still needs to be aware that his UI might be mirrored for these cases.

14.4. Location - GPS

The [Location](#) [<https://www.codenameone.com/javadoc/com/codename1/location/Location.html>] API allows us to track changes in device location or the current user position.



The Simulator includes a **Location Simulation** tool that you can launch to determine the current position of the simulator and debug location events

The most basic usage for the API allows us to just fetch a device Location, notice that this API is blocking and can take a while to return:

```
Location position = LocationManager.getLocationManager().getCurrentLocationSync();
```



In order for location to work on iOS you **MUST** define the build hint `ios.locationUsageDescription` and describe why your application needs access to location. Otherwise you won't get location updates!

The `getCurrentLocationSync()` method is very good for cases where you only need to fetch a current location once and not repeatedly query location. It activates the GPS then turns it off to avoid excessive battery usage. However, if an application needs to track motion or position over time it should use the location listener API to track location as such:



Notice that there is a method called `getCurrentLocation()` which will return the current state immediately and might not be accurate for some cases.

```
public MyListener implements LocationListener {  
    public void locationUpdated(Location location) {  
        // update UI etc.  
    }  
  
    public void providerStateChanged(int newState) {  
        // handle status changes/errors appropriately  
    }  
}  
LocationManager.getLocationManager().setLocationListener(new MyListener());
```



On Android location maps to low level API's if you disable the usage of Google Play Services. By default location should perform well if you leave the Google Play Services on

14.4.1. Location In The Background - Geofencing

Polling location is generally expensive and requires a special permission on iOS. Its also implemented rather differently both in iOS and Android. Both platforms place restrictions on the location API usage in the background.

Because of the nature of background location the API is non-trivial. It starts with the venerable `LocationManager` but instead of using the standard API you need to use `setBackgroundLocationListener`.

Instead of passing a `LocationListener` instance you need to pass a `Class` object instance. This is

important because background location might be invoked when the app isn't running and an object would need to be allocated.

Notice that you should **NOT** perform long operations in the background listener callback. IOS wake-up time is limited to approximately 10 seconds and the app could get killed if it exceeds that time slice.

Notice that the listener can also send events when the app is in the foreground, therefore it is recommended to check the app state before deciding how to process this event. You can use `Display.isMinimized()` to determine if the app is currently running or in the background.

When implementing this make sure that:

- The class passed to the API is a public class in the global scope. Not an inner class or anything like that!
- The class has a public no-argument constructor
- You need to pass it as a class literal e.g. `MyClassName.class`. Don't use `Class.forName("my.package.MyClassName")`!

Class names are problematic since device builds are obfuscated, you should only use literals which the obfuscator detects and handles correctly.

The following code demonstrates usage of the GeoFence API:

```
Geofence gf = new Geofence("test", loc, 100, 100000);

LocationManager.getLocoationManager()
    .addGeoFencing(GeofenceListenerImpl.class, gf);
```

```
public class GeofenceListenerImpl implements GeofenceListener {
    @Override
    public void onExit(String id) {

    }

    @Override
    public void onEntered(String id) {
        if(Display.getInstance().isMinimized()) {
            Display.getInstance().callSerially(() -> {
                Dialog.show("Welcome", "Thanks for arriving", "OK", null);
            });
        } else {
            LocalNotification ln = new LocalNotification();
            ln.setAlertTitle("Welcome");
            ln.setAlertBody("Thanks for arriving!");
            Display.getInstance().scheduleLocalNotification(ln, 10, false);
        }
    }
}
```

14.5. Background Music Playback

Codename One supports playing music in the background (e.g. when the app is minimized) which is quite useful for developers building a music player style application.

This support isn't totally portable since the Android and iOS approaches for background music playback differ a great deal. To get this to work on Android you need to use the API: [MediaManager.createBackgroundMedia\(\)](#).

You should use that API when you want to create a media stream that will work even when your app is minimized.

For iOS you will need to use a special build hint: [ios.background_modes=music](#).

Which should allow background playback of music on iOS and would work with the [createBackgroundMedia\(\)](#) method.

14.6. Capture - Photos, Video, Audio

The capture API allows us to use the camera to capture photographs or the microphone to capture audio. It even includes an API for video capture.

The API itself couldn't be simpler:

```
String filePath = Capture.capturePhoto();
```

Just captures and returns a path to a photo you can either open it using the [Image](#) [<https://www.codenameone.com/javadoc/com/codenname1/ui/Image.html>] class or save it somewhere.



The returned file is a temporary file, you shouldn't store a reference to it and instead copy it locally or work with the [Image](#) object

E.g. you can copy the [Image](#) to [Storage](#) using:

```
String filePath = Capture.capturePhoto();
if(filePath != null) {
    Util.copy(FileSystemStorage.getInstance().openInputStream(filePath),
    Storage.getInstance().createOutputStream(myImageFileName));
}
```



When running on the simulator the [Capture](#) API opens a file chooser API instead of physically capturing the data. This makes debugging device or situation specific issues simpler

We can capture an image from the camera using an API like this:

```

Form hi = new Form("Capture", new BorderLayout());
hi.setToolbar(new Toolbar());
Style s = UIManager.getInstance().getComponentStyle("Title");
FontImage icon = FontImage.createMaterial(FontImage.MATERIAL_CAMERA, s);

ImageViewer iv = new ImageViewer(icon);

hi.getToolbar().addCommandToRightBar("", icon, (ev) -> {
    String filePath = Capture.capturePhoto();
    if(filePath != null) {
        try {
            DefaultListModel<Image> m = (DefaultListModel<Image>)iv.getImageList();
            Image img = Image.createImage(filePath);
            if(m == null) {
                m = new DefaultListModel<>(img);
                iv.setImageList(m);
                iv.setImage(img);
            } else {
                m.addItem(img);
            }
            m.setSelectedIndex(m.getSize() - 1);
        } catch(IOException err) {
            Log.e(err);
        }
    }
});

hi.add(BorderLayout.CENTER, iv);
hi.show();

```



Figure 419. Captured photos previewed in the ImageViewer

We demonstrate video capture in the [MediaManager section](https://www.codenameone.com/manual/components.html#mediamanager-section) [<https://www.codenameone.com/manual/components.html#mediamanager-section>].

The sample below captures audio recordings (using the 'Capture' API) and copies them locally under unique names. It also demonstrates the storage and organization of captured audio:

```

Form hi = new Form("Capture", BoxLayout.y());
hi.setToolbar(new Toolbar());
Style s = UIManager.getInstance().getComponentStyle("Title");
FontImage icon = FontImage.createMaterial(FontImage.MATERIAL_MIC, s);

FileSystemStorage fs = FileSystemStorage.getInstance();
String recordingsDir = fs.getAppHomePath() + "recordings/";
fs.mkdir(recordingsDir);
try {
    for(String file : fs.listFiles(recordingsDir)) {
        MultiButton mb = new MultiButton(file.substring(file.lastIndexOf("/") + 1));
        mb.addActionListener((e) -> {
            try {
                Media m = MediaManager.createMedia(recordingsDir + file, false);
                m.play();
            } catch(IOException err) {
                Log.e(err);
            }
        });
        hi.add(mb);
    }
}

hi.getToolbar().addCommandToRightBar("", icon, (ev) -> {
    try {
        String file = Capture.captureAudio();
        if(file != null) {
            SimpleDateFormat sd = new SimpleDateFormat("yyyy-MMM-dd kk-mm");
            String fileName =sd.format(new Date());
            String filePath = recordingsDir + fileName;
            Util.copy(fs.openInputStream(file), fs.openOutputStream(filePath));
            MultiButton mb = new MultiButton(fileName);
            mb.addActionListener((e) -> {
                try {
                    Media m = MediaManager.createMedia(filePath, false);
                    m.play();
                } catch(IOException err) {
                    Log.e(err);
                }
            });
            hi.add(mb);
            hi.revalidate();
        }
    } catch(IOException err) {
        Log.e(err);
    }
});
hi.show();
}

```



Figure 420. Captured recordings in the demo

Alternatively, you can use the `Media`, `MediaManager` and `MediaRecorderBuilder` APIs to capture audio, as a more customizable approach than using the Capture API:

```
private static final EasyThread countTime = EasyThread.start("countTime");

public void start() {
    if (current != null) {
        current.show();
        return;
    }
    Form hi = new Form("Recording audio", BoxLayout.y());
    hi.add(new SpanLabel("Example of recording and playback audio using the Media, MediaManager and
MediaRecorderBuilder APIs"));
    hi.add(recordAudio((String filePath) -> {
        ToastBar.showInfoMessage("Do something with the recorded audio file: " + filePath);
    }));
    hi.show();
}

public static Component recordAudio(OnComplete<String> callback) {
    try {
        // mime types supported by Android: audio/amr, audio/aac, audio/mp4
        // mime types supported by iOS: audio/mp4, audio/aac, audio/m4a
        // mime type supported by Simulator: audio/wav
        // more info: https://www.iana.org/assignments/media-types/media-types.xhtml

        List<String> availableMimetypes = Arrays.asList(MediaManager.getAvailableRecordingMimeTypes());
        String mimetype;
        if (availableMimetypes.contains("audio/aac")) {
            // Android and iOS
            mimetype = "audio/aac";
        } else if (availableMimetypes.contains("audio/wav")) {
            // Simulator
            mimetype = "audio/wav";
        } else {
            // others
            mimetype = availableMimetypes.get(0);
        }
        String fileName = "audioExample." + mimetype.substring(mimetype.indexOf("/") + 1);
        String output = FileSystemStorage.getInstance().getAppHomePath() + "/" + fileName;
        // https://tritondigitalcommunity.force.com/s/article/Choosing-Audio-Bitrate-Settings
        MediaRecorderBuilder options = new MediaRecorderBuilder()
            .mimeType(mimetype)
            .path(output)
            .bitRate(64000)
            .samplingRate(44100);
        Media[] microphone = {MediaManager.createMediaRecorder(options)};
        Media[] speaker = {null};
    }
}
```

```

Container recordingUI = new Container(BoxLayout.y());
Label time = new Label("0:00");
Button recordBtn = new Button("", FontImage.MATERIAL_FIBER_MANUAL_RECORD, "Button");
Button playBtn = new Button("", FontImage.MATERIAL_PLAY_ARROW, "Button");
Button stopBtn = new Button("", FontImage.MATERIAL_STOP, "Button");
Button sendBtn = new Button("Send");
sendBtn.setEnabled(false);
Container buttons = GridLayout.encloseIn(3, recordBtn, stopBtn, sendBtn);
recordingUI.addAll(FlowLayout.encloseCenter(time), FlowLayout.encloseCenter(buttons));

recordBtn.addActionListener(l -> {
    try {
        // every time we have to create a new instance of Media to make it working correctly (as reported in
        the Javadoc)
        microphone[0] = MediaManager.createMediaRecorder(options);
        if (speaker[0] != null && speaker[0].isPlaying()) {
            return; // do nothing if the audio is currently recorded or played
        }
        recordBtn.setEnabled(false);
        sendBtn.setEnabled(true);
        Log.p("Audio recording started", Log.DEBUG);
        if (buttons.contains(playBtn)) {
            buttons.replace(playBtn, stopBtn, CommonTransitions.createEmpty());
            buttons.revalidateWithAnimationSafety();
        }
        if (speaker[0] != null) {
            speaker[0].pause();
        }

        microphone[0].play();
        startWatch(time);
    } catch (IOException ex) {
        Log.p("ERROR recording audio", Log.ERROR);
        Log.e(ex);
    }
});

stopBtn.addActionListener(l -> {
    if (!microphone[0].isPlaying() && (speaker[0] == null || !speaker[0].isPlaying())) {
        return; // do nothing if the audio is NOT currently recorded or played
    }
    recordBtn.setEnabled(true);
    sendBtn.setEnabled(true);
    Log.p("Audio recording stopped");
    if (microphone[0].isPlaying()) {
        microphone[0].pause();
    } else if (speaker[0] != null) {
        speaker[0].pause();
    } else {
        return;
    }
    stopWatch(time);
    if (buttons.contains(stopBtn)) {
        buttons.replace(stopBtn, playBtn, CommonTransitions.createEmpty());
        buttons.revalidateWithAnimationSafety();
    }
    if (FileSystemStorage.getInstance().exists(output)) {
        Log.p("Audio saved to: " + output);
    } else {
        ToastBar.showErrorMessage("Error recording audio", 5000);
        Log.p("ERROR SAVING AUDIO");
    }
});

playBtn.addActionListener(l -> {
    // every time we have to create a new instance of Media to make it working correctly (as reported in the
    Javadoc)
}

```

```

        if (microphone[0].isPlaying() || (speaker[0] != null && speaker[0].isPlaying())) {
            return; // do nothing if the audio is currently recorded or played
        }
        recordBtn.setEnabled(false);
        sendBtn.setEnabled(true);
        if (buttons.contains(playBtn)) {
            buttons.replace(playBtn, stopBtn, CommonTransitions.createEmpty());
            buttons.revalidateWithAnimationSafety();
        }
        if (FileSystemStorage.getInstance().exists(output)) {
            try {
                speaker[0] = MediaManager.createMedia(output, false, () -> {
                    // callback on completion
                    recordBtn.setEnabled(true);
                    if (speaker[0].isPlaying()) {
                        speaker[0].pause();
                    }
                    stopWatch(time);
                    if (buttons.contains(stopBtn)) {
                        buttons.replace(stopBtn, playBtn, CommonTransitions.createEmpty());
                        buttons.revalidateWithAnimationSafety();
                    }
                });
                speaker[0].play();
                startWatch(time);
            } catch (IOException ex) {
                Log.p("ERROR playing audio", Log.ERROR);
                Log.e(ex);
            }
        }
    });
}

sendBtn.addActionListener(l -> {
    if (microphone[0].isPlaying()) {
        microphone[0].pause();
    }
    if (speaker[0] != null && speaker[0].isPlaying()) {
        speaker[0].pause();
    }
    if (buttons.contains(stopBtn)) {
        buttons.replace(stopBtn, playBtn, CommonTransitions.createEmpty());
        buttons.revalidateWithAnimationSafety();
    }
    stopWatch(time);
    recordBtn.setEnabled(true);

    callback.completed(output);
});

return FlowLayout.encloseCenter(recordingUI);
}

} catch (IOException ex) {
    Log.p("ERROR recording audio", Log.ERROR);
    Log.e(ex);
    return new Label("Error recording audio");
}
}

private static void startWatch(Label label) {
    label.putClientProperty("stopTime", Boolean.FALSE);
    countTime.run(() -> {
        long startTime = System.currentTimeMillis();
        while (label.getClientProperty("stopTime") == Boolean.FALSE) {
            // the sleep is every 200ms instead of 1000ms to make the app more reactive when stop is tapped
            Util.sleep(200);
            int seconds = (int) ((System.currentTimeMillis() - startTime) / 1000);
        }
    });
}

```

```

        String min = (seconds / 60) + "";
        String sec = (seconds % 60) + "";
        if (sec.length() == 1) {
            sec = "0" + sec;
        }
        String newTime = min + ":" + sec;
        if (!label.getText().equals(newTime)) {
            CN.callSerially(() -> {
                label.setText(newTime);
                if (label.getParent() != null) {
                    label.getParent().revalidateWithAnimationSafety();
                }
            });
        }
    });

private static void stopWatch(Label label) {
    label.putClientProperty("stopTime", Boolean.TRUE);
}

```

[Example of recording and playback audio using Media API] | <https://user->

14.6.1. Capture Asynchronous API

The **Capture** API also includes a callback based API that uses the **ActionListener** interface to implement capture. E.g. we can adapt the previous sample to use this API as such:

```
hi.getToolbar().addCommandToRightBar("", icon, (ev) -> {
    Capture.capturePhoto((e) -> {
        if(e != null && e.getSource() != null) {
            try {
                DefaultListModel<Image> m = (DefaultListModel<Image>)iv.getImageList();
                Image img = Image.createImage((String)e.getSource());
                if(m == null) {
                    m = new DefaultListModel<>(img);
                    iv.setImageList(m);
                    iv.setImageResource(img);
                } else {
                    m.addItem(img);
                }
                m.setSelectedIndex(m.getSize() - 1);
            } catch(IOException err) {
                Log.e(err);
            }
        }
    });
});
```

14.7. Gallery

The gallery API allows picking an image and/or video from the cameras gallery (camera roll).



Like the **Capture** API the image returned is a temporary image that should be copied locally, this is due to device restrictions that don't allow direct modifications of the gallery

We can adapt the **Capture** sample above to use the gallery as such:

```

Form hi = new Form("Capture", new BorderLayout());
hi.setToolbar(new Toolbar());
Style s = UIManager.getInstance().getComponentStyle("Title");
FontImage icon = FontImage.createMaterial(FontImage.MATERIAL_CAMERA, s);

ImageViewer iv = new ImageViewer(icon);

hi.getToolbar().addCommandToRightBar("", icon, (ev) -> {
    Display.getInstance().openGallery((e) -> {
        if(e != null && e.getSource() != null) {
            try {
                DefaultListModel<Image> m = (DefaultListModel<Image>)iv.getImageList();
                Image img = Image.createImage((String)e.getSource());
                if(m == null) {
                    m = new DefaultListModel<>(img);
                    iv.setImageList(m);
                    iv.setImage(img);
                } else {
                    m.addItem(img);
                }
                m.setSelectedIndex(m.getSize() - 1);
            } catch(IOException err) {
                Log.e(err);
            }
        }
    }, Display.GALLERY_IMAGE);
});

hi.add(BorderLayout.CENTER, iv);

```



There is no need for a screenshot as it will look identical to the capture image screenshot above

The last value is the type of content picked which can be one of: `Display.GALLERY_ALL`, `Display.GALLERY_VIDEO` or `Display.GALLERY_IMAGE`.

14.8. Analytics Integration

One of the features in Codename One is builtin support for analytic instrumentation. Currently Codename One has builtin support for [Google Analytics](https://www.google.com/analytics/) [<https://www.google.com/analytics/>], which provides reasonable enough statistics of application usage.

Analytics is pretty seamless for the old GUI builder since navigation occurs via the Codename One API and can be logged without developer interaction. However, to begin the instrumentation one needs to add the line:

```
AnalyticsService.setAppsMode(true);
AnalyticsService.init(agent, domain);
```

To get the value for the agent value just create a Google Analytics account and add a domain, then copy and paste the string that looks something like UA-99999999-8 from the console to the agent string. Once this is in place you should start receiving statistic events for the application.

If your application is not a GUI builder application or you would like to send more detailed data you can use the `Analytics.visit()` method to indicate that you are entering a specific page.

14.8.1. Application Level Analytics

In 2013 Google introduced an improved application level analytics API that is specifically built for mobile apps. However, it requires a slightly different API usage. You can activate this specific mode by invoking `setAppsMode(true)`.

When using this mode you can also report errors and crashes to the Google analytics server using the `sendCrashReport(Throwable, String message, boolean fatal)` method.

We generally recommend using this mode and setting up an apps analytics account as the results are more refined.

14.8.2. Overriding The Analytics Implementation

The Analytics API can also be enhanced to support any other form of analytics solution of your own choosing by deriving the `AnalyticsService` class.

This allows you to integrate with any 3rd party via native or otherwise by overriding methods in the `AnalyticsService` class then invoking:

```
AnalyticsService.init(new MyAnalyticsServiceSubclass());
```

Notice that this removes the need to invoke the other `init` method or `setAppsMode(boolean)`.

14.9. Native Facebook Support



Check out the [ShareButton section](#) [<https://www.codenameone.com/manual/components.html#sharebutton-section>] it might be enough for most of your needs.

Codename One supports Facebooks [Oauth2](#) [<https://www.codenameone.com/javadoc/com/codename1/io/Oauth2.html>] login and Facebooks single sign on for iOS and Android.

14.9.1. Getting Started - Web Setup

To get started first you will need to create a facebook app on the Facebook developer portal at <https://developers.facebook.com/apps/>



Figure 421. Create New App

You need to repeat the process for web, Android & iOS (web is used by the simulator):

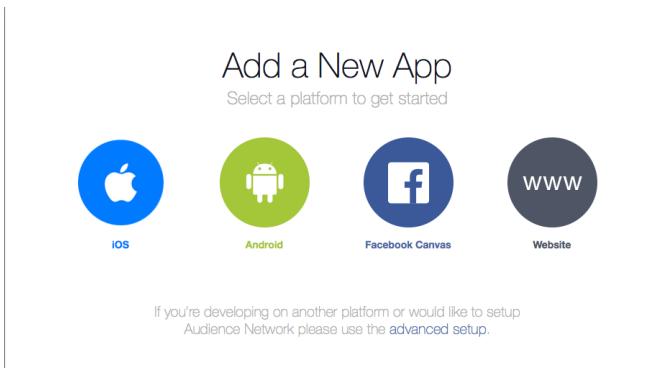


Figure 422. Pick Platform

For the first platform you need to enter the app name:



Figure 423. Pick app name

And provide some basic details:

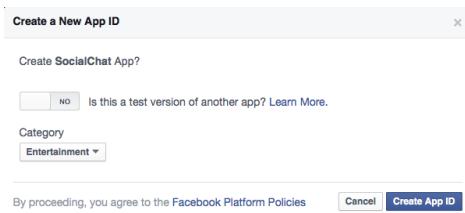


Figure 424. Basic details for the app

For iOS we need the bundle ID which is the exact same thing we used in the Google+ login to identify the iOS app its effectively your package name:

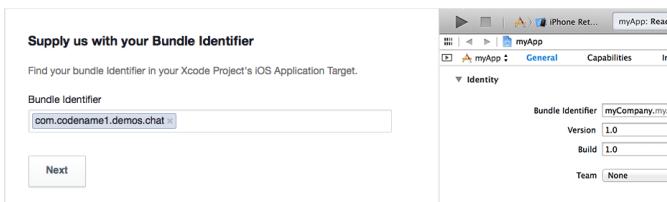


Figure 425. iOS specific basic details

You should end up with something that looks like this:

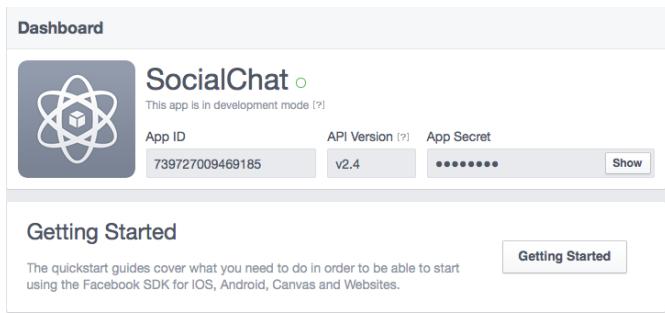


Figure 426. Finished Facebook app

The Android process is pretty similar but in this case we need the activity name too.

! The activity name should match the main class name followed by the word **Stub** (uppercase s). E.g. for the main class **SocialChat** we would use **SocialChatStub** as the activity name

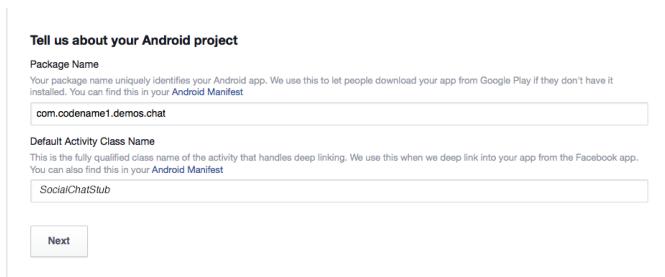


Figure 427. Android Activity definition

To build the native Android app we must make sure that we setup the keystore correctly for our application. If you don't have an Android certificate you can use the visual wizard (in the Android section in the project preferences the button labeled **Generate**) or use the command line:

```
keytool -genkey -keystore Keystore.ks -alias [alias_name] -keyalg RSA -keysize 2048 -validity 15000 -dname "CN=[full name], OU=[ou], O=[comp], L=[City], S=[State], C=[Country Code]" -storepass [password] -keypass [password]
```

! You can reuse the certificate in all your apps, some developers like having a different certificate for every app. This is like having one master key for all your doors, or a huge keyring filled with keys.

With the certificate we need an SHA1 key to further authenticate us to Facebook and we do this using the keytool command line on Linux/Mac:

```
keytool -exportcert -alias (your_keystore_alias) -keystore (path_to_your_keystore) | openssl sha1 -binary | openssl base64
```

And on Windows:

```
keytool -exportcert -alias androiddebugkey -keystore %HOMEPATH%\.android\debug.keystore | openssl sha1 -binary | openssl base64
```

You can read more about it on the [Facebook guide here](https://developers.facebook.com/docs/android/) [<https://developers.facebook.com/docs/android/>]

getting-started].

Add your development and release key hashes

To ensure the authenticity of the interactions between your app and Facebook, you need to supply us with the Android key hash for your development environment. If your app has already been published, you should add your release key hash too.

You'll have a unique development key hash for each Android development environment. To generate a development key hash, on Mac, run the following command:

```
keytool -exportcert -alias androiddebugkey -keystore ~/android/debug.keystore | openssl sha1 -binary
```

On Windows, run this command:

```
keytool -exportcert -alias androiddebugkey -keystore %HOMEPATH%\android\debug.keystore | openssl sha1 -binary
```

This command will generate a 28-character key hash unique to your development environment. Copy and paste it into the field below. You will need to provide a development key hash for the development environment of each person who works on your app.

Development Key Hashes

HhjRXZga0ZNtDCz8rVubYCjStk=

If your app has already been published, you should also add a hash of your release key.

Show how to generate a release key hash

Release Key Hash

HhjRXZga0ZNtDCz8rVubYCjStk=

Next

Figure 428. Hash generation process, notice the command lines are listed as part of the web wizard

Lastly you need to publish the Facebook app by flipping the switch in the apps "Status & Review" page as such:

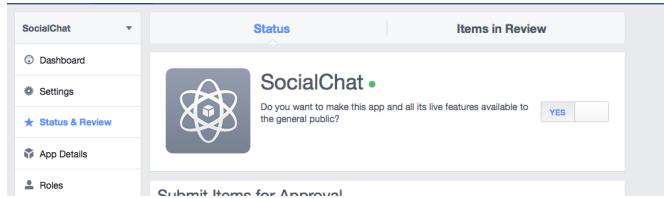


Figure 429. Without flipping the switch the app won't "appear"

14.9.2. IDE Setup

We now need to set some important build hints in the project so it will work correctly. To set the build hints just right click the project select project properties and in the Codename One section pick the second tab. Add this entry into the table:

```
facebook.appId=...
```

The app ID will be visible in your Facebook app page in the top left position.

14.9.3. The Code

To bind your mobile app into the Facebook app you can use the following code:

```

Login fb = FacebookConnect.getInstance();

fb.setClientId("9999999");
fb.setRedirectURI("http://www.youruri.com/");
fb.setClientSecret("-----");

// Sets a LoginCallback listener
fb.setCallback(new LoginCallback() {
    public void loginSuccessful() {
        // we can now start fetching stuff from Facebook!
    }

    public void loginFailed(String errorMessage) {}
});

// trigger the login if not already logged in
if(!fb.isUserLoggedIn()){
    fb.doLogin();
} else {
    // get the token and now you can query the Facebook API
    String token = fb.getAccessToken().getToken();
    // ...
}

```

All of these values are from the web version of the app!

They are only used in the simulator and on "unsupported" platforms as a fallback.
Android and iOS will use the native login

14.9.4. Facebook Publish Permissions

In order to post something to Facebook you need to request a write permission, you can only do write operations within the callback which is invoked when the user approves the permission.

You can prompt the user for publish permissions by using this code on a logged in [FacebookConnect](https://www.codenameone.com/javadoc/com/codename1/social/FacebookConnect.html) [<https://www.codenameone.com/javadoc/com/codename1/social/FacebookConnect.html>]:

```

FacebookConnect.getInstance().askPublishPermissions(new LoginCallback() {
    public void loginSuccessful() {
        // do something...
    }
    public void loginFailed(String errorMessage) {
        // show error or just ignore
    }
});

```



Notice that this won't always prompt the user, but its required to verify that your token is valid for writing.

14.10. Google Sign-In

Google Login is a bit of a moving target, as they are regularly creating new APIs and deprecating old ones. Codename One 3.7 and earlier used the Google+ API for sign-in, which is now deprecated. While this API still works, it is no longer useful on iOS as it redirects to Safari to perform login, and Apple no longer allows this practice.

The new, approved API is called Google Sign-In. Rather than using Safari to handle login (on iOS), it uses an embedded web view, which is permitted by Apple.

The process involves four parts:

1. [Section 14.10.1, “iOS Setup Instructions”](#)
2. [Section 14.10.2, “Android Setup Instructions”](#)
3. [Section 14.10.3, “OAuth Setup \(Simulator and REST API Access\)”](#)
4. [Section 14.10.5, “The Code”](#)

OAuth Setup is required for using Google Sign-In in the simulator, and for accessing other Google APIs in Android.

14.10.1. iOS Setup Instructions

Short Version

Go to [the Google Developer Portal](#) [<https://developers.google.com/mobile/add>], follow the steps to create an App, and enable Google Sign-In, and download the GoogleService-Info.plist file. Then copy this file into your project’s native/ios directory.

Long Version

Point your browser to [this page](#) [<https://developers.google.com/mobile/add>].

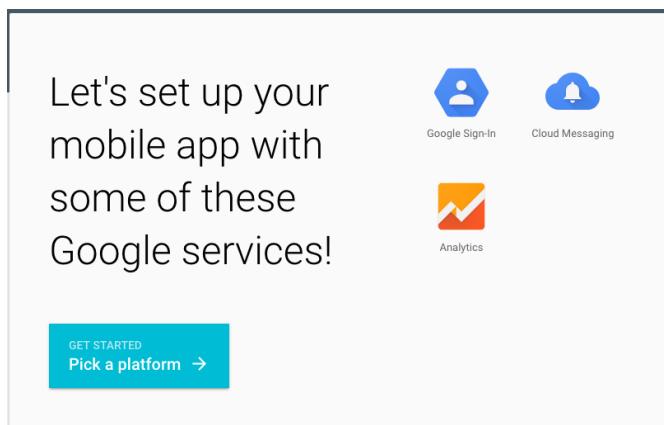


Figure 430. Set up mobile app form on Google

Click on the "Getting Started" button.



Figure 431. Getting started button

Then click "iOS App"

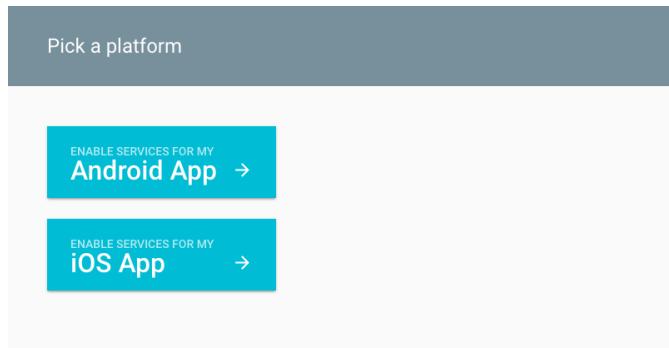


Figure 432. Pick a platform

Now enter an app name and the bundle ID for your app on the form below. The app name doesn't necessary need to match your app's name, but the bundle ID should match the package name of your app.

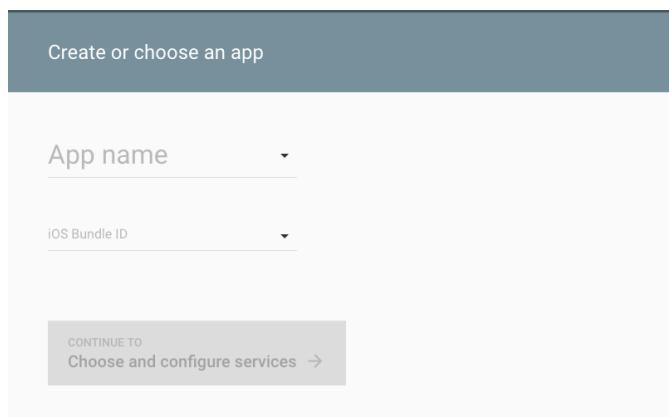


Figure 433. Create or Choose App

Select your country, and then click the "Choose and Configure Services" button.



Figure 434. Choose and Configure Services

You'll be presented with the following screen

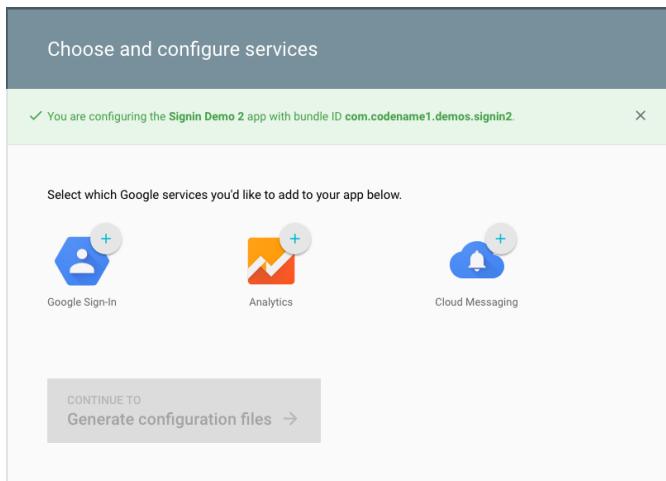


Figure 435. Choose and Configure Services form

Click on "Google Sign-In".

Then press the "Enable Google Sign-In" button that appears.

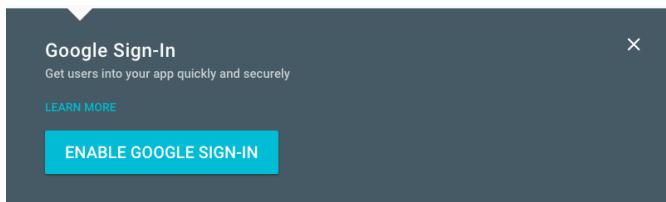


Figure 436. Enable Google Sign-In

You should then be presented with another button to "Generate Configuration Files" as shown below



Figure 437. Generate Configuration Files

Finally you will be presented with a button to "Download GoogleService-Info.plist".



Figure 438. Download GoogleService-Info.plist file

Press this button to download the GoogleService-Info.plist file. Then copy this into the "native/ios" directory of your Codename One project.

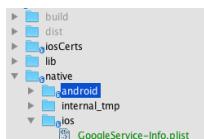


Figure 439. Project file structure after placing the GoogleService-Info.plist into the native/ios directory

At this point, your app should be able to use Google Sign-In. Notice that we don't require any build hints. Only that the GoogleService-Info.plist file is added to the project's native/ios directory.

14.10.2. Android Setup Instructions

Short Version

Go to the [Google Developer Portal](https://developers.google.com/mobile/add) [<https://developers.google.com/mobile/add>], follow the steps to create an App, and enable Google Sign-In, and download the google-services.json file. Then copy this file into your project's native/android directory.

Long Version

Point your browser to [this page](https://developers.google.com/mobile/add) [<https://developers.google.com/mobile/add>].

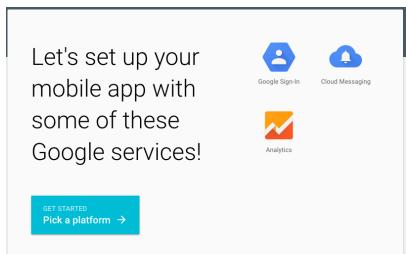
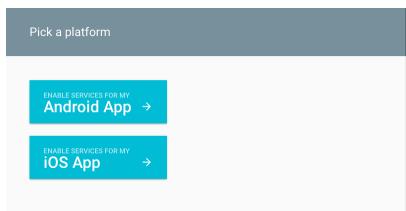


Figure 440. Set up mobile app form on Google

Click on the "Getting Started" button.



Then click "Android App"



Now enter an app name and the platform for your app on the form below. The app name doesn't necessary need to match your app's name, but the package name should match the package name of your app.

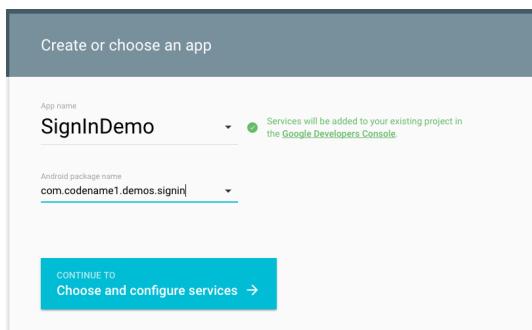


Figure 441. Create or Choose App

Select your country, and then click the "Choose and Configure Services" button.

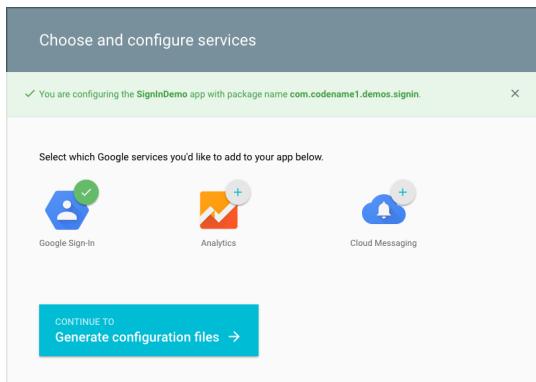


Figure 442. Choose and Configure Services

Click on "Google Sign-In"

Then you'll be presented with a field to enter the Android Signing Certificate SHA-1.

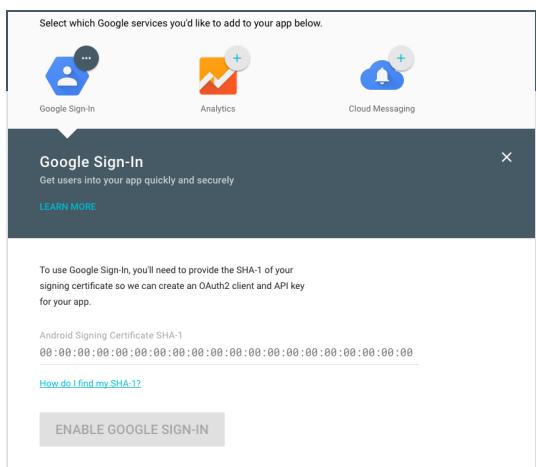


Figure 443. Android Signing Certificate SHA-1

The value that you enter here should be obtained from the certificate that you are using to build your app. You can use the **keytool** app that is distributed with the JDK to extract this value.

```
$ keytool -exportcert -alias myAlias -keystore /path/to/my-keystore.keystore -list -v
```

The snippet above assumes that your keystore is located at `/path/to/my-keystore.keystore`, and the certificate alias is "myAlias". You'll be prompted to enter the password for your keystore, then the output will look something like:

```

Alias name: myAlias
Creation date: 22-Jan-2014
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=My Own Company Corp., OU=, O=, L=Vancouver, ST=British Columbia, C=CA
Issuer: CN=My Own Company Corp., OU=, O=, L=Vancouver, ST=British Columbia, C=CA
Serial number: 56b2fd42
Valid from: Wed Jan 22 12:23:50 PST 2014 until: Tue Feb 16 12:23:50 PST 2055
Certificate fingerprints:
MD5: 98:F9:34:5B:B5:1A:14:2D:3C:5D:F4:92:D2:73:30:6B
SHA1: 76:BA:AA:11:A9:22:42:24:93:82:6D:33:7E:48:BC:AF:45:4D:79:B0
SHA256: 3D:04:33:67:6A:13:FF:4F:EE:E8:C9:7D:D2:CC:DF:70:33:E1:90:44:BF:22:B6:96:11:C7:00:67:8D:CD:53:BC
Signature algorithm name: SHA256withRSA
Version: 3

```

Extensions:

```

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C2 A0 48 AA 60 BA DD E3  0C 3F 00 B4 2C D5 92 A5  ..H.`.....D...
0010: 31 16 EF A2                               1...
]
]

```

You will be interested in SHA1 fingerprint. In the snippet above, the SHA1 fingerprint is:

```
76:BA:AA:11:A9:22:42:24:93:82:6D:33:7E:48:BC:AF:45:4D:79:B0
```

You would paste this value into the "Android Signing Certificate SHA-1" field in the web form.

After pasting that in, you'll see a new button with label "Enable Google Sign-in"

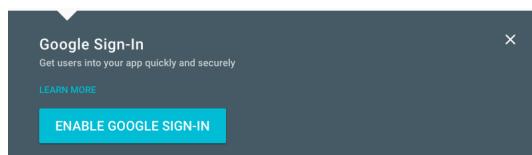


Figure 444. Enable Google Sign-In

Press this button and you'll be presented with another button to "Generate Configuration Files" as shown below



Figure 445. Generate Configuration Files

Finally you will be presented with a button to "Download google-services.json".



Figure 446. Download google-services.json file

Press this button to download the google-services.json file. Then copy this into the "native/android" directory of your Codename One project.

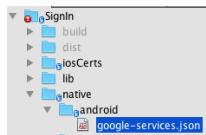


Figure 447. Project file structure after placing the GoogleService-Info.plist into the native/android directory

At this point, your app should be able to use Google Sign-In. Notice that we don't require any build hints. Only that the google-services.json file is added to the project's native/android directory.

If you want to access additional information about the logged in user using Google's REST APIs, you will require an OAuth2.0 client ID of type Web Application for this project as well. See [Section 14.10.3, “OAuth Setup \(Simulator and REST API Access\)”](#) for details.

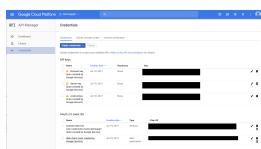
14.10.3. OAuth Setup (Simulator and REST API Access)

Getting Google Sign-In to work in the Codename One simulator requires an additional step after you've set up iOS and/or Android apps. The Simulator can't use the native Google Sign-In APIs, so it uses the standard Web Application OAuth2.0 API. In addition, the Android App requires a Web Application OAuth2.0 client ID to access additional Google REST APIs.

If you've set up the Google Sign-In API for either Android or iOS, then Google will have already automatically generated a Web Application OAuth2.0 client ID for you. You just need to provide the ClientID and ClientSecret to the `GoogleConnect` instance (in your java code).

Client ID, Client Secret and Redirect URL

1. Log into [the Google Cloud Platform API console](#) [<https://console.cloud.google.com/apis>].
2. Select your app from the drop-down-menu in the top bar
3. Click on "Credentials" in the left menu. You'll see a screen like this



4. Under the "OAuth2.0 Client IDs", find the row with "Web application" listed in the type column
5. Click the "Edit icon" for that row.
6. Make note of the "Client ID" and "Client Secret" on this page, as you'll need to add them to your Java source in the next step.
7. In the "Authorized redirect URIs" section, you will need to enter the URL to the page that the user will be sent to after a successful login. This page will only appear in the simulator for a split second, as Codename One's `BrowserComponent` will intercept this request to obtain the access token upon successful login. You can use any URL you like here, but it must match the value you give to `GoogleConnect.setRedirectURL()` in [Section 14.10.5, “The Code”](#).

Authorized redirect URIs
For OAuth 2.0 requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

https://www.codenameone.com/oauth2callback

http://www.example.com/oauth2callback

14.10.4. Javascript Setup Instructions

The Javascript port can use the same OAuth2.0 credentials as the simulator does. It doesn't require your Client Secret or redirect URL. It only requires your Client ID, which you can specify using the [GoogleConnect.setClientID\(\)](#) method.

14.10.5. The Code

```
 Login gc = GoogleConnect.getInstance();
 gc.setClientId("*****.apps.googleusercontent.com");
 gc.setRedirectURI("https://yourURL.com/");
 gc.setClientSecret("-----");

 // Sets a LoginCallback listener
 gc.setCallback(new LoginCallback() {
     public void loginSuccessful() {
         // we can now start fetching stuff from Google+!
     }

     public void loginFailed(String errorMessage) {}
 });

 // trigger the login if not already logged in
 if(!gc.isUserLoggedIn()){
     gc.doLogin();
 } else {
     // get the token and now you can query the Google API
     String token = gc.getAccessToken().getToken();
     // NOTE: On Android, this token will be null unless you provide valid
     // client ID and secrets.
 }
```



The client ID and client secret values here are the ones from your [OAuth2.0 Web Application](#).



The **Client ID** and **Client Secret** values are used on both the Simulator and on Android. On simulator these values are required for login to work at all. On Android these values are required to obtain an access token to query the Google API further using its various REST APIs. If you do not include these values on Android, login will still work, but `gc.getAccessToken().getToken()` will return `null`.

14.11. Lead Component

Codename One has two basic ways to create new components:

1. Subclass a **Component** override **paint**, implement event callbacks etc.
2. Compose multiple components into a new component, usually by subclassing a **Container**.

Components such as **Tabs** [<https://www.codenameone.com/javadoc/com/codename1/ui/Tabs.html>] subclass **Container** which make a lot of sense for that component since it is physically a **Container**.

However, components like **MultiButton** [<https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html>], **SpanButton** [<https://www.codenameone.com/javadoc/com/codename1/components/SpanButton.html>] & **SpanLabel** [<https://www.codenameone.com/javadoc/com/codename1/components/SpanLabel.html>] don't necessarily seem like the right candidate for compositing but they are all **Container** subclasses.

Using a **Container** provides us a lot of flexibility in terms of layout & functionality for a specific component. **MultiButton** is a great example of that. It's a **Container** internally that is composed of 5 labels and a **Button**.

Codename One makes the **MultiButton** "feel" like a single button thru the use of **setLeadComponent(Component)** which turns the button into the "leader" of the component.

When a **Container** hierarchy is placed under a leader all events within the hierarchy are sent to the leader, so if a label within the lead component receives a pointer pressed event this event will really be sent to the leader.

E.g. in the case of the **MultiButton** the internal button will receive that event and send the action performed event, change the state etc.

This creates some potential issues for instance in **MultiButton**:

```
myMultiButton.addActionListener((e) -> {
    if(e.getComponent() == myMultiButton) {
        // this won't occur since the source component is really a button!
    }
    if(e.getActualComponent() == myMultiButton) {
        // this will happen...
    }
});
```

The leader also determines the style state, so all the elements being lead are in the same state. E.g. if the the button is pressed all elements will display their pressed states, notice that they will do so with their own styles but they will each pick the pressed version of that style so a **Label** UIID within a lead component in the pressed state would return the Pressed state for a **Label** not for the **Button**.

This is very convenient when you need to construct more elaborate UI's and the cool thing about it is that you can do this entirely in the designer which allows assembling containers and defining the lead component inside the hierarchy.

E.g. the **SpanButton** class is very similar to this code:

```
public class SpanButton extends Container {  
    private Button actualButton;  
    private TextArea text;  
  
    public SpanButton(String txt) {  
        setUIID("Button");  
        setLayout(new BorderLayout());  
        text = new TextArea(getUIManager().localize(txt, txt));  
        text.setUIID("Button");  
        text.setEditable(false);  
        text.setFocusable(false);  
        actualButton = new Button();  
        addComponent(BorderLayout.WEST, actualButton);  
        addComponent(BorderLayout.CENTER, text);  
        setLeadComponent(actualButton);  
    }  
  
    public void setText(String t) {  
        text.setText(getUIManager().localize(t, t));  
    }  
  
    public void setIcon(Image i) {  
        actualButton.setIcon(i);  
    }  
  
    public String getText() {  
        return text.getText();  
    }  
  
    public Image getIcon() {  
        return actualButton.getIcon();  
    }  
  
    public void addActionListener(ActionListener l) {  
        actualButton.addActionListener(l);  
    }  
  
    public void removeActionListener(ActionListener l) {  
        actualButton.removeActionListener(l);  
    }  
}
```

14.11.1. Blocking Lead Behavior

The **Component** class has two methods that allow us to exclude a component from lead behavior:

`setBlockLead(boolean) & isBlockLead()`.

Effectively when you have a `Component` within the lead hierarchy that you would like to treat differently from the rest you can use this method to exclude it from the lead component behavior while keeping the rest in line...

This should have no effect if the component isn't a part of a lead component.

The sample below is based on the `Accordion` component which uses a lead component internally.

```
Form f = new Form("Accordion", new BorderLayout());
Accordion accr = new Accordion();
f.getToolbar().addMaterialCommandToRightBar("", FontImage.MATERIAL_ADD, e -> addEntry(accr));
addEntry(accr);
f.add(BorderLayout.CENTER, accr);
f.show();

void addEntry(Accordion accr) {
    TextArea t = new TextArea("New Entry");
    Button delete = new Button();
    FontImage.setMaterialIcon(delete, FontImage.MATERIAL_DELETE);
    Label title = new Label(t.getText());
    t.addActionListener(ee -> title.setText(t.getText()));
    delete.addActionListener(ee -> {
        accr.removeContent(t);
        accr.animateLayout(200);
    });
    delete.setBlockLead(true);
    delete.setUUID("Label");
    Container header = BorderLayout.center(title).
        add(BorderLayout.EAST, delete);
    accr.addContent(header, t);
    accr.animateLayout(200);
}
```

This allows us to add/edit entries but it also allows the delete button above to actually work separately. Without a call to `setBlockLead(true)` the delete button would cat as the rest of the accordion title.



Figure 448. Accordion with delete button entries that work despite the surrounding lead

14.12. Pull To Refresh

Pull to refresh is the common UI paradigm that Twitter popularized where the user can pull down the form/container to receive an update. Adding this to Codename One couldn't be simpler!

Just invoke `addPullToRefresh(Runnable)` on a scrollable container (or form) and the runnable method will be invoked when the refresh operation occurs.



Pull to refresh is implicitly implements in the `InfiniteContainer`

```
Form hi = new Form("Pull To Refresh", BoxLayout.y());
hi.getContentPane().addPullToRefresh(() -> {
    hi.add("Pulled at " + L10NManager.getInstance().formatDateTimeShort(new Date()));
});
hi.show();
```

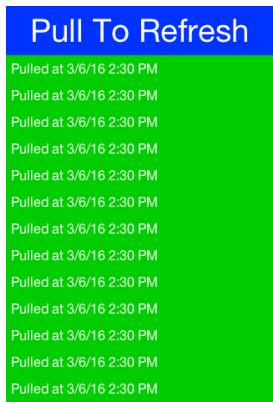


Figure 449. Pull to refresh demo

14.13. Running 3rd Party Apps Using Display's execute

The `Display` [<https://www.codenameone.com/javadoc/com/codenname1/ui/Display.html>] class's `execute` method allows us to invoke a URL which is bound to a particular application.

This works rather well assuming the application is installed. E.g. `this list` [http://wiki.akosma.com/IPhone_URL_Schemes] contains a set of valid URL's that can be used on iOS to run common applications and use builtin functionality.

Some URL's might not be supported if an app isn't installed, on Android there isn't much that can be done but iOS has a `canOpenURL` method for Objective-C.

On iOS you can use the `Display.canExecute()` method which returns a `Boolean` instead of a `boolean` which allows us to support 3 result states:

1. `Boolean.TRUE` - the URL can be executed
2. `Boolean.FALSE` - the URL isn't supported or the app is missing
3. `null` - we have no idea whether the URL will work on this platform.

The sample below launches a "godfather" search on IMDB only when this is sure to work (only on iOS currently). We can actually try to search in the case of null as well but this sample plays it safe by using the http link which is sure to work:

```
Boolean can = Display.getInstance().canExecute("imdb:///find?q=godfather");
if(can != null && can) {
    Display.getInstance().execute("imdb:///find?q=godfather");
} else {
    Display.getInstance().execute("http://www.imdb.com");
}
```

14.14. Automatic Build Hint Configuration

We try to make Codename One "seamless", this expresses itself in small details such as the automatic detection of permissions on Android etc. The build servers go a long way in setting up the environment as intuitive. But it's not enough, build hints are often confusing and obscure. It's hard to abstract the mess that is native mobile OS's and the odd policies from Apple/Google...

A good example for a common problem developers face is location code that doesn't work in iOS. This is due to the `ios.locationUsageDescription` build hint that's required. The reason that build hint was added is a requirement by Apple to provide a description for every app that uses the location service.

To solve this sort of used case we have two API's in `Display`:

```
/**
 * Returns the build hints for the simulator, this will only work in the debug environment and it's
 * designed to allow extensions/API's to verify user settings/build hints exist
 * @return map of the build hints that isn't modified without the codename1.arg. prefix
 */
public Map<String, String> getProjectBuildHints() {}

/**
 * Sets a build hint into the settings while overwriting any previous value. This will only work in the
 * debug environment and it's designed to allow extensions/API's to verify user settings/build hints exist.
 * Important: this will throw an exception outside of the simulator!
 * @param key the build hint without the codename1.arg. prefix
 * @param value the value for the hint
 */
public void setProjectBuildHint(String key, String value) {}
```

Both of these allow you to detect if a build hint is set and if not (or if it's set incorrectly) set its value...

So if you will use the location API from the simulator and you didn't define `ios.locationUsageDescription` Codename One will implicitly define a string there. The cool thing is that you will now see that string in your settings and you would be able to customize it easily.

However, this gets way better than just that trivial example!

The real value is for 3rd party cn1lib authors. E.g. Google Maps or Parse. They can inspect the build hints in the simulator and show an error in case of a misconfiguration. They can even show a setup UI. Demos that need special keys in place can force the developer to set them up properly before continuing.

14.15. Easy Thread

Working with threads is usually ranked as one of the least intuitive and painful tasks in programming. This is such an error prone task that some platforms/languages took the route of avoiding threads entirely. I needed to convert some code to work on a separate thread but I still wanted the ability to communicate and transfer data from that thread.

This is possible in Java but non-trivial, the thing is that this is relatively easy to do in Codename One with tools such as `callSerially` I can let arbitrary code run on the EDT. Why not offer that to any random thread?

That's why I created `EasyThread` which takes some of the concepts of Codeame One's threading and makes them more accessible to an arbitrary thread. This way you can move things like resource loading into a separate thread and easily synchronize the data back into the EDT as needed...

Easy thread can be created like this:

```
EasyThread e = EasyThread.start("ThreadName");
```

You can just send a task to the thread using:

```
e.run(() -> doThisOnTheThread());
```

But it gets better, say you want to return a value:

```
e.run((success) -> success.onSuccess(doThisOnTheThread()), (myResult) -> onEDTGotResult(myResult));
```

Lets break that down... We ran the thread with the success callback on the new thread then the callback got invoked on the EDT as a result. So this code `(success) → success.onSuccess(doThisOnTheThread())` ran off the EDT in the thread and when we invoked the `onSuccess` callback it sent it asynchronously to the EDT here: `(myResult) → onEDTGotResult(myResult)`.

These asynchronous calls make things a bit painful to wade thru so instead I chose to wrap them in a simplified synchronous version:

```
EasyThread e = EasyThread.start("Hi");
int result = e.run(() -> {
    System.out.println("This is a thread");
    return 3;
});
```

There are a few other variants like `runAndWait` and there is a `kill()` method which stops a thread and releases its resources.

14.16. Mouse Cursor

Codename one can change the mouse cursor when hovering over specific areas to indicate resizability, movability etc. For obvious reasons this feature is only available in the desktop and JavaScript ports as the other ports rely mostly on touch interaction. The feature is off by default and needs to be enabled on a `Form` by using `Form.setEnableCursors(true);`. If you are writing a custom component that can use cursors such as `SplitPane` you can use:

```
@Override  
protected void initComponent() {  
    super.initComponent();  
    getComponentForm().setEnableCursors(true);  
}
```

Once this is enabled you can set the cursor over a specific region using `cmp.setCursor()` which accepts one of the cursor constants defined in `Component`.

14.17. Working With GIT

Working with GIT for storing Codename One projects isn't exactly a feature but since it is so ubiquitous we think it's important to have a common guideline.

When we first started committing to git we used something like this for netbeans projects:

```
*.jar  
nbproject/private/  
build/  
dist/  
lib/CodenameOne_SRC.zip
```

Removing the jars, build, private folder etc. makes a lot of sense but there are a few nuances that are missing here...

14.17.1. cn1lib's

You will notice we excluded the jars which are stored under lib and we exclude the Codename One source zip. But I didn't exclude cn1libs... That was an omission since the original project we committed didn't have cn1libs. But should we commit a binary file to git?

I don't know. Generally git isn't very good with binaries but cn1libs make sense. In another project that did have a cn1lib I did this:

```
*.jar  
nbproject/private/  
build/  
dist/  
lib/CodenameOne_SRC.zip  
lib/impl/  
native/internal_tmp/
```

The important lines are `lib/impl/` and `native/internal_tmp/`. Technically cn1libs are just zips. When you do a refresh libs they unzip into the right directories under `lib/impl` and `native/internal_tmp`. By excluding these directories we can remove duplicates that can result in conflicts.

14.17.2. Resource Files

Committing the res file is a matter of personal choice. It is committed in the git ignore files above but you can remove it. The res file is at risk of corruption and in that case having a history we can refer to, matters a lot.

But the resource file is a bit of a problematic file. As a binary file if we have a team working with it the conflicts can be a major blocker. This was far worse with the old GUI builder, that was one of the big motivations of moving into the new GUI builder which works better for teams.

Still, if you want to keep an eye of every change in the resource file you can switch on the `File → XML Team Mode` which should be on by default. This mode creates a file hierarchy under the `res` directory to match the res file you opened. E.g. if you have a file named `src/theme.res` it will create a matching `res/theme.xml` and also nest all the images and resources you use in the res directory.

That's very useful as you can edit the files directly and keep track of every file in git. However, this has two big drawbacks:

- It's flaky - while this mode works it never reached the stability of the regular res file mode
- It conflicts - the simulator/device are oblivious to this mode. So if you fetch an update you also need to update the res file and you might still have conflicts related to that file

Ultimately both of these issues shouldn't be a deal breaker. Even though this mode is a bit flaky it's better than the alternative as you can literally "see" the content of the resource file. You can easily revert and reapply your changes to the res file when merging from git, it's tedious but again not a deal breaker.

14.17.3. Eclipse Version

Building on the gitignore we have for NetBeans the eclipse version should look like this:

```
.DS_Store  
*.jar  
build/  
dist/  
lib/impl/  
native/internal_tmp/  
.metadata  
bin/  
tmp/  
*.tmp  
*.bak  
*.swp  
*.zip  
*~.nib  
local.properties  
.settings/  
.loadpath  
.recommenders  
.externalToolBuilders/  
*.launch  
*.pydevproject  
.cproject  
.factorypath  
.buildpath  
.project  
.classpath
```

14.17.4. IntelliJ/IDEA

```
.DS_Store  
*.jar  
build/  
dist/  
lib/impl/  
native/internal_tmp/  
*.zip  
.idea/**/workspace.xml  
.idea/**/tasks.xml  
.idea/dictionaries  
.idea/**/dataSources/  
.idea/**/dataSources.ids  
.idea/**/dataSources.xml  
.idea/**/dataSources.local.xml  
.idea/**/sqlDataSources.xml  
.idea/**/dynamic.xml  
.idea/**/uiDesigner.xml  
.idea/**/gradle.xml  
.idea/**/libraries  
*.iws  
/out/  
atlassian-ide-plugin.xml
```


15. Performance, Size & Debugging

15.1. Reducing Resource File Size

It's easy to lose track of size/performance when you are working within the comforts of a visual tool like the Codename One Designer. When optimizing resource files you need to keep in mind one thing: it's all about image sizes.



Images will take up 95-99% of the resource file size; everything else pales in comparison.

Like every optimization the first rule is to reduce the size of the biggest images which will provide your biggest improvements, for this purpose we introduced the ability to see image sizes in kilobytes. To launch that feature use the menu item **Images → Image Sizes (KB)** in the designer.

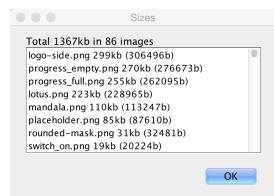


Figure 450. Image sizes window that allows us to find the biggest impact on our RAM/Storage

This produces a list of images sorted by size with their sizes. Often the top entries will be multi-images, which include HD resolution values that can be pretty large. These very high-resolution images take up a significant amount of space!

Just going to the multi-images, selecting the unnecessary resolutions & deleting these images can save significant amounts of space:

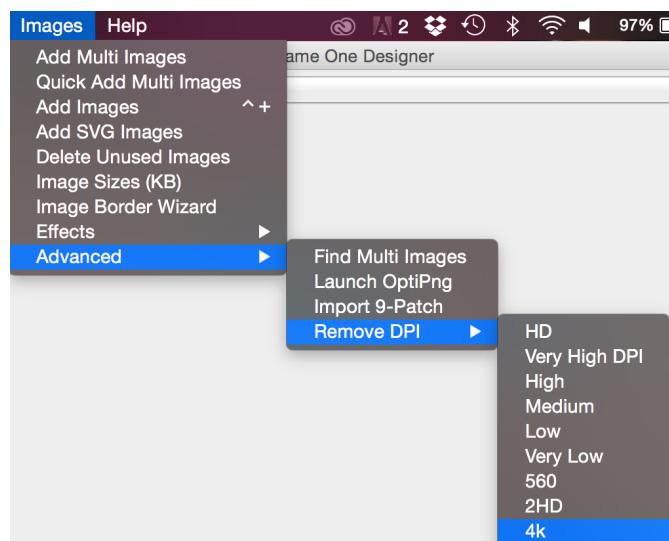


Figure 451. Removing unused DPI's



You can see the size in KB at the top right side in the designers image viewer

Applications using the old GUI builder can use the **Images → Delete Unused Images** menu option (it's also under the Images menu). This tool allows detecting and deleting images that aren't used

within the theme/GUI.

If you have a very large image that is opaque you might want to consider converting it to JPEG and replacing the built in PNG's. Notice that JPEG's work on all supported devices and are typically smaller.

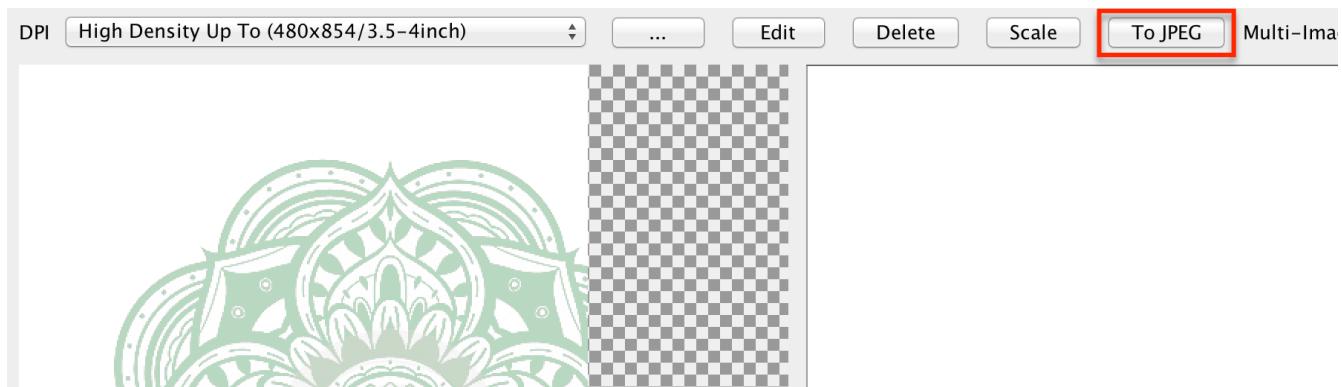


Figure 452. Convert a MultiImage to use JPEGs instead of PNGs

You can use the excellent [OptiPng](http://optipng.sourceforge.net/) [<http://optipng.sourceforge.net/>] tool to optimize image files right from the Codename One designer. To use this feature you need to install OptiPng then select **Images** → **Launch OptiPng** from the menu. Once you do that the tool will automatically optimize all your PNG's.

When faced with size issues make sure to check the size of your res file, if your JAR file is large open it with a tool such as 7-zip and sort elements by size. Start reviewing which element justifies the size overhead.

15.2. Improving Performance

There are quite a few things you can do as a developer in order to improve the performance and memory footprint of a Codename One application. This sometimes depends on specific device behaviors but some of the tips here are true for all devices.

The simulator contains some tools to measure performance overhead of a specific component and also detect EDT blocking logic. Other than that follow these guidelines to create more performance code:

- **Avoid round rect borders** - they have a huge overhead on all platforms. Use image borders instead (counter intuitively they are MUCH faster)
- **Avoid Gradients** - they perform poorly on most OS's. Use a background image instead
- **Use larger images** when tiling or building image borders, using a 1 pixel (or even a few pixels) wide or high image and tiling it repeatedly can be very expensive
- **Shrink resource file sizes** - Otherwise data might get collected by the garbage collector and reloading data might be expensive
- **Check that you don't have too many image lock misses** - this is discussed in the graphics section
- **On some platforms mutable images are slow** - mutable images are images you can draw on (using `getGraphics()`). On some platforms they perform quite badly (e.g. iOS) and should

generally be avoided. You can check if mutable images are fast in a platform using `Display.areMutableImagesFast()`

- * Make components either transparent or opaque * - a translucent component must paint its parent every time. This can be expensive. An opaque component might have margins that would require that we paint the parent so there is often overdraw in such cases (overdraw means the same pixel being painted twice).

15.3. Performance Monitor

The Performance Monitor tool can be accessible via the `Simulator → Performance Monitor` menu option in the simulator. This launches the following UI that can help you improve application performance:

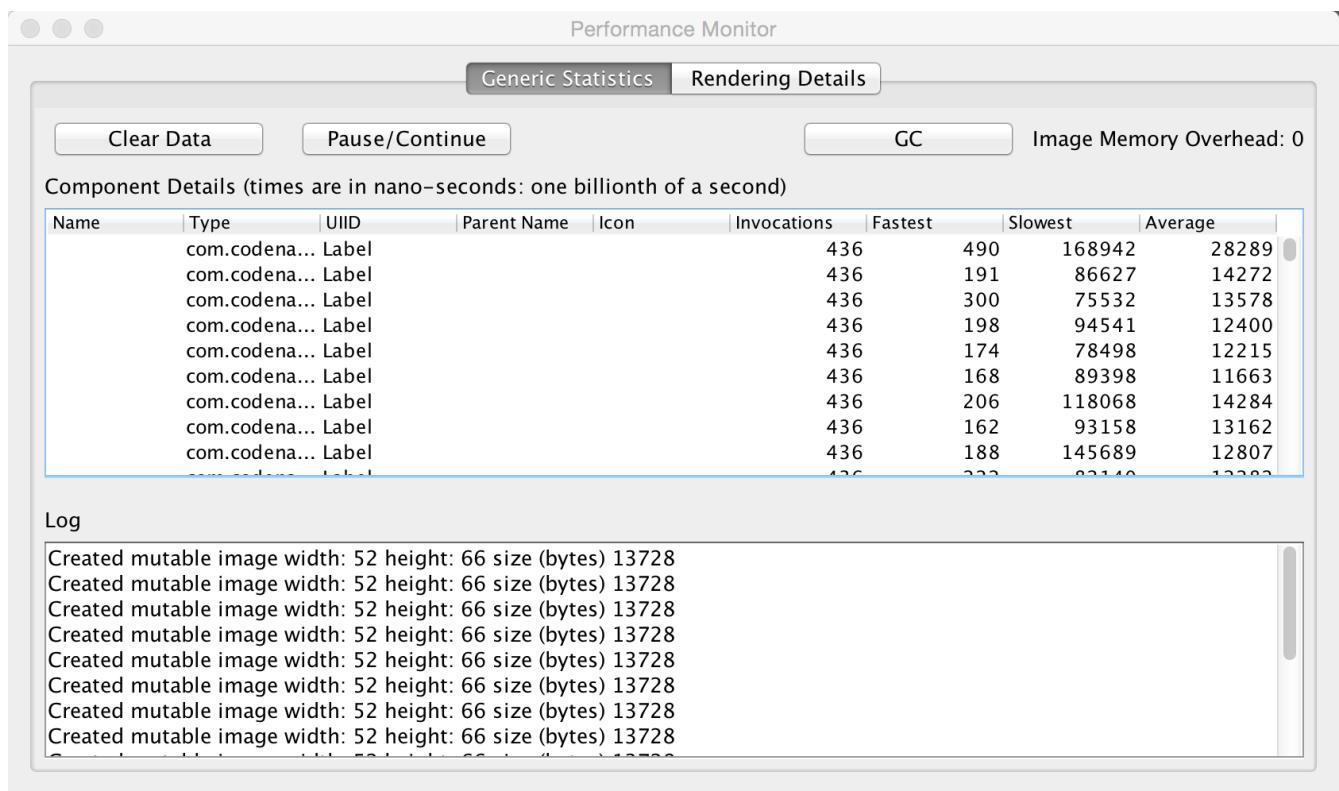


Figure 453. Main tab of the performance monitor: Logs and timings

The first tab of the performance monitor includes a table of the drawn components. Each entry includes the number of times it was drawn and the slowest/fastest and average drawing time.

This is useful if a `Form` is slow. You might be able to pinpoint it to a specific component using this tool.

The Log on the bottom includes debug related information. E.g. it warns about the usage of mutable images which might be slow on some platforms. This also displays warnings when an unlocked image is drawn etc.

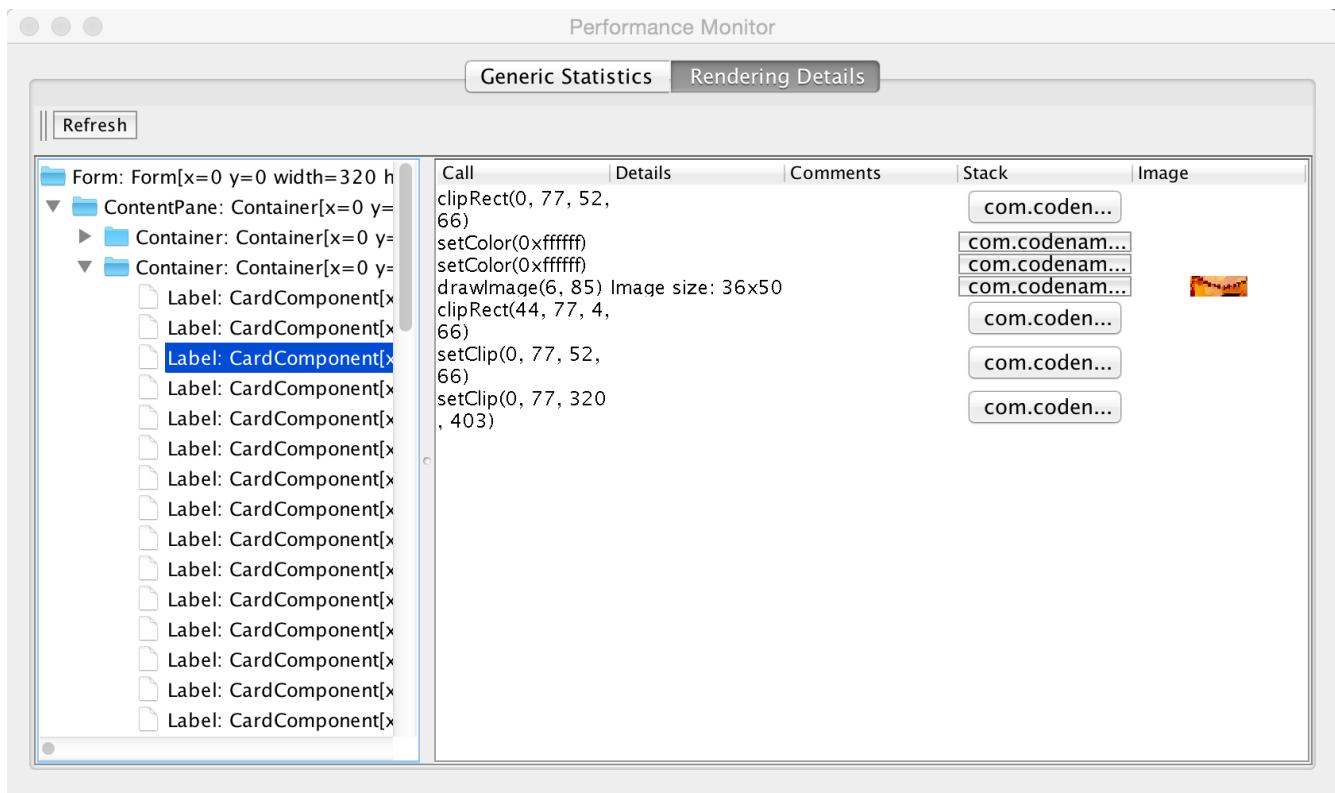


Figure 454. Rendering tree

The rendering tree view allows us to inspect the hierarchy painting. You can press the refresh button which will trigger the painting of the current **Form**. Every graphics operation is logged and so is the stack to it.

You can then inspect the hierarchy and see what was drawn by the various components. You can click the "stack" buttons to see the specific stack trace that lead to that specific drawing operation.

This is a remarkably powerful debugging tool as you can literally see "overdraw" within this tool. E.g if you see **fillRect** or similar API's invoked in the parent and then again and again in the children this could indicate a problem.



Android devices have a very nice overdraw debugging tool

15.4. Network Speed



Figure 455. Network speed tool

This feature is actually more useful for general debugging however it's sometimes useful to simulate a slow/disconnected network to see how this affects performance.

For this purpose the Codename One simulator allows you to slow down networking or even fake a disconnected network to see how your application handles such cases.

15.5. Debugging Codename One Sources

When you debug your app with our source code you can place breakpoints deep within Codename One and gain unique insight. You can also use the profilers and profile into Codename One to gain similar performance specific insight.

When you run into a bug or a missing feature you can push that feature/fix back to [Codename One](https://www.codenameone.com/) [<https://www.codenameone.com/>] using a pull request. Github makes that process trivial and in this new video and slides below we show you how. The steps to use the code are:

1. Signup for Github
2. Fork <http://github.com/codenameone/CodenameOne> and <http://github.com/codenameone/codenameone-skins> (also star and watch the projects for good measure).
3. Clone the git URL's from the projects into the IDE using the **Team** → **Git** → **Clone** menu option. Notice that you must deselect projects in the IDE for the menu to appear.
4. Download the cn1-binaries project from github [here](#) [<https://github.com/codenameone/cn1-binaries/archive/master.zip>].
5. Unzip the cn1-binaries project and make sure the directory has the name cn1-binaries. Verify that cn1-binaries, CodenameOne and codenameone-skins are within the same parent directory. In your own project remove the jars both in the build & run libraries section. Replace the build libraries with the [CodenameOne/CodenameOne](#) project. Replace the runtime libraries with the [CodenameOne/Ports/JavaSEPort](#) project.

This allows you to run the existing Codename One project with the Codename One source code and debug into Codename One. You can now also commit, push and send a pull request with the changes.

15.6. Device Testing Framework/Unit Testing

Codename One includes a built in testing framework and test recorder tool as part of the simulator. This allows developers to build both functional and unit test execution on top of Codename One. It even enables sending tests for execution on the device (pro-only feature).

To get started with the testing framework, launch the application and open the test recorder in the simulator menu.

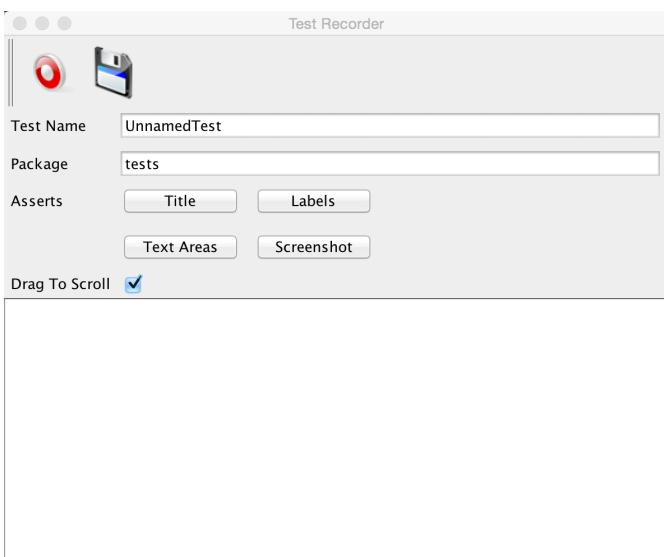


Figure 456. The test recorder tool in the simulator

Once you press record a test will be generate for you as you use the application.

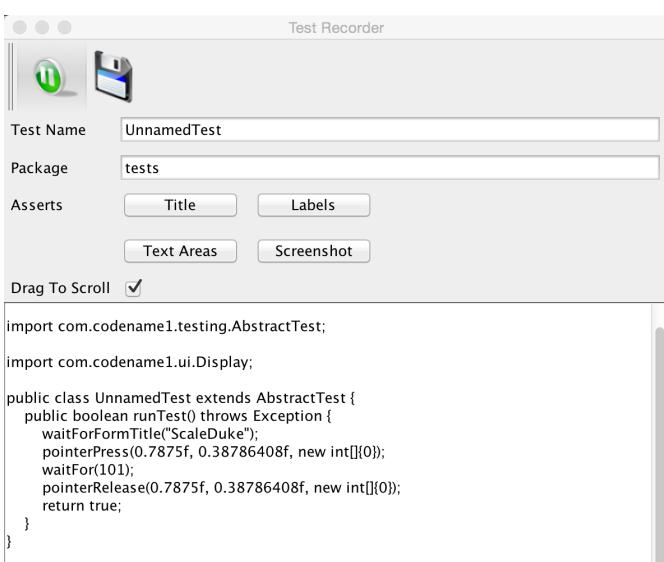


Figure 457. Test recording in progress, when done just press the save icon

You can build tests using the Codename One testing package to manipulate the Codename One UI programmatically and perform various assertions.

Unlike frameworks such as JUnit which assign a method per test, the Codename One test framework uses a class per test. This allows the framework to avoid reflection and thus allows it to work properly on the device.

15.7. EDT Error Handler and sendLog

Handling errors or exceptions in a deployed product is pretty difficult, most users would just throw away your app and some would give it a negative rating without providing you with the opportunity to actually fix the bug that might have happened.



Figure 458. Default error dialog

Google improved on this a bit by allowing users to submit stack traces for failures on Android devices but this requires the users approval for sending personal data which you might not need if you only want to receive the stack trace and maybe some basic application state (without violating user privacy).

For quite some time Codename One had a very powerful feature that allows you to both catch and report such errors, the error reporting feature uses the Codename One cloud which is exclusive for pro/enterprise users. Normally in Codename One we catch all exceptions on the EDT (which is where most exceptions occur) and just display an error to the user as you can see in the picture. Unfortunately this isn't very helpful to us as developers who really want to see the stack; furthermore we might prefer the user doesn't see an error message at all!

Codename One allows us to grab all exceptions that occur on the EDT and handle them using the method `addEdtErrorHandler` in the `Display` [<https://www.codenameone.com/javadoc/com/codenname1/ui/Display.html>] class. Adding this to the Log's ability to report errors directly to us and we can get a very powerful tool that will send us an email with information when a crash occurs!

This can be accomplished with a single line of code:

```
Log.bindCrashProtection(true);
```

We normally place this in the `init(Object)` method so all future on-device errors are emailed to you. Internally this method uses the `Display.getInstance().addEdtErrorHandler()` API to bind error listeners to the EDT. When an exception is thrown there it is swallowed (using `ActionEvent.consume()`). The `Log` data is then sent using `Log.sendLog()`.

To truly benefit from this feature we need to use the `Log` class for all logging and exception handling instead of API's such as `System.out`.

To log standard printouts you can use the `Log.p(String)` method and to log exceptions with their stack trace you can use `Log.e(Throwable)`.

15.8. Kitchen Sink Case Study

Performance is one of those vague subjects that is often taught by example.

During our debugging of the contacts demo that is a part of the new kitchen sink demo we noticed its performance was sub par. We assumed this was due to the implementation of `getAllContacts` & that there is nothing to do. While debugging another issue we noticed an anomaly during the loading of the contacts.

This led to the discovery that we are loading the same resource file over and over again for every single contact in the list!

In the new Contacts demo we have a share button for each contact, the code for constructing a `ShareButton` looks like this:

```

public ShareButton() {
    setUUID("ShareButton");
    FontImage.setMaterialIcon(this, FontImage.MATERIAL_SHARE);
    addActionListener(this);
    shareServices.addElement(new SMSShare());
    shareServices.addElement(new EmailShare());
    shareServices.addElement(new FacebookShare());
}

```

This seems reasonable until you realize that the constructors for `SMSShare`, `EmailShare` & `FacebookShare` load the icons for each of those...

These icons are in a shared resource file that we load and don't properly cache. The initial workaround was to cache this resource but a better solution was to convert this code:

```

public SMSShare() {
    super("SMS", Resources.getSystemResource().getImage("sms.png"));
}

```

Into this code:

```

public SMSShare() {
    super("SMS", null);
}

@Override
public Image getIcon() {
    Image i = super.getIcon();
    if(i == null) {
        i = Resources.getSystemResource().getImage("sms.png");
        setIcon(i);
    }
    return i;
}

```

This way the resource uses lazy loading as needed.

This small change boosted the loading performance and probably the general performance due to less memory fragmentation.

The lesson that we should learn every day is to never assume about performance...

15.8.1. Scroll Performance - Threads aren't magic

Another performance pitfall in this same demo came during scrolling. Scrolling was janky (uneven/unsmooth) right after loading finished would recover after a couple of minutes.

This relates to the images of the contacts.

To hasten the loading of contacts we load them all without images. We then launch a thread that iterates the contacts and loads an individual image for a contact. Then sets that image to the contact and replaces the placeholder image.

This performed well in the simulator but didn't do too well even on powerful mobile phones. We assumed this wouldn't be a problem because we used `Util.sleep()` to yield CPU time but that wasn't enough.

Often when we see performance penalty the response is: "move it to a separate thread". The problem is that this separate thread needs to compete for the same system resources and merge its changes back into the EDT. When we perform something intensive we need to make sure that the CPU isn't needed right now...

In this and past cases we solved this using a class member indicating the last time a user interacted with the UI.

Here we defined:

```
private long lastScroll;
```

Then we did this within the background loading thread:

```
// don't do anything while we are scrolling or animating
long idle = System.currentTimeMillis() - lastScroll;
while(idle < 1500 || contactsDemo.getAnimationManager().isAnimating() || scrollY != contactsDemo.getScrollY()) {
    scrollY = contactsDemo.getScrollY();
    Util.sleep(Math.min(1500, Math.max(100, 2000 - ((int)idle))));
    idle = System.currentTimeMillis() - lastScroll;
}
```

This effectively sleeps when the user interacts with the UI and only loads the images if the user hasn't touched the UI in a while.

Notice that we also check if the scroll changes, this allows us to notice cases like the animation of scroll winding down.

All we need to do now is update the `lastScroll` variable whenever user interaction is in place. This works for user touches:

```
parentForm.addPointerDraggedListener(e -> lastScroll = System.currentTimeMillis());
```

This works for general scrolling:

```
contactsDemo.addScrollListener(new ScrollListener() {
    int initial = -1;
    @Override
    public void scrollChanged(int scrollX, int scrollY, int oldscrollX, int oldscrollY) {
        // scrolling is sensitive on devices...
        if(initial < 0) {
            initial = scrollY;
        }
        lastScroll = System.currentTimeMillis();
        ...
    }
});
```



Due to technical constraints we can't use a lambda in this specific case...

16. Advanced Topics/Under the Hood

16.1. Sending Arguments To The Build Server

When sending a build to the server we can provide additional parameters to the build, which are incorporated into the build process on the server to "hint" on multiple different build time options.

These hints are often referred to as "build hints" or "build arguments", they are effectively very much like souped up compiler flags that you can use to tune the build server's behavior. This is useful for fast iteration on new functionality without building plugin UI for every change. This is also useful for exposing very low level behavior such as customizing the Android manifest XML or the iOS plist.

You can set these hints by right clicking the project in the IDE and selecting **Codename One** → **Codename One Settings** → **Build Hints**. The hints use the **key=value** style of data.

Build Hints		Delete	Up	Down
Key	Value			
android.debug	false	x		
android.newPeer	false	x		
android.release	true	x		
ios.application_exits	false	x		
ios.dsym	false	x		
ios.includePush	false	x		
ios.interface_orientati	faceOrientationPortraitUpsideDown:UIInterfaceOrientationLandscapeLeft:UIInterfaceOrientationLandscapeRight	x		
ios.prerendered_icon	false	x		
ios.project_type	ios	x		
ios.statusbar_hidden	false	x		
ios.testFlight	false	x		
jcme.nativeThemeCo	0	x		
rim.abbreviation	false	x		
vserv.category	29	x		
vserv.scaleMode	false	x		
vserv.transition	20000	x		
win.ver	8	x		

Figure 459. The build hints UI in Codename One Settings

You can set the build hints in the `codenameone_settings.properties` file directly notice that when you do that all settings need to start with the `codename1.arg.` prefix. When editing the properties file directly we would need to define something like `android.debug=true` as `codename1.arg.android.debug=true`.

Here is the current list of supported arguments, notice that build hints are added all the time so consult the discussion forum if you don't find what you need here:

Table 10. Build hints

Name	Description
android.debug	true/false defaults to true - indicates whether to include the debug version in the build
android.release	true/false defaults to true - indicates whether to include the release version in the build
android.installLocation	Maps to android:installLocation manifest entry defaults to auto. Can also be set to internalOnly or preferExternal.

Name	Description
android.gradle	Deprecated, this mode is no longer supported. true/false defaults to false prior to 3.3 and true after. Uses Gradle instead of Ant to build the Android app
android.xapplication	defaults to an empty string. Allows developers of native Android code to add text within the application block to define things such as widgets, services etc.
android.permission.PERMISSION_NAME	true/false Whether to include a particular permission. Use of these build hints is preferred to <code>android.xpermissions</code> since they avoid possible conflicts with libraries. See Android's Manifest.permission docs [https://developer.android.com/reference/android/Manifest.permission.html] for a full list of permissions.
android.permission.PERMISSION_NAME.maxSdkVersion	Will be translated to the <code>maxSdkVersion</code> attribute of the <code><uses-permission></code> tag for the corresponding <code>android.permission.PERMISSION_NAME</code> build hint. (Optional)
android.permission.PERMISSION_NAME.required	true/false Will be translated to the <code>required</code> attribute of the <code><uses-permission></code> tag for the corresponding <code>android.permission.PERMISSION_NAME</code> build hint. (Optional)
android.xpermissions	additional permissions for the Android manifest
android.xintent_filter	Allows adding an intent filter to the main android activity
android.activity.launchMode	Allows explicitly setting the <code>android:launchMode</code> attribute of the main activity in android. Default is "singleTop", but for some applications you may need to change this behaviour. In particular, apps that are meant to open a file type will need to set this to "singleTask". See Android docs for the activity element [https://developer.android.com/guide/topics/manifest/activity-element.html] for more information about the <code>android:launchMode</code> attribute.
android.licenseKey	The license key for the Android app, this is required if you use in-app-purchase on Android
android.stack_size	Size in bytes for the Android stack thread

Name	Description
android.statusbar_hidden	true/false defaults to false. When set to true hides the status bar on Android devices.
android.facebook_permissions	Permissions for Facebook used in the Android build target, applicable only if Facebook native integration is used.
android.googleAdUnitId	Allows integrating admob/google play ads, this is effectively identical to google.adUnitId but only applies to Android
android.googleAdUnitTestDevice	Device key used to mark a specific Android device as a test device for Google Play ads defaults to C6783E2486F0931D9D09FABC65094FDF
android.includeGPlayServices	Deprecated, please android.playService.*! Indicates whether Goolge Play Services should be included into the build, defaults to false but that might change based on the functionality of the application and other build hints. Adding Google Play Services support allows us to use a more refined location implementation and invoke some Google specific functionality from native code.
android.playService.plus, android.playService.auth, android.playService.base, android.playService.identity, android.playService.indexing, android.playService.appInvite, android.playService.analytics, android.playService.cast, android.playService.gcm, android.playService.drive, android.playService.fitness, android.playService.location, android.playService.maps, android.playService.ads, android.playService.vision, android.playService.nearby, android.playService.panorama, android.playService.games, android.playService.safetynet, android.playService.wallet, android.playService.wearable	Allows including only a specific play services library portion. Notice that this setting conflicts with the deprecated android.includeGPlayServices and only works with the gradle build (which is on by default but can be toggled using android.gradle). If none of the services are defined to true then plus, auth, base, analytics, gcm, location, maps & ads will be set to true. If one or more of the android.playService entries are defined to something then all entries will default to false.

Name	Description
android.playServicesVersion	The version number of play services to build against. Experimental. Use with caution as building against versions other than the server default may introduce incompatibilities with some Codename One APIs.
xxx.minPlayServicesVersion	This is a special case build hint. You can use any prefix to the build hint and the convention is to use your cn1lib name. It's identical to android.minPlayServicesVersion with the exception that the "highest version wins". That way if your cn1lib requires play services 9+ and uses: myLib.minPlayServicesVersion=9.0.0 and another library has otherLib.minPlayServicesVersion=10.0.0 then play services will be 10.0.0
android.multidex	Boolean true/false defaults to false. Multidex allows Android binaries to reference more than 65536 methods. This slows builds a bit so we have it off by default but if you get a build error mentioning this limit you should turn this on.
android.headphoneCallback	Boolean true/false defaults to false. When set to true it assumes the main class has two methods: headphonesConnected & headphonesDisconnected which it invokes appropriately as needed
android.gpsPermission	Indicates whether the GPS permission should be requested, it is auto-detected by default if you use the location API. However, some code might want to explicitly define it
android.asyncPaint	Boolean true/false defaults to true. Toggles the Android pipeline between the legacy pipeline (false) and new pipeline (true)
android.stringsXml	Allows injecting additional entries into the strings.xml file using a value that includes something like this `<string name="key1">value1</string><string name="key2">value2</string>`
android.supportV4	Boolean true/false defaults to false but that can change based on usage (e.g. push implicitly activates this). Indicates whether the android support v4 library should be included in the build

Name	Description
android.style	Allows injecting additional data into the <code>styles.xml</code> file right before the closing resources tag
android.cusom_layout1	Applies to any number of layouts as long as they are in sequence (e.g. android.cusom_layout2, android.cusom_layout3 etc.). Will write the content of the argument as a layout xml file and give it the name <code>cusom_layout1.xml</code> onwards. This can be used by native code to work with XML files
android.keyboardOpen	Boolean true/false defaults to true. Toggles the new async keyboard mode that leaves the keyboard open while we move between text components
android.versionCode	Allows overriding the auto generated version number with a custom internal version number specifically used for the xml attribute <code>android:versionCode</code>
android.captureRecord	Indicates whether the <code>RECORD_AUDIO</code> permission should be requested. Can be <code>enabled</code> or any other value to disable this option
android.nonconsumable	Comma delimited string of items that are non-consumable in the in-app-purchase API
android.removeBasePermissions	Boolean true/false defaults to false. Disables the builtin permissions specifically <code>INTERNET</code> permission (i.e. no networking...)
android.blockExternalStoragePermission	Boolean true/false defaults to false. Disables the external storage (SD card) permission
android.min_sdk_version	The minimum SDK required to run this app, the default value changes based on functionality but can be as low as 7. This corresponds to the XML attribute <code>android:minSdkVersion</code> .
android.mockLocation	Boolean true/false defaults to true. Toggles the mock location permission which is on by default, this allows easier debugging of Android device location based services
android.smallScreens	Boolean true/false defaults to true. Corresponds to the <code>android:smallScreens</code> XML attribute and allows disabling the support for very small phones
android.xapplication_attr	Allows injecting additional attributes into the <code>application`</code> tag in the Android XML

Name	Description
android.xactivity	Allows injecting additional attributes into the activity tag in the Android XML
android.streamMode	The mode in which the volume key should behave, defaults to OS default. Allows setting it to music for music playback apps
android.pushVibratePattern	Comma delimited long values to describe the push pattern of vibrate used for the setVibrate native method
android.enableProguard	Boolean true/false defaults to true. Allows disabling the proguard obfuscation even on release builds, notice that this isn't recommended
android.proguardKeep	Arguments for the keep option in proguard allowing us to keep a pattern of files e.g. -keep class com.mypackage.ProblemClass { *; }
android.shrinkResources	Boolean true/false defaults to false. Used only in conjunction with android.enableProguard. Strips out unused resources to reduce apk size. Since 7.0
android.sharedUserId	Allows adding a manifest attribute for the sharedUserId option
android.sharedUserLabel	Allows adding a manifest attribute for the sharedUserLabel option
android.targetSDKVersion	Indicates the Android SDK used to compile the Android build currently defaults to 21. Notice that not all targets will work since the source might have some limitations and not all SDK targets are installed on the build servers.
android.useAndroidX	Use Android X instead of support libraries. This will also run a find/replace on all source files to replace support libraries and artifacts with AndroidX equivalents.
android.theme	Light or Dark defaults to Light. On Android 4+ the default Holo theme is used to render the native widgets in some cases and this indicates whether holo light or holo dark is used. Currently this doesn't affect the Codename One theme but that might change in the future.
android.web_loading_hidden	true/false defaults to false - set to true to hide the progress indicator that appears when loading a web page on Android.

Name	Description
block_server_registration	true/false flag defaults to false. By default Codename One applications register with our server, setting this to true blocks them from sending information to our cloud. We keep this data for statistical purposes and intend to provide additional installation stats in the future.
facebook.appId	The application ID for an app that requires native Facebook login integration, this defaults to null which means native Facebook support shouldn't be in the app
gcm.sender_id	The Android/chrome push identifier, see the push section for more details
android.background_push_handling	Deliver push messages on Android when the app is minimized by setting this to "true". Default behaviour is to deliver the message only if the app is in the foreground when received, or after the user taps on the notification to open the app, if the app was in the background when the message was received.
ios.associatedDomains	Comma-delimited list of domains associated with this app. Since 6.0. Note that each domain should be prefixed by a supported prefix. E.g. "applinks:" or "webcredentials:". See Apple's documentation on Associated domains [https://developer.apple.com/documentation/security/password_autofill/setting_up_an_app_s_associated_domains?language=objc] for more information.
ios.bitcode	true/false defaults to false. Enables bitcode support for the build.
ios.debug.archs	Can be set to "arm64" to force iOS debug builds to be 64 bit. By default, debug builds are 32 bit in order to shorten build-times and maintain compatibility with older devices.
ios.distributionMethod	Specifies distribution type for debug iOS builds. This is generally used for enterprise or ad-hoc builds (using values "enterprise" and "ad-hoc" respectively).
ios.debug.distributionMethod	Specifies distribution type for debug iOS builds only. This is generally used for enterprise or ad-hoc builds (using values "enterprise" and "ad-hoc" respectively).

Name	Description
ios.release.distributionMethod	Specifies distribution type for release iOS builds only. This is generally used for enterprise or ad-hoc builds (using values "enterprise" and "ad-hoc" respectively).
ios.keyboardOpen	Flips between iOS keyboard open mode and auto-fold keyboard mode. Defaults to true which means the keyboard will remain open and not fold automatically when editing moves to another field.
ios.urlScheme	Allows intercepting a URL call using the syntax <code><string>urlPrefix<string></code>
ios.useAVKit	Use AVKit for video components on iOS rather than <code>MPMoviePlayerController</code> on iOS versions 8 through 12. iOS 13 will always use AVKit, and iOS 7 and lower will always use <code>MPMoviePlayerController</code> . Default value <code>false</code>
ios.teamId	Specifies the team ID associated with the iOS provisioning profile and certificate. Use <code>ios.debug.teamId</code> and <code>ios.release.teamId</code> to specify different team IDs for debug and release builds respectively.
ios.debug.teamId	Specifies the team ID associated with the iOS debug provisioning profile and certificate.
ios.release.teamId	Specifies the team ID associated with the iOS release provisioning profile and certificate.
ios.project_type	one of ios, ipad, iphone (defaults to ios). Indicates whether the resulting binary is targeted to the iphone only or ipad only. Notice that the IDE plugin has a "Project Type" combo box you should use under the iOS section.
ios.rpmalloc	<code>true/false</code> Use <code>rpmalloc</code> [https://github.com/rampantpixels/rpmalloc] instead of malloc/free for memory allocation in ParparVM. This will cause the deployment target to be changed to a minimum of iOS 8.0.
ios.statusbar_hidden	<code>true/false</code> defaults to false. Hides the iOS status bar if set to true.

Name	Description
ios.newStorageLocation	true/false defaults to false but defined on new projects as true by default. This changes the storage directory on iOS from using caches to using the documents directory which is more correct but might break compatibility. This is described in this issue [https://github.com/codenameone/CodenameOne/issues/1480]
ios.prerendered_icon	true/false defaults to false. The iOS build process adapts the submitted icon for iOS conventions (adding an overlay) that might not be appropriate on some icons. Setting this to true leaves the icon unchanged (only scaled).
ios.app_groups	Space-delimited list of app groups that this app belongs to as described in Apple's documentation [https://developer.apple.com/library/content/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html#/apple_ref/doc/uid/TP40011195-CH4-SW19]. These are added to the entitlements file with key com.apple.security.application-groups .
ios.keychainAccessGroup	Space-delimited list of keychain access groups that this app has access to as described in Apple's documentation [https://developer.apple.com/library/content/documentation/Security/Conceptual/keychainServConcepts/02concepts/concepts.html#/apple_ref/doc/uid/TP30000897-CH204-SW11]. These are added to the entitlements file with the key keychain-access-groups .
ios.application_exits	true/false (defaults to false). Indicates whether the application should exit immediately on home button press. The default is to exit, leaving the application running is only partially tested at the moment.
ios.blockScreenshotsOnEnterBackground	true/false (defaults to false). Indicates that app should prevent iOS from taking screenshots when app enters background. Described here [https://shannah.github.io/cn1-recipes/#_hiding_sensitive_data_when_entering_background].

Name	Description
ios.applicationQueriesSchemes	Comma separated list of url schemes that <code>canExecute</code> will respect on iOS. If the url scheme isn't mentioned here <code>canExecute</code> will return false starting with iOS 9. Notice that this collides with <code>ios.plistInject</code> when used with the <code><key>LSApplicationQueriesSchemes</key>…</code> value so you should use one or the other. E.g. to enable <code>canExecute</code> for a url like <code>myurl://xys</code> you can use: <code>myurl,myotherurl</code>
ios.themeMode	default/legacy/modern/auto (defaults to default). Default means you don't define a theme mode. Currently this is equivalent to legacy. In the future we will switch this to be equivalent to auto. legacy - this will behave like iOS 6 regardless of the device you are running on. modern - this will behave like iOS 7 regardless of the device you are running on. auto - this will behave like iOS 6 on older devices and iOS 7 on newer devices.
ios.interface_orientation	UIInterfaceOrientationPortrait by default. Indicates the orientation, one or more of (separated by colon :): UIInterfaceOrientationPortrait, UIInterfaceOrientationPortraitUpsideDown, UIInterfaceOrientationLandscapeLeft, UIInterfaceOrientationLandscapeRight. Notice that the IDE plugin has an "Interface Orientation" combo box you should use under the iOS section.
ios.xcode_version	The version of xcode used on the server. Defaults to 4.5; currently accepts 5.0 as an option and nothing else.
ios.multitasking	Set to true to enable iOS multitasking and split-screen support. This only works if <code>ios.xcode_version=9.2</code> .
java.version	Valid values include 5 or 8. Indicates the JVM version that should be used for server compilation, this is defined by default for newly created apps based on the Java 8 mode selection

Name	Description
javascript.inject_proxy	true/false (defaults to true) By default, the build server will configure the .war version of your app to use the bundled proxy servlet for HTTP requests (to get around same-origin restrictions on network requests). Setting this to false prevents this, causing the application to make network requests without a proxy.
javascript.inject.beforeHead	Content to be injected into the index.html file at the beginning of the <code><head></code> tag.
javascript.inject.afterHead	Content to be injected into the index.html file at the end of the <code><head></code> tag.
javascript.minifying	true/false (defaults to true). By default the javascript code is minified to reduce file size. You may optionally disable minification by setting javascript.minifying to false .
javascript.proxy.url	The URL to the proxy servlet that should be used for making network requests. If this is omitted, the .war version of the app will be set to use the bundled proxy servlet, and the .zip version of the app will be set to use no proxy. If javascript.inject_proxy is false , this build-hint will be ignored.
javascript.sourceFilesCopied	true/false (defaults to false). Setting this flag to true will cause available java source files to be included in the resulting .zip and .war files. These may be used by Chrome during debugging.
javascript.stopOnErrors	true/false (defaults to true). Cause javascript build to fail if there are warnings during the build. In some cases build warnings won't affect the running of the app. E.g. if the Javascript port is missing a method that the app depends on, but it isn't used in most of the app. Or if there is multithreaded code detected in static initializers, but that code-path isn't used by the app. Setting this to false may allow you to get past some build errors, but it might just result in runtime errors later on, which are much more difficult to debug. *This build hint is only available in Codename One 3.4 and later.
javascript.teavm.version	(Optional) The version of TeaVM to use for the build. Use caution , only use this property if you know what you are doing!

Name	Description
rim.askPermissions	true/false defaults to true. Indicates whether the user is prompted for permissions on Blackberry devices.
google.adUnitId	Allows integrating Admob/Google Play ads into the application see this [https://www.codenameone.com/blog/adding-google-play-ads.html]
rim.ignor_legacy	true/false defaults to false. When set to true the Blackberry build targets only 5.0 devices and newer and doesn't build the 4.x version. rim.nativeBrowser true/false defaults to false. Enables the native blackberry browser on OS 5 or higher. It is disabled by default since it might casue crashes on some cases.
rim.obfuscation	true/false defaults to false. Obfuscate the JAR before invoking the rimc compiler.
ios.plistInject	entries to inject into the iOS plist file during build.
ios.includePush	true/false (defaults to false). Whether to include the push capabilities in the iOS build. Notice that the IDE plugin has an "Include Push" check box you should use under the iOS section.
ios.newPipeline	Boolean true/false defaults to true. Allows toggling the OpenGL ES 2.0 drawing pipeline off to the older OGL ES 1.0 pipeline.
ios.headphoneCallback	Boolean true/false defaults to false. When set to true it assumes the main class has two methods: headphonesConnected & headphonesDisconnected which it invokes appropriately as needed
ios.facebook_permissions	Permissions for Facebook used in the Android build target, applicable only if Facebook native integration is used.
ios.applicationDidEnterBackground	Objective-C code that can be injected into the iOS callback method (message) applicationDidEnterBackground .
ios.enableAutoplayVideo	Boolean true/false defaults to false. Makes videos "auto-play" when loaded on iOS
ios.googleAdUnitId	Allows integrating admob/google play ads, this is effectively identical to google.adUnitId but only applies to iOS

Name	Description
ios.viewDidLoad	Objective-C code that can be injected into the iOS callback method (message) <code>viewDidLoad</code>
ios.googleAdUnitIdPadding	Indicates the amount of padding to pass to the Google ads placed at the bottom of the screen with <code>google.adUnitId</code>
ios.enableBadgeClear	Boolean true/false defaults to true. Clears the badge value with every load of the app, this is useful if the app doesn't manually keep track of number values for the badge
ios.gAppDelegateHeader	Objective-C code that can be injected into the iOS app delegate at the top of the file. E.g. if you need to include headers or make special imports for other injected code
ios.gAppDelegateBody	Objective-C code that can be injected into the iOS app delegate within the body of the file before the end. This only makes sense for methods that aren't already declared in the class
ios.beforeFinishLaunching	Objective-C code that can be injected into the iOS app delegate at the top of the body of the <code>didFinishLaunchingWithOptions</code> callback method
ios.afterFinishLaunching	Objective-C code that can be injected into the iOS app delegate at the bottom of the body of the <code>didFinishLaunchingWithOptions</code> callback method
ios.locationUsageDescription	This flag is required for iOS 8 and newer if you are using the location API. It needs to include a description of the reason for which you need access to the users location

Name	Description
ios.NSXXXUsageDescription	iOS privacy flags for using certain APIs. Starting with Xcode 8, you are required to add usage description strings for certain APIs. Find a full list of the available keys in Apple's docs [https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html]. Some relevant ones include <code>ios.NSCameraUsageDescription</code> , <code>ios.NSContactsUsageDescription</code> , <code>ios.NSLocationAlwaysUsageDescription</code> , <code>NSLocationUsageDescription</code> , <code>ios.NSMicrophoneUsageDescription</code> , <code>ios.NSPhotoLibraryAddUsageDescription</code> , <code>ios.NSSpeechRecognitionUsageDescription</code> , <code>ios.NSSiriUsageDescription</code>
ios.add_libs	A semicolon separated list of libraries that should be linked to the app in order to build it
ios.pods	A comma separated list of Cocoa Pods [https://cocoapods.org/] that should be linked to the app in order to build it. E.g. <code>AFNetworking ~> 2.6</code> , <code>ORStackView ~> 3.0</code> , <code>SwiftyJSON ~> 2.3</code>
ios.pods.platform	Sets the Cocoapods 'platform' for the Cocoapods. Some Cocoapods require a minimum platform level. E.g. <code>ios.pods.platform=7.0</code> .
ios.deployment_target	Sets the deployment target for iOS builds. This is the minimum version of iOS required by a device to install the app. E.g. <code>ios.deployment_target=8.0</code> . Default is '6.0'. Note: This build hint interacts with the <code>ios.rpmalloc</code> build hint. If <code>ios.deployment_target</code> is 8.0 or higher, ParparVM will use <code>rpmalloc</code> [https://github.com/rampantpixels/rpmalloc] by default. You can disable this default and revert back to using malloc/free by setting the <code>ios.rpmalloc=false</code> build hint.
ios.bundleVersion	Indicates the version number of the bundle, this is useful if you want to create a minor version number change for the beta testing support
ios.objC	Added the <code>-ObjC</code> compile flag to the project files which some native libraries require

Name	Description
ios.testFlight	Boolean true/false defaults to false and works only for pro accounts. Enables the testflight support in the release binaries for easy beta testing. Notice that the IDE plugin has a "Test Flight" check box you should use under the iOS section.
ios.generateSplashScreens	Boolean true/false defaults to false as of 5.0. Enable legacy generation of splash screen images for use when launching the app. These have been replaced now by the new launch storyboards.
desktop.width	Width in pixels for the form in desktop builds, will be doubled for retina grade displays. Defaults to 800.
desktop.height	Height in pixels for the form in desktop builds, will be doubled for retina grade displays. Defaults to 600.
desktop.adaptToRetina	Boolean true/false defaults to true. When set to true some values will be implicitly doubled to deal with retina displays and icons etc. will use higher DPI's
desktop.resizable	Boolean true/false defaults to true. Indicates whether the UI in the desktop build is resizable
desktop.fontSizes	Indicates the sizes in pixels for the system fonts as a comma delimited string containing 3 numbers for small,medium,large fonts.
desktop.theme	Name of the theme res file to use as the "native" theme. By default this is native indicating iOS theme on Mac and Windows Metro on Windows. If its something else then the app will try to load the file /themeName.res.
desktop.themeMac	Same as <code>desktop.theme</code> but specific to Mac OS
desktop.themeWin	Same as <code>desktop.theme</code> but specific to Windows
desktop.windowsOutput	Can be exe or msi depending on desired results
windows.extensions	Content to be embedded into the <code><Extensions></code> section of the Package.appxmanifest file for windows (UWP) builds.

Name	Description
win.vm32bit	true/false (defaults to false). Forces windows desktop builds to use the Win32 JVM instead of the 64 bit VM making them compatible with older Windows Machines. This is off by default at the moment because of a bug in JDK 8 update 112 that might cause this to fail for some cases
noExtraResources	true/false (defaults to false). Blocks codename one from injecting its own resources when set to true, the only effect this has is in slightly reducing archive size. This might have adverse effects on some features of Codename One so it isn't recommended.
j2me.iconSize	Defaults to 48x48. The size of the icon in the format of width x height (without the spacing).

16.2. Offline Build



Offline build is an enterprise feature

At this time Codename One supports iOS & Android targets for offline builds. We require an Enterprise grade subscription as explained in the sidebar.



If you signup for Enterprise and cancel you can still do the offline build. You won't be able to update the builder though

Why only Enterprise?

There are several reasons, the technical one is that offline builds are no panacea. Things fail. The support effort for offline builds is huge, as evidence despite the fact that all of our code is open source very few people bothered trying to compile it because of the complexities.

We don't think building offline is convenient and we always recommended avoiding it. When we build our own apps we use the cloud just like everyone else because it's surprisingly faster and more convenient...

However, some government and regulated industries have issues with SaaS delivered solutions and thus must use offline build. These organizations also require enterprise grade support for most cases and so it makes sense to bundle as an enterprise only solution.

16.2.1. Prerequisites for iOS Builds

You need the following installed tools/versions for Codename One's offline build process:

- Mac ideally with El Capitan, newer should work

- Xcode 7+ (but not 8+ at this time)
- Oracle's JDK 8
- Cocoapods - in the terminal type `sudo gem install cocoapods --pre`.
- xcodedproj - in the terminal type `sudo gem install xcodedproj`

16.2.2. Prerequisites for Android Builds

Android builds need the following:

- Android Studio
- Oracle's JDK 8
- Gradle version 2.11

16.2.3. Installation

To build offline you need to install the offline builder code which is a stripped down version of the build servers. When you install a version of the offline builder it maps to the time in which you downloaded it...

That means that features like versioned builds won't work. You can download/keep multiple offline builders and toggle between them which is similar in scope.

E.g. if you installed an offline builder then installed a newer version and the newer version has a bug you can revert to the old version. Notice that the older version might not have features that exist in a newer version.



Since installation requires an enterprise account, you might need to re-login in the Codename One Settings UI

To install an offline builder open the Codename One Settings UI by right clicking the project and selecting **Codename One** → **Codename One Settings**.

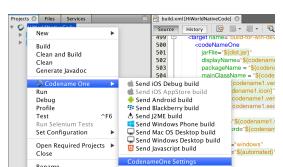


Figure 460. Open Codename One settings



Even though the settings are a part of a project, the offline build settings are global and apply to all the projects...

Once the Codename One settings UI launches select the **Offline Builds** entry:

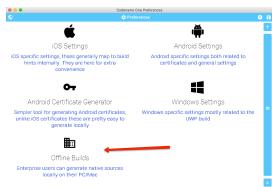


Figure 461. Offline build entry

This should launch the settings UI which would be blank the first time around:

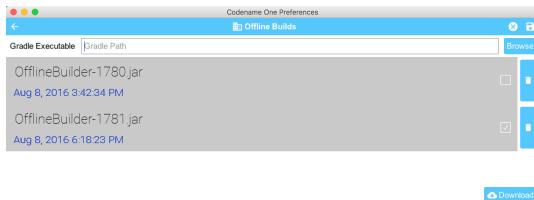


Figure 462. Offline builds setting UI

When you are in this form you can press the download button to download the current version from the build server. If there is no update nothing will happen. If there is the latest version will download and tag with a version number/date.

You can see/change the selected version in this UI. This allows building against an older version. You can also delete older builds to save space.

16.2.4. Building

Offline building is almost like building with the cloud. In the right click menu you can select one of the offline build targets as such:

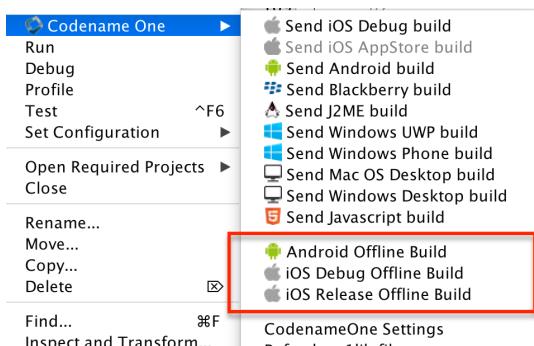


Figure 463. The offline build targets

Once selected build generates a project under the `build/and` or `build/iphone` respectively.

Open these directories in Android Studio or xcode to run/build in the native IDE to the device or native emulator/simulator.



Build deletes previous offline builds, if you want to keep the sources of a build you need to move it to a different directory!

To get this to work with Android Studio you will need one more step. You will need to configure Android studio to use your local version of gradle 2.11 by following these steps:

- Open the Android Studio preferences

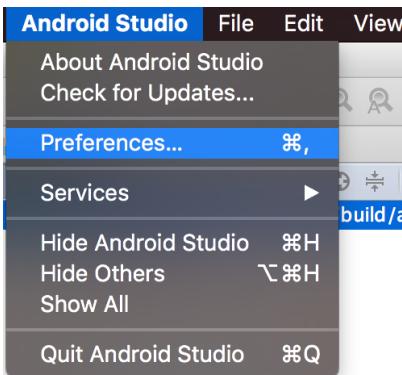


Figure 464. Android Studio Preferences

- Select Build, Execution, Deployment → Build Tools → Gradle
- Select the Use Local gradle distribution
- Press the ... and pick your local gradle 2.11 install

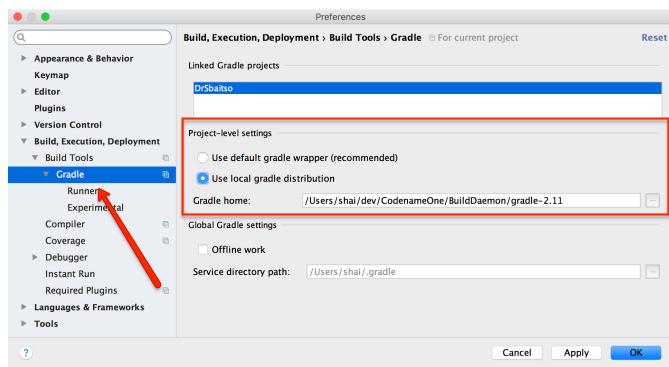


Figure 465. Local gradle config

16.2.5. FAQ

Should I use the Offline Builder?

Probably not.

Cloud build is far more convenient, simple. Doesn't require any installs (other than the plugin) and is much faster.

We built this tool for developers who work in situations that prohibit cloud build. E.g. government, banking etc. where regulation is restrictive.

Can I Move/Backup my Builders?

No.

We protect all the builders to avoid abuse. If you backup and restore on a new system the builders might stop working even if you are a paying enterprise customer.

Can I install the builders for all our developers?

Our licensing terms require a parallel developer seat for the Codename One developers in your company. If you have 5 Codename One developers they must all have an enterprise developer

account to comply.

E.g. You can't have one enterprise account and 4 basic accounts.

The reason behind this is simple, in the past we saw a lot of funneling from developers who built such a licensing structure.

What Happens if I Cancel?

If you cancel your enterprise subscription all your existing installed offline builders should work as before but you won't be able to update them or get support for this.

When are Versions Released?

We will try to keep this in the same release pace as library updates i.e. once a week typically on a Friday.

Are Version Numbers Sequential?

They grow but we sometimes skip versions. Versions map to our cloud deployment versioning scheme and we might skip versions in some cases.

Why is this Feature Limited to Enterprise Subscribers?

This is a complex tool to support & maintain. SaaS has a well defined business model where we can reduce prices and maintenance costs.

Offline builds are more like a shrinkwrap business model in which case our pricing needs to align itself to shrinkwrap pricing models for long term sustainability.

The main use case this product tries to address is government and highly regulated industries who are in effect enterprise users.

How Different is the Code From Cloud Builds?

We use the same code as we do in the cloud build process with minor modifications in the process. Since the cloud servers are setup by us they work differently but should align reasonably well.

16.3. Android Permissions

One of the annoying tasks when programming native Android applications is tuning all the required permissions to match your codes requirements, Codename One aims to simplify this. The build server automatically introspects the classes sent to it as part of the build and injects the right set of permissions required by the app.

However, sometimes developers might find the permissions that come up a bit confusing and might not understand why a specific permission came up. This maps Android permissions to the methods/classes in Codename One that would trigger them. Notice that this list isn't exhaustive as the API is rather large:

`android.permission.WRITE_EXTERNAL_STORAGE` - this permission appears by default for Codename One applications, since the `FileSystemStorage` API (which is used extensively) might have some dependencies on it. You can explicitly disable it using the build hint `android.blockExternalStoragePermission=true`, notice that this is something we don't test and it might fail on devices.

`android.permission.INTERNET` - this is a hardcoded permission in Codename One, the ability to connect to the network is coded into all Codename One applications.

`android.hardware.camera` & `android.permission.RECORD_AUDIO` - are triggered by `com.codename1.Capture`

`android.permission.RECORD_AUDIO` - is triggered by usage of `MediaManager.createMediaRecorder()` & `Display.createMediaRecorder()`

`android.permission.READ_PHONE_STATE` - is triggered by `com.codename1.ads` package, `com.codename1.components.Ads`, `com.codename1.components.ShareButton`, `com.codename1.media`, `com.codename1.push`, `Display.getUdid()` & `Display.getMsisdn()`. This permission is required for media in order to suspend audio playback when you get a phone call.

`android.hardware.location`, `android.hardware.location.gps`,
`android.permission.ACCESS_FINE_LOCATION`, `android.permission.ACCESS_MOCK_LOCATION` &
`android.permission.ACCESS_COARSE_LOCATION` - map to `com.codename1.maps` & `com.codename1.location`.

`package.permission.C2D_MESSAGE`, `com.google.android.c2dm.permission.RECEIVE`,
`android.permission.RECEIVE_BOOT_COMPLETED` - are requested by the `com.codename1.push` package

`android.permission.READ_CONTACTS` - triggers by the package `com.codename1.contacts` & `Display.getAllContacts()`.

`android.permission.VIBRATE` - is triggered by `Display.vibrate()` and `Display.notifyStatusBar()`

`android.permission.SEND_SMS` - is triggered by `Display.sendSMS()`

`android.permission.WAKE_LOCK` - is triggered by `Display.lockScreen()` & `Display.setScreenSaverEnabled()`

`android.permission.WRITE_CONTACTS` - is triggered by `Display.createContact()`, `Display.deleteContact()`, `ContactsManager.createContact()` & `ContactsManager.deleteContact()`

16.3.1. Permissions Under Marshmallow (Android 6+)

Starting with Marshmallow (Android 6+ API level 23) Android shifted to a permissions system that prompts users for permission the first time an API is used e.g. when accessing contacts the user will receive a prompt whether to allow contacts access.



Permission can be denied and a user can later on revoke/grant a permission via external settings UI

This is really great as it allows apps to be installed with a single click and no permission prompt during install which can increase conversion rates!

Enabling Permissions

Codenmae One compiles Android targets with SDK level 23 but not with target level 23!

This means that by default the new permission mode is still off and you won't see any of the effects mentioned below.



This will probably change to the default in the future but at the moment the target SDK defaults to 21

To activate this functionality you will need to set the target SDK to level 23 by using the `android.targetSdkVersion=23` build hint.

Permission Prompts

To test this API see the following simple contacts app:

```
Form f = new Form("Contacts", BoxLayout.y());
f.add(new InfiniteProgress());
Display.getInstance().invokeAndBlock(() -> {
    Contact[] ct = Display.getInstance().getAllContacts(true, true, false, true, true, false);
    Display.getInstance().callSerially(() -> {
        f.removeAll();
        for(Contact c : ct) {
            MultiButton mb = new MultiButton(c.getDisplayName());
            mb.setTextLine2(c.getPrimaryPhoneNumber());
            f.add(mb);
        }
        f.revalidate();
    });
});
f.show();
```

When we try to install this app without changing anything on an Android 6 device we see this UI:

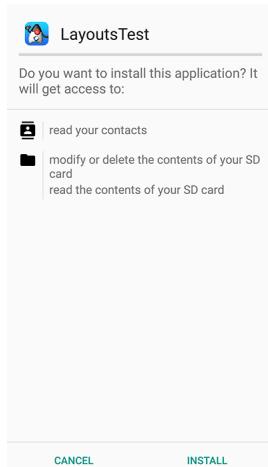


Figure 466. Install UI when using the old permissions system

When we set `android.targetSdkVersion=23` in the build hints and try to install again the UI looks like

this:

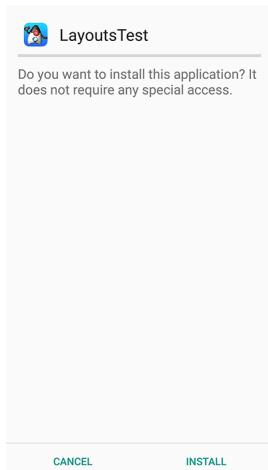


Figure 467. Install UI when using the new permissions system

When we launch the UI under the old permissions system we see the contacts instantly. In the new system we are presented with this UI:

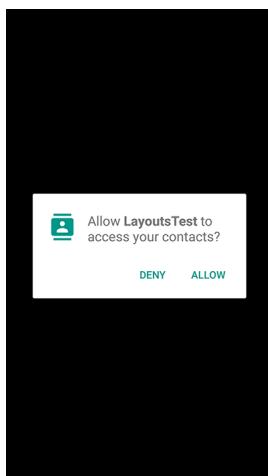


Figure 468. Native permission prompt first time

If we accept and allow all is good and the app loads as usual but if we deny then Codename One gives the user another chance to request the permission. Notice that in this case you can customize the prompt string as explained below.

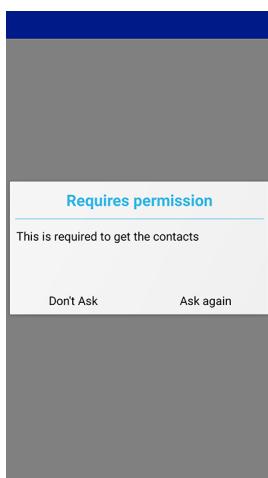


Figure 469. Codename One permission prompt

If we select don't ask then you will get a blank screen since the contacts will return as a 0 length array. This makes sense as the user is aware he denied permission and the app will still function as expected on a device where no contacts are available. However, if the user realizes his mistake he can double back and ask to re-prompt for permission in which case he will see this native prompt:

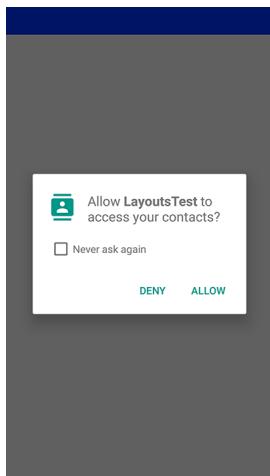


Figure 470. Native permission prompt second time

Notice that denying this second request will not trigger another Codename One prompt.

Code Changes

There are no explicit code changes needed for this functionality to "just work". The respective API's will work just like they always worked and will prompt the user seamlessly for permissions.



Some behaviors that never occurred on Android but were perfectly legal in the past might start occurring with the switch to the new API. E.g. the location manager might be null and your app must always be ready to deal with such a situation

When permission is requested a user will be seamlessly prompted/warned, Codename One has builtin text to control such prompts but you might want to customize the text. You can customize permission text via the **Display** properties e.g. to customize the text of the contacts permission we can do something such as:

```
Display.getInstance().setProperty("android.permission.READ_CONTACTS", "MyCoolChatApp needs access to your contacts so we can show you which of your friends already have MyCoolChatApp installed");
```

This is optional as there is a default value defined. You can define this once in the **init(Object)** method but for some extreme cases permission might be needed for different things e.g. you might ask for this permission with one reason at one point in the app and with a different reason at another point in the app.

The following permission keys are supported: `"android.permission.READ_PHONE_STATE"`, `android.permission.WRITE_EXTERNAL_STORAGE`, `android.permission.ACCESS_FINE_LOCATION`, `android.permission.SEND_SMS`, `android.permission.READ_CONTACTS`, `android.permission.WRITE_CONTACTS`, `android.permission.RECORD_AUDIO`.

Simulating Prompts

You can simulate permission prompts by checking that option in the simulator menu.

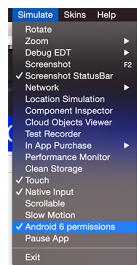


Figure 471. Simulate permission prompts menu item in the simulator

This will produce a dialog to the user whenever this happens in Android and will try to act in a similar way to the device. Notice that you can test it in the iOS simulator too.

AndroidNativeUtil's checkForPermission

If you write Android native code using native interfaces you are probably familiar with the `AndroidNativeUtil` class from the `com.codename1.impl.android` package.

This class provides access to many low level capabilities you would need as a developer writing native code. Since native code might need to request a permission we introduced the same underlying logic we used namely: `checkForPermission`.

To get a permission you can use this code as such:

```
if(!AndroidNativeUtil.checkForPermission(  
    Manifest.permission.READ_PHONE_STATE, "  
    This should be the description shown to the user...")){  
    // you didn't get the permission, you might want to return here  
}  
// you have the permission, do what you need
```

This will prompt the user with the native UI and later on with the fallback option as described above. Notice that the `checkForPermission` method is a blocking method and it will return when there is a final conclusion on the subject. It uses `invokeAndBlock` and can be safely invoked on the event dispatch thread without concern.

16.4. On Device Debugging

Codename One supports debugging applications on devices by using the natively generated project. All paid subscription levels include the ability to check an `Include Source` flag in the settings that returns a native OS project. You can debug that project in the respective native IDE.

In iOS this is usually strait forward, just open the project with xcode and run it optionally disabling bitcode. Unzip the `.bz2` file and open the `.xcworkspace` file if it's available otherwise open the `.xcodeproj` file inside the `dist` directory.



Only the `.xcworkspace` if it is there, it is activated by the CocoaPods build pipeline so it won't always be there

With Android Studio this is sometimes as very easy task as it is possible to actually open the gradle project in Android Studio and just run it. However, due to the fragile nature of the gradle project this stopped working for some builds and has been "flaky".

16.4.1. Android Studio Debugging (Easy Way)

By default you should be able to open the gradle project in Android Studio and just run it. To get this to work open the Android Studio **Setting** and select gradle **2.11**.

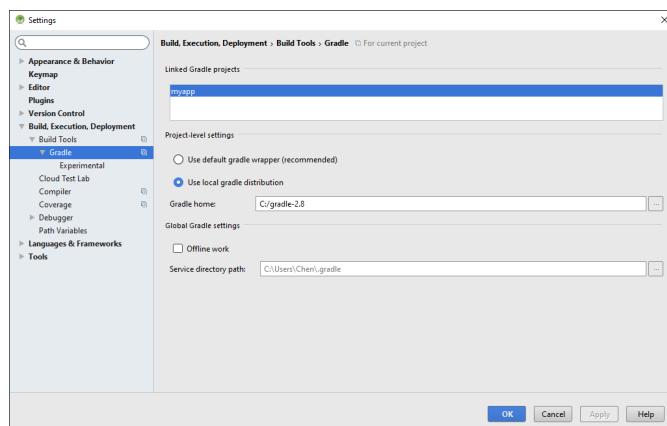


Figure 472. Gradle settings UI in Android Studio (notice you need gradle 2.11 and not 2.8 as pictured here)

If this works for you then you can ignore the section below.

16.4.2. Android Studio Debugging the Hard Way

In some cases the gradle project might not work or this might fail with a change from Google.

Here are steps that should work for everyone:

1. Check the include source flag in the IDE and send a build
2. Download the `sources.zip` result from the build server
3. Launch Android Studio and create a new project
4. Make sure to use the same package and app name as you did in the Codename One project, select to not create an activity
5. Unzip the `sources.zip` file and copy the `main` directory from its `src` directory to the Android Studio projects `src` directory make sure to overwrite files/directories.
6. Copy its `libs` directory on top of the existing `libs`
7. Copy the source gradle dependencies content to the destination gradle file
8. Connect your device and press the Debug button for the IDE



You might need to copy additional gradle file meta-data such as multi-dexing etc.

You might not need to repeat the whole thing with every build. E.g. it might be practical to only

copy the `userSources.jar` from the `libs` directory to get the latest version of your code. You can copy the `src/main` directory to get the latest up to date Android port.

16.5. Native Interfaces

Sometimes you may wish to use an API that is unsupported by Codename One or integrate with a 3rd party library/framework that isn't supported. These are achievable tasks when writing native code and Codename One lets you encapsulate such native code using native interfaces.

16.5.1. Introduction

Notice that when we say "native" we do not mean C/C++ always but rather the platforms "native" environment. So in the case of Android the Java code will be invoked with full access to the Android API, in case of iOS an Objective-C message would be sent and so forth.



You can still access C code under Android either by using JNI from the Android native code or by using a library

Native interfaces are designed to only allow primitive types, Strings, arrays of primitive types (single dimension only) & `PeerComponent` [<https://www.codenameone.com/javadoc/com/codename1/ui/PeerComponent.html>] values. Any other type of parameter/return type is prohibited. However, once in the native layer the native code can act freely and query the Java layer for additional information.



The reason for the limits is the disparity between the platforms. Mapping a Java `Object` to an Objective-C `NSObject` is possible but leads to odd edge cases and complexity e.g. GC vs. ARC in a disparate object graph

Furthermore, native methods should avoid features such as overloading, varargs (or any Java 5+ feature for that matter) to allow portability for languages that do not support such features.



Do not rely on pass by reference/value behavior since they vary between platforms

Implementing a native layer effectively means:

1. Creating an interface that extends `NativeInterface` [<https://www.codenameone.com/javadoc/com/codename1/system/NativeInterface.html>] and only defines methods with the arguments/return values declared in the previous paragraph.
2. Creating the proper native implementation hierarchy based on the call conventions for every platform within the native directory

E.g. to create a simple hello world interface do something like:

```

package com.mycompany.myapp;
import com.codename1.system.NativeInterface;
public interface MyNative extends NativeInterface {
    String helloWorld(String hi);
}

```

We now need to right click the class in the IDE and select the **Generate Native Access** menu item:

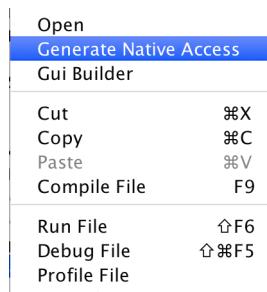


Figure 473. Generating the native code

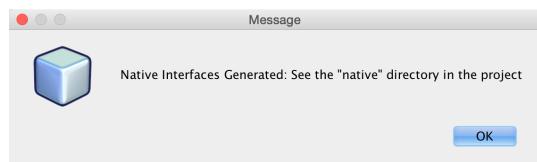


Figure 474. Once generated we are prompted that the native code is in the "native" directory

We can now look int the **native** directory in the project root (in NetBeans you can see that in the **Files** tab) and you can see something that looks like this:

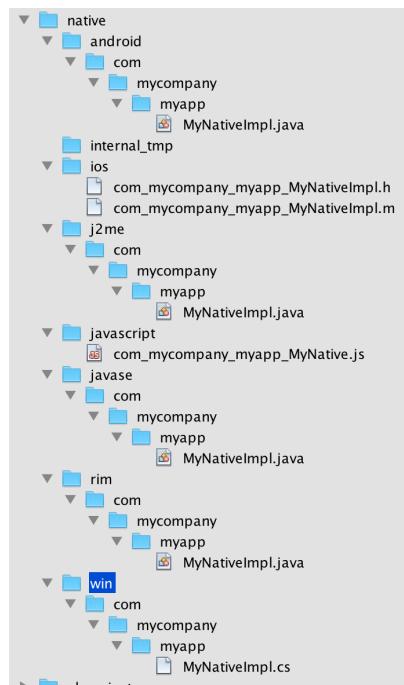


Figure 475. Native directory structure containing stubs for the various platforms

These are effectively stubs you can edit to implement the methods in native code.



If you re-run the **Generate Native Access** tool you will get this dialog, if you answer yes all the files will be overwritten, if you answer no only files you deleted/renamed will be recreated

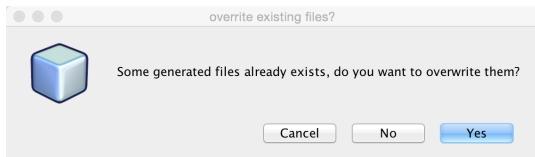


Figure 476. Running "Generate Native Access" when some/all of the native files exist already

For now lets leave the stubs and come back to them soon. From the Codename One Java code we can call the implementation of this native interface using:

```
MyNative my = NativeLookup.create(MyNative.class);
if(my != null && my.isSupported()) {
    Log.p(my.helloWorld("Hi"));
}
```

Notice that for this to work you must implement the native code on all supported platforms.

We'll start with Android which should be familiar and intuitive to many developers, this is how the generated file under the `native/android` directory looks:

```
package com.mycompany.myapp;

public class MyNativeImpl {
    public String helloWorld(String param) {
        return null;
    }

    public boolean isSupported() {
        return false;
    }
}
```

The stub implementation always returns `false`, `null` or `0` by default. The `isSupported` also defaults to `false` thus allowing us to implement a `NativeInterface` on some platforms and leave the rest out without really knowing anything about these platforms.

We can implement the Android version using code similar to this:

```

package com.mycompany.myapp;

import android.util.Log; ①

public class MyNativeImpl { ②
    ③
    public String helloWorld(String param) {
        Log.d("MyApp", param);
        return "Tada";
    }

    public boolean isSupported() { ④
        return true;
    }
}

```

① Notice that we are using the Android native `android.util.Log` class which isn't accessible from standard Codename One code

② The impl class doesn't physically implement the `MyNative` interface!

This is intentional and due to the `PeerComponent` functionality mentioned below. You don't need to add an `implements` clause.

③ Notice that there is no constructor and the class is public. It is crucial that the system will be able to allocate the class without obstruction. You can use a constructor but it can't have any arguments and you shouldn't rely on semantics of construction.

④ We implemented the native method and that we set `isSupported` to true.

The IDE won't provide completion suggestions and will claim that there are errors in the code!

 Codename One doesn't include the native platforms in its bundle e.g. the full Android SDK or the full xcode Objective-C runtime. However, since the native code is compiled on the servers (where these runtimes are present) this shouldn't be a problem

 When implementing a non-trivial native interface, send a server build with the "Include Source" option checked. Implement the native interface in the native IDE then copy and paste the native code back into Codename One

The implementation of this interface is nearly identical for Android, J2ME & Java SE.

Use the Android Main Thread (Native EDT)

iOS, Android & pretty much any modern OS has an EDT like thread that handles events etc. The problem is that they differ in their nuanced behavior. E.g. Android will usually respect calls off of the EDT and iOS will often crash. Some OS's enforce EDT access rigidly and will throw an exception when you violate that...

Normally you don't need to know about these things, hidden functionality within our

implementation bridges between our EDT and the native EDT to provide consistent cross platform behavior. But when you write native code you need awareness.

Why not Implicitly call Native Interfaces on the Native EDT?

Calling into the native EDT includes overhead and it might not be necessary for some features (e.g. IO, polling etc.). Furthermore, some calls might work well with asynchronous calls while others might need synchronous results and we can't know in advance which ones you would need.

How do we Access the Native EDT?

Within your native code in Android do something like:

```
com.codename1.impl.android.AndroidNativeUtil.getActivity().runOnUiThread(new Runnable() {  
    public void run() {  
        // your native code here...  
    }  
});
```

This will execute the block within `run()` asynchronously on the native Android UI thread. If you need synchronous execution we have a special method for Codename One:

```
com.codename1.impl.android.AndroidImplementation.runOnUiThreadAndBlock(new Runnable() {  
    public void run() {  
        // your native code here...  
    }  
});
```

This blocks in a way that's OK with the Codename One EDT which is unique to our Android port.

Gradle Dependencies

Integrating a native OS library isn't hard but it sometimes requires some juggling. Most instructions target developers working with xcode or Android Studio & you need to twist your head around them. In Android the steps for integration in most modern libraries include a gradle dependency.

E.g. we published a library that added support for [Intercom](#) [<https://www.codenameone.com/blog/intercom-support.html>]. The native Android integration instructions for the library looked like this:

Add the following dependency to your app's `build.gradle` file:

```
dependencies {  
    compile 'io.intercom.android:intercom-sdk:3.+'  
}
```

Which instantly raises the question: "How in the world do I do that in Codename One"?

Well, it's actually pretty simple. You can add the build hint:

```
android.gradleDep=compile 'io.intercom.android:intercom-sdk:3.+'
```

This would "work" but there is a catch...

You might need to define the specific version of the Android SDK used and specific version of Google play services version used. Intercom is pretty sensitive about those and demanded that we also add:

```
android.playServices=9.8.0  
android.sdkVersion=25
```

Once those were defined the native code for the Android implementation became trivial to write and the library was easy as there were no jars to include.

16.5.2. Objective-C (iOS)

When generating the Objective-C code the "Generate Native Sources" tool produces two files: `com_mycompany_myapp_MyNativeImpl.h` & `com_mycompany_myapp_MyNativeImpl.m`.

The `.m` files are the Objective-C equivalent of `.c` files and `.h` files contain the header/include information. In this case the `com_mycompany_myapp_MyNativeImpl.h` contains:

```
#import <Foundation/Foundation.h>  
  
@interface com_mycompany_myapp_MyNativeImpl : NSObject {  
}  
  
-(NSString*)helloWorld:(NSString*)param;  
-(BOOL)isSupported;  
@end
```

And `com_mycompany_myapp_MyNativeImpl.m` contains:

```
#import "com_mycompany_myapp_MyNativeImpl.h"

@implementation com_mycompany_myapp_MyNativeImpl

-(NSString*)helloWorld:(NSString*)param{
    return nil;
}

-(BOOL)isSupported{
    return NO;
}

@end
```



Objective-C relies on argument names as part of the message (method) signature.

So `- (NSString*)helloWorld:(NSString*)param` isn't the same as `- (NSString*)helloWorld:(NSString*)iChangedThisName!`

Don't change argument names in the Objective-C native interface!

Here is a simple implementation similar to above:

```
#import "com_mycompany_myapp_MyNativeImpl.h"

@implementation com_mycompany_myapp_MyNativeImpl

-(NSString*)helloWorld:(NSString*)param{
    NSLog(@"MyApp: %@", param);
    return @"Tada";
}

-(BOOL)isSupported{
    return YES;
}

@end
```

Using the iOS Main Thread (Native EDT)

iOS has a native thread you should use for all calls just like Android. Check out the Native EDT on Android section above for reference.

On iOS this is pretty similar to Android (if you consider objective-c to be similar). This is used for asynchronous invocation:

```
dispatch_async(dispatch_get_main_queue(), ^{
    // your native code here...
});
```

You can use this for synchronous invocation, notice the lack of the `a` in the dispatch call:

```
dispatch_sync(dispatch_get_main_queue(), ^{
    // your native code here...
});
```

The problem with the synchronous call is that it will block the caller thread, if the caller thread is the EDT this can cause performance issues and even a deadlock. It's important to be very cautious with this call!

Use Cocoapods For Dependencies

Cocoapods are the iOS equivalent of gradle dependencies.

CocoaPods allow us to add a native library dependency to iOS far more easily than Gradle. By default we target iOS 7.0 or newer which is supported by Intercom only for older versions of the library. Annoyingly CocoaPods might seem to work but some specific API's won't work since it fell back to an older version... To solve this you have to explicitly define the build hint `ios.pods.platform=8.0` to force iOS 8 or newer. You might need to force it to even newer versions as some libraries force an iOS 9 minimum etc.

Including intercom itself required a single build hint: `ios.pods=Intercom` which you can obviously extend by using commas to include multiple libraries. You can search the [cocoapods website](https://cocoapods.org/) [<https://cocoapods.org/>] for supported 3rd party libraries which includes everything you would expect. One important advantage when working with CocoaPods is the faster build time as the upload to the Codename One website is smaller and the bandwidth we have to CocoaPods is faster. Another advantage is the ability to keep up with the latest developments from the library providers.

16.5.3. Javascript

Native interfaces in Javascript look a little different than the other platforms since Javascript doesn't natively support threads or classes. The native implementation should be placed in a file with name matching the name of the package and the class name combined where the `.` elements are replaced by underscores.

The default generated stubs for the JavaScript build look like this `com_mycompany_myapp_MyNative`:

```
(function(exports){

var o = {};

o.helloWorld__java_lang_String = function(param1, callback) {
    callback.error(new Error("Not implemented yet"));
};

o.isSupported_ = function(callback) {
    callback.complete(false);
};

exports.com_mycompany_myapp_MyNative= o;

})(cn1_get_native_interfaces());
```

A simple implementation looks like this.

```
(function(exports){

var o = {};

o.helloWorld__java_lang_String = function(param1, callback) {
    callback.complete("Hello World!!!");
}

o.isSupported_ = function(callback) {
    callback.complete(true);
};

exports.com_my_code_MyNative = o;

})(cn1_get_native_interfaces());
```

Notice that we use the `complete()` method of the provided callback to pass the return value rather than using the `return` statement. This is to work around the fact that Javascript doesn't natively support threads. The **Java** thread that is calling your native interface will block until your method calls `callback.complete()`. This allows you to use asynchronous APIs inside your native method while still allowing Codename One to work use your native interface via a synchronous API.



Make sure you call either `callback.complete()` or `callback.error()` in your method at some point, or you will cause a deadlock in your app (code calling your native method will just sit and "wait" forever for your method to return a value).

The naming conventions for the methods themselves are modeled after the naming conventions shown in the previous examples:

<method-name>__<param-1-type>_<param-2-type>_⋯<param-n-type>

Where <method-name> is the name of the method in Java, and the `<param-X-type>`'s are a string representing the parameter type. The general rule for these strings are:

1. Primitive types are mapped to their type name. (E.g. `int` to "int", `double` to "double", etc...).
2. Reference types are mapped to their fully-qualified class name with '.' replaced with underscores. E.g. `java.lang.String` would be "java_lang_String".
3. Array parameters are marked by their scalar type name followed by an underscore and "1ARRAY". E.g. `int[]` would be "int_1ARRAY" and `String[]` would be "java_lang_String_1ARRAY".

JavaScript Examples

Java API:

```
public void print(String str);
```

becomes

```
o.print_java_lang_String = function(param1, callback) {  
    console.log(param1);  
    callback.complete();  
}
```

Java API:

```
public int add(int a, int b);
```

becomes

```
o.add_int_int = function(param1, param2, callback) {  
    callback.complete(param1 + param2);  
}
```

```
public int add(int[] a);
```

becomes

```

o.add__int_1ARRAY = function(param1, callback) {
    var c = 0, len = param1.length;
    for (var i = 0; i < len; i++) {
        c += param1[i];
    }
    callback.complete(c);
}

```

16.5.4. Native GUI Components

PeerComponent [https://www.codenameone.com/javadoc/com/codename1/ui/PeerComponent.html] return values are automatically translated to the platform native peer as an expected return value. E.g. for a **NativeInterface** method such as this:

```
public PeerComponent` createPeer();
```

Android native implementation would use:

```
public View createPeer() {
    return null;
}
```

The iPhone would need to return a pointer to a view e.g.:

- (UIView*)createPeer;



Not all platforms support native peers. Specifically JavaSE doesn't support them due to the way the JavaSE native interfaces are mapped to their implementation. Note that this won't limit the code from running on an unsupported platform. Only that specific method won't work.

Javascript would expect a DOM Element (e.g. a `<div>` tag to be returned.). E.g.

```

o.createHelloComponent_ = function(callback) {
    var c = jQuery('<div>Hello World</div>')
        .css({'background-color' : 'yellow', 'border' : '1px solid blue'});
    callback.complete(c.get(0));
};

```

Notice that if you want to use a native library (jar, .a file etc.) just places it within the appropriate native directory and it will be packaged into the final executable. You would only be able to reference it from the native code and not from the Codename One code, which means you will need to build native interfaces to access it.

This is discussed further below.

16.5.5. Type Mapping & Rules

Several rules govern the creation of NativeInterfaces and we only briefly covered some of them.

- The implementation class must have a default public constructor or no constructor at all
- Native methods can't throw exceptions, checked or otherwise
- A native method can't have the name `init` as this is a reserved method in Objective-C
- Only the supported types listed below can be used
- Native implementations can't rely on pass by reference/value semantics as those might change between platforms
- `hashCode`, `equals` & `toString` are reserved and won't be mapped to native code

Table 11. NativeInterface Supported Types

Java	Android	JavaSE	Obj-C	C#	byte
byte	byte	char	sbyte	boolean	boolean
boolean	BOOL	bool	char	char	char
int	char	short	short	short	short
short	int	int	int	int	int
long	long	long	long long	long	float
float	float	float	float	double	double
double	double	double	String	String	String
NSString*	String	byte[]	byte[]	byte[]	NSSData*
sbyte[]	boolean[]	boolean[]	boolean[]	NSData*	bool[]
char[]	char[]	char[]	NSData	char[]	short[]
short[]	short[]	NSData*	short[]	int[]	int[]
int[]	NSData*	int[]	long[]	long[]	long[]
NSData*	long[]	float[]	float[]	float[]	NSData*
float[]	double[]	double[]	double[]	NSData*	double[]



JavaScript is excluded from the table above as it isn't a type safe language and thus has no such type mapping



`PeerComponent` on iOS is `void*` but `UIView` is expected as a result

The examples below demonstrate the signatures for this method on all platforms:

Listing 22. NativeInterface definition

```
public void test(byte b, boolean boo, char c, short s,
    int i, long l, float f, double d, String ss,
    byte[] ba, boolean[] booa, char[] ca, short[] sa, int[] ia,
    long[] la, float[] fa, double[] da,
    PeerComponent cmp);
```

Listing 23. Android Version

```
public void test(byte param, boolean param1, char param2,
    short param3, int param4, long param5, float param6,
    double param7, String param8, byte[] param9,
    boolean[] param10, char[] param11, short[] param12,
    int[] param13, long[] param14, float[] param15,
    double[] param16, android.view.View param17) {
}
```

Listing 24. iOS Version

```
-(void)test:(char)param param1:(BOOL)param1
    param2:(int)param2 param3:(short)param3 param4:(int)param4
    param5:(long long)param5 param6:(float)param6
    param7:(double)param7 param8:(NSString*)param8
    param9:(NSData*)param9 param10:(NSData*)param10
    param11:(NSData*)param11 param12:(NSData*)param12
    param13:(NSData*)param13 param14:(NSData*)param14
    param15:(NSData*)param15 param16:(NSData*)param16
    param17:(void*)param17;
}
```



We had to break lines for the print version, the JavaScript version is a really long method name that literally broke the book!

Listing 25. JavaScript Version

```
o.test__byte_boolean_char_short_int_long_float_double
_java_lang_String_byte_1ARRAY_boolean_1ARRAY_char_1ARRAY
_short_1ARRAY_int_1ARRAY_long_1ARRAY_float_1ARRAY_double
_1ARRAY_com_codename1_ui_PeerComponent = function(param1, param2, param3, param4, param5, param6, param7, param8, param9,
param10, param11, param12, param13, param14, param15, param16, param17, param18, callback) {
    callback.error(new Error("Not implemented yet"));
};
```

Listing 26. Java SE Version

```
public void test(byte param, boolean param1, char param2, short param3, int param4, long param5, float param6, double
param7, String param8, byte[] param9, boolean[] param10, char[] param11, short[] param12, int[] param13, long[] param14,
float[] param15, double[] param16, com.codename1.ui.PeerComponent param17) {
```

Listing 27. C# Version

```
public void test(byte param, bool param1, char param2, short param3, int param4, long param5, float param6, double param7, String param8, byte[] param9, boolean[] param10, char[] param11, short[] param12, int[] param13, long[] param14, float[] param15, double[] param16, FrameworkElement param17) {  
}
```

16.5.6. Android Native Permissions

Normally permissions in Codename One are seamless. Codename One traverses the bytecode and automatically assigns permissions to Android applications based on the API's used by the developer.

However, when accessing native functionality this just won't work since native code might require specialized permissions and we don't/can't run any serious analysis on it (it can be just about anything).

So if you require additional permissions in your Android native code you need to define them in the build arguments using `android.permission.<PERMISSION_NAME>=true` for each permission you want to include. A full list of permissions are listed in Android's [Manifest.permission documentation](#) [<https://developer.android.com/reference/android/Manifest.permission.html>].

E.g.

```
android.permission.ADD_VOICEMAIL=true  
android.permission.BATTERY_STATS=true  
...
```

You can specify the maximum SDK version in which the permission is needed using the `android.permission.<PERMISSION_NAME>.maxSdkVersion` build hint. You can also specify whether the permission is **required** for the app to run using the `android.permission.<PERMISSION_NAME>.required` build hint.

E.g.

```
android.permission.ADD_VOICEMAIL=true  
android.permission.BATTERY_STATS=true  
android.permission.ADD_VOICEMAIL.required=false  
android.permission.ADD_VOICEMAIL.maxSdkVersion=18  
...
```

You can alternatively use the `android.xpermissions` build hint to inject `<uses-permission>` tags into the manifest file. E.g.:

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



You need to include the full XML snippet. You can unify multiple lines into a single line in the GUI as XML allows that.

16.5.7. Native AndroidNativeUtil

If you do any native interfaces programming in Android you should be familiar with the `AndroidNativeUtil` class which allows you to access native device functionality more easily from the native code. E.g. many Android API's need access to the `Activity` which you can get by calling `AndroidNativeUtil.getActivity()`.

The native util class includes quite a few other features such as:

- `runOnUiThreadAndBlock(Runnable)` - this is such a common pattern that it was generalized into a public static method. Its identical to `Activity.runOnUiThread` but blocks until the runnable finishes execution.
- `addLifecycleListener/removeLifecycleListener` - These essentially provide you with a callback to lifecycle events: `onCreate` etc. which can be pretty useful for some cases.
- `registerViewRenderer` - `PeerComponent` [<https://www.codenameone.com/javadoc/com/codename1/ui/PeerComponent.html>]'s are usually shown on top of the UI since they are rendered within their own thread outside of the EDT cycle. So when we need to show a `Dialog` [<https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html>] on top of the peer we grab a screenshot of the peer, hide it and then show the dialog with the image as the background (the same applies for transitions). Unfortunately some components (specifically the `MapView`) might not render properly and require custom code to implement the transferal to a native Bitmap, this API allows you to do just that.

You can work with `AndroidNativeUtil` using native code such as this:

```
import com.codename1.impl.android.AndroidNativeUtil;

class NativeCallsImpl {
    public void nativeMethod() {
        AndroidNativeUtil.getActivity().runOnUiThread(new Runnable() {
            public void run() {
                ...
            }
        });
    }
    ....
}
```

16.5.8. Broadcast Receiver

A common way to implement features in Android is the `BroadcastReceiver` API. This allows intercepting operating system events for common use cases.

A good example is intercepting incoming SMS which is specific to Android so we'd need a

broadcast receiver to implement that. This is often confusing to developers who sometimes derive the impl class from broadcast receiver. That's a mistake...

The solution is to place any native Android class into the `native/android` directory. It will get compiled with the rest of the native code and "just works". So you can place this class under `native/android/com/codename1/sms/intercept`:

```
package com.codename1.sms.intercept;

import android.content.*;
import android.os.Bundle;
import android.telephony.*;
import com.codename1.io.Log;

public class SMSListener extends BroadcastReceiver {

    @Override
    public void onReceive(Context ctxt, Intent intent) {
        // based on code from https://stackoverflow.com/questions/39526138/broadcast-receiver-for-receive-sms-is-not-working-when-declared-in-manifeststat
        if(intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
            Bundle bundle = intent.getExtras();
            SmsMessage[] msgs = null;
            if (bundle != null){
                try{
                    Object[] pdus = (Object[]) bundle.get("pdus");
                    msgs = new SmsMessage[pdus.length];
                    for(int i=0; i<msgs.length; i++){
                        msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                        String msgBody = msgs[i].getMessageBody();
                        SMSCallback.smsReceived(msgBody);
                    }
                } catch(Exception e) {
                    Log.e(e);
                    SMSCallback.smsReceiveError(e);
                }
            }
        }
    }
}
```

The code above is pretty standard native Android code, it's just a callback in which most of the logic is similar to the native Android code mentioned in this [stackoverflow question](#) [<https://stackoverflow.com/questions/39526138/broadcast-receiver-for-receive-sms-is-not-working-when-declared-in-manifeststat>].

But there is still more you need to do. In order to implement this natively we need to register the permission and the receiver in the `manifest.xml` file as explained in that question. This is how their native manifest looked:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bulsy.smstalk1">
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_CONTACTS" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name="com.bulsy.smstalk1.SmsListener">
            android:enabled="true"
            android:permission="android.permission.BROADCAST_SMS"
            android:exported="true">
            <intent-filter android:priority="2147483647">//this doesnt work
                <category android:name="android.intent.category.DEFAULT" />
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>

```

We only need the broadcast permission XML and the permission XML. Both are doable via the build hints. The former is pretty easy:

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

The latter isn't much harder, notice I took multiple lines and made them into a single line for convenience:

```

android.xapplication=<receiver android:name="com.codename1.sms.intercept.SMSListener" android:enabled="true"
    android:permission="android.permission.BROADCAST_SMS" android:exported="true"> <intent-filter
    android:priority="2147483647"><category android:name="android.intent.category.DEFAULT" /> <action
    android:name="android.provider.Telephony.SMS_RECEIVED" /> </intent-filter> </receiver>

```

Here it is formatted nicely:

```
<receiver android:name="com.codename1.sms.intercept.SMSListener"
    android:enabled="true"
    android:permission="android.permission.BROADCAST_SMS"
    android:exported="true">
    <intent-filter android:priority="2147483647">
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

Listening & Permissions

You will notice that these don't include the actual binding or permission prompts you would expect for something like this. To do this we need a native interface.

The native sample in stack overflow bound the listener in the activity but here we want the app code to decide when we should bind the listening:

```
public interface NativeSMSInterceptor extends NativeInterface {
    public void bindSMSListener();
    public void unbindSMSListener();
}
```

That's easy!

Notice that `isSupported()` returns false for all other OS's so we won't need to ask whether this is "Android" we can just use `isSupported()`.

The implementation is pretty easy too:

```

package com.codename1.sms.intercept;

import android.Manifest;
import android.content.IntentFilter;
import com.codename1.impl.android.AndroidNativeUtil;

public class NativeSMSInterceptorImpl {
    private SMSListener smsListener;
    public void bindSMSListener() {
        if(AndroidNativeUtil.checkForPermission(Manifest.permission.RECEIVE_SMS, "We can automatically enter the SMS code for you")) { ①
            smsListener = new SMSListener();
            IntentFilter filter = new IntentFilter();
            filter.addAction("android.provider.Telephony.SMS_RECEIVED");
            AndroidNativeUtil.getActivity().registerReceiver(smsListener, filter); ②
        }
    }

    public void unbindSMSListener() {
        AndroidNativeUtil.getActivity().unregisterReceiver(smsListener);
    }

    public boolean isSupported() {
        return true;
    }
}

```

① This will trigger the permission prompt on Android 6 and newer. Even though the permission is declared in XML this isn't enough for 6+. Notice that even when you run on Android 6 you still need to declare permissions in XML!

② Here we actually bind the listener, this allows us to grab one SMS and not listen in on every SMS coming thru

16.5.9. Native Code Callbacks

Native interfaces standardize the invocation of native code from Codename One, but it doesn't standardize the reverse of callbacks into Codename One Java code. The reverse is naturally more complicated since its platform specific and more error prone.

A common "trick" for calling back is to just define a static method and then trigger it from native code. This works nicely for Android, Java SE, Blackberry & Java ME since those platforms use Java for their "native code". Mapping this to iOS requires some basic understanding of how the iOS VM works.

For the purpose of this explanation lets pretend we have a class called NativeCallback in the src hierarchy under the package **com.mycompany** that has the method: **public static void callback()**.

```

package com.mycompany;
public class NativeCallback {
    public static void callback() {
        // do stuff
    }
}

```

So if I want to call it from Android or all of the Java based platforms I can just write this in the "native" code:

```
com.mycompany.NativeCallback.callback();
```

I can also pass a argument as we do later on:

```
com.mycompany.NativeCallback.callback("My Arg");
```

Accessing Callbacks from Objective-C

If we want to invoke that method from Objective-C we need to do the following.

Add an include statement as such:

```
#include "com_mycompany_NativeCallback.h"  
#include "CodenameOne_GLViewController.h"
```

Notice that the `CodenameOne_GLViewController.h` include defines various macros such as `CN1_THREAD_STATE_PASS_SINGLE_ARG`.

Then when we want to trigger the method just do:

```
com_mycompany_NativeCallback_callback__(CN1_THREAD_STATE_PASS_SINGLE_ARG);
```



For most callbacks you should use the macro `CN1_THREAD_GET_STATE_PASS_SINGLE_ARG` instead of `CN1_THREAD_STATE_PASS_SINGLE_ARG` also make sure to add `#include "cn1_globals.h" in the file

The VM passes the thread context along method calls to save on API calls (thread context is heavily used in Java for synchronization, gc and more).

We can easily pass arguments like:

```
public static void callback(int arg)
```

Which maps to native as (notice the extra _ before the int):

```
com_mycompany_NativeCallback_callback___int(CN1_THREAD_GET_STATE_PASS_ARG intValue);
```

Notice that there is no comma between the `CN1_THREAD_GET_STATE_PASS_ARG` and the value!

Why No Comma?

The comma is included as part of the macro which makes for code that isn't as readable.

The reason for this dates to the migration from XMLVM^[10] to the current ParparVM implementation. CN1_THREAD_GET_STATE_PASS_ARG is defined as nothing in XMLVM since it didn't use that concept. Yet under ParparVM it will include the necessary comma.

A common use case is passing string values to the Java side, or really NSString* which is iOS equivalent. Assuming a method like this:

```
public static void callback(String arg)
```

You would need to convert the `NSString*` value you already have to a `java.lang.String` which the callback expects.

The `fromNSString` function also needs this special argument so you will need to modify the method as such:

```
com_mycompany_NativeCallback_callback__java_lang_String(CN1_THREAD_GET_STATE_PASS_ARG  
fromNSString(CN1_THREAD_GET_STATE_PASS_ARG nsStringValue));
```

And finally you might want to return a value from callback as such:

```
public static int callback(int arg)
```

This is tricky since the method name changes to support covariant return types and so the signature would be:

```
com_mycompany_NativeCallback_callback__int_R_int(intValue);
```

The upper case R allows us to differentiate between void `callback(int, int)` and `int callback(int)`.



Covariant return types are a little known Java 5 feature. E.g. the method `Object getX()` can be overridden by `MyObject getX()`. However, in the VM level they can both exist side by side.

Accessing Callbacks from Javascript

The mechanism for invoking static callback methods from Javascript (for the Javascript port only) is similar to Objective-C's. The `this` object in your native interface method contains a property named `$GLOBAL$` that provides access to static java methods. This object will contain Javascript mirror objects for each Java class (though the property name is mangled by replacing `"."` with underscores). Each mirror object contains a wrapper method for its underlying class's static

methods where the method name follows the same naming convention as is used for the Javascript native methods themselves (and very similar to the naming conventions used in Objective-C).

For example, the Google Maps project includes the static callback method:

```
static void fireMapChangeEvent(int mapId, final int zoom, final double lat, final double lon) { ... }
```

defined in the `com.codename1.googlemaps.MapContainer` class.

This method is called from Javascript inside a native interface using the following code:

```
var fireMapChangeEvent = this.$GLOBAL$.com_codename1_googlemaps_MapContainer.fireMapChangeEvent__int_int_double_double;
google.maps.event.addListener(this.map, 'bounds_changed', function() {
    fireMapChangeEvent(self.mapId, self.map.getZoom(), self.map.getCenter().lat(), self.map.getCenter().lng());
});
```

In this example we first obtain a reference to the `fireMapChangeEvent` method, and then call it later. However, we could have called it directly also.

Your code **MUST** contain the full string path `this.$GLOBAL$.your_class_name.your_method_name` or the build server will not be able to recognize that your code requires this method. The `$GLOBAL$` object is populated by the build server only with those classes and methods that are used inside your native methods. If the build server doesn't recognize that the methods are being used (via this pattern) it won't generate the necessary wrappers for your Javascript code to access the Java methods.



Callbacks of the SMS Receiver

The SMS Broadcast Receiver code from before also used callbacks such as this:

```

package com.codename1.sms.intercept; ①

import com.codename1.util.FailureCallback;
import com.codename1.util.SuccessCallback;
import static com.codename1.ui.CN.*;

/***
 * This is an internal class, it's package protect to hide that
 */
class SMSCallback {
    static SuccessCallback<String> onSuccess;
    static FailureCallback onFail;

    public static void smsReceived(String sms) {
        if(onSuccess != null) {
            SuccessCallback<String> s = onSuccess;
            onSuccess = null;
            onFail = null;
            SMSInterceptor.unbindListener();
            callSerially(() -> s.onSucess(sms)); ②
        }
    }

    public static void smsReceiveError(Exception err) {
        if(onFail != null) {
            FailureCallback f = onFail;
            onFail = null;
            SMSInterceptor.unbindListener();
            onSuccess = null;
            callSerially(() -> f.onError(null, err, 1, err.toString()));
        } else {
            if(onSuccess != null) {
                SMSInterceptor.unbindListener();
                onSuccess = null;
            }
        }
    }
}

```

- ① Notice that the package is the same as the native code and the other classes. This allows the callback class to be package protected so it isn't exposed via the API (the class doesn't have the public modifier)
- ② We wrap the callback in call serially to match the Codename One convention of using the EDT by default. The call will probably arrive on the Android native thread so it makes sense to normalize it and not expose the Android native thread to the user code

Asynchronous Callbacks & Threading

One of the problematic aspects of calling back into Java from Javascript is that Javascript has no

notion of multi-threading. Therefore, if the method you are calling uses Java's threads at all (e.g. It includes a `wait()`, `notify()`, `sleep()`, `callSerially()`, etc...) you need to call it asynchronously from Javascript. You can call a method asynchronously by appending `$async` to the method name. E.g. With the Google Maps example above, you would change :

```
this.$GLOBAL$.com_codename1_goolglemaps_MapContainer.fireMapChangeEvent__int_int_double_double;
```

to

```
this.$GLOBAL$.com_codename1_goolglemaps_MapContainer.fireMapChangeEvent__int_int_double_double$async;
```

This will cause the call to be wrapped in the appropriate bootstrap code to work properly with threads - and it is absolutely necessary in cases where the method **may** use threads of any kind. The side-effect of calling a method with the `$async` suffix is that you can't use return values from the method.



In most cases you should use the **async** version of a method when calling it from your native method. Only use the synchronous (default) version if you are absolutely sure that the method doesn't use any threading primitives.

16.6. Libraries - cn1lib

Support for JAR files in Codename One has been a source of confusion so its probably a good idea to revisit this subject again and clarify all the details.

The first source of confusion is changing the classpath. You should NEVER change the classpath or add an external JAR via the IDE classpath UI. The reasoning here is very simple, these IDE's don't package the JAR's into the final executable and even if they did these JAR's would probably use features unavailable or inappropriate for the device (e.g. `java.io.File` etc.).

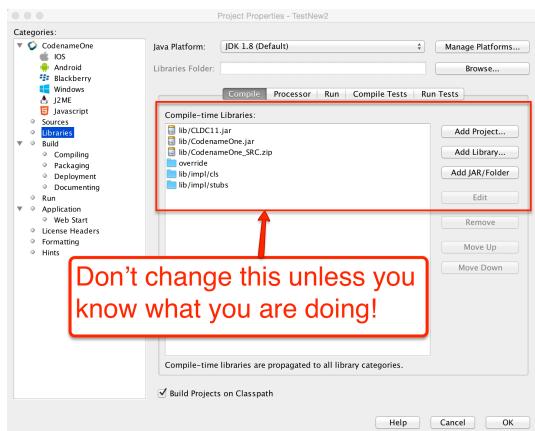


Figure 477. Don't change the classpath, this is how it should look for a typical Java 8 Codename One application

Cn1libs are Codename One's file format for 3rd party extensions. It's physically a zip file containing other zip files and some meta-data.

16.6.1. Why Not Use JAR?

A jar can be compiled with usage of any Java API that might not be supported, it can be compiled with a Java target version that isn't tested.

Jars don't include support for writing native code, you could use JNI in jars (awkwardly) but that doesn't match Codename One's needs for native support (see section above).

Jars don't support "proper" code completion, a common developer trick is to stick source code into the jar but that prevents usage with proprietary code. Cn1libs provide full IDE code completion (with JavaDoc hints) without exposing the sources.

There are two use cases for wanting JAR's and they both have very different solutions:

1. Modularity
2. Working with an existing JARs

Cn1lib's address the modularity aspect allowing you to break that down. Existing jars can sometimes be used native code settings but for the most part you would want to adapt the code to abide by Codename One restrictions.

16.6.2. How To Use cn1libs?

Codename One has a large repository of [3rd party cn1libs](https://www.codenameone.com/cn1libs.html) [<https://www.codenameone.com/cn1libs.html>], you can install a cn1lib by placing it in the lib directory of your project then right clicking the project and selecting **Codename One** → **Refresh cn1lib files**.

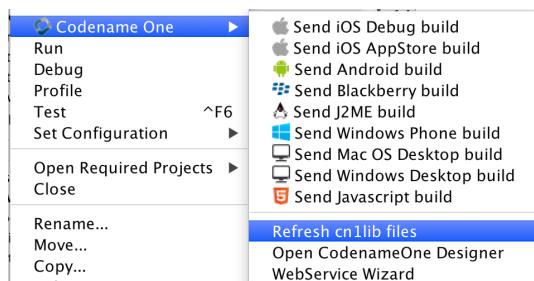


Figure 478. Refresh cn1lib files menu option

Once refreshed the content of the cn1lib will be available to code completion and you could just use it.



Notice that some cn1libs require additional configurations such as build hints etc. so make sure to read the developers instructions when integrating a 3rd party library.

Under the Hood of cn1lib Install

Refresh cn1lib files invokes the ant task `refresh-libs`. You could automatically trigger a refresh as part of your build process by invoking that ant task manually.

Technically that task invokes a custom task that unzips the content of the cn1lib into a set of directories accessible to the build process. Classes and stub sources are installed in `lib/impl/cls` & `lib/impl/stubs` respectively.

The native files are extracted to `lib/impl/native`. The classpath for the main project and the ant build process know about these directories and include them within their path.

16.6.3. Creating a Simple cn1lib

Creating a cn1lib is trivial, we will get into more elaborate uses soon enough but for a hello world cn1lib we can just use this 2 step process:

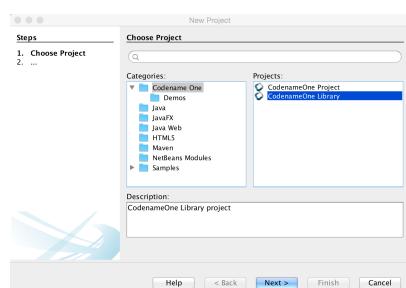


Figure 479. Select the CodenameOne Library Option

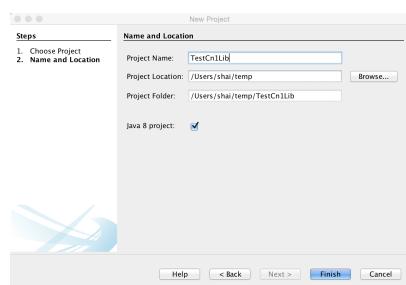


Figure 480. Select the file name/destination. Notice that a Java 8 cn1lib requires Java 8 support in the parent project!

Once we go thru these steps we can define any source file within the library and it will be accessible to the users of the library.

16.6.4. Build Hints in cn1libs

Some cn1libs are pretty simple to install, just place them under the lib directory and refresh. However, many of the more elaborate cn1libs need some pretty complex configurations. This is the case when native code is involved where we need to add permissions or plist entries for the various native platforms to get everything to work. This makes the cn1lib's helpful but less than seamless which is where we want to go.

Codename One cn1libs include two files that can be placed into the root:

`codenameone_library_required.properties` & `codenameone_library_appended.properties`.

In these files you can just write a build hint as `codename1.arg.ios.plistInject=…` for the various hints.



Notice the usage of the properties syntax for the build hint with the `codename1.arg` prefix you would also need to escape reserved characters for properties files.

The best way to discover the right syntax for such build hints is to set them via the build hints GUI in a regular project and copy/paste them from `codenameone_settings.properties` into the cn1lib file.

The obvious question is why do we need two files?

There are two types of build hints: required and appended.

Required build hints can be something like `ios.objC=true` which we want to always work. E.g. if a cn1lib defines `ios.objC=true` and another cn1lib defines `ios.objC=false` things won't work since one cn1lib won't get what it needs...

In this case we'd want the build to fail so we can remove the faulty cn1lib.



If two cn1libs define `ios.objC=true` there will be no collision as the value would be identical

An appended property would be something like `ios.plistInject=<key>UIBackgroundModes</key><array><string>audio</string> </array>`

Notice that this can still collide e.g. if a different cn1lib defines its own background mode. However, there are many valid cases where `ios.plistInject` can be used for other things. In this case we'll append the content of the `ios.plistInject` into the build hint if it's not already there.

There are a couple of things you need to keep in mind:

- Properties are merged with every "refresh libs" call not dynamically on the server. This means it should be pretty simple for the developer to investigate issues in this process.
- Changing flags is problematic - there is no "uninstall" process. Since the data is copied into the `codenameone_settings.properties` file. If you need to change a flag later on you might need to alert users to make changes to their properties essentially negating the value of this feature... So be very careful when adding properties here.

It's your responsibility as a library developer to decide which build hint goes into which file! Codename One can't automate this process as the whole process of build hints is by definition an ad hoc process.

The rule of thumb is that a build hint with a numeric or boolean value is always a required property. If an entry has a string that you can append with another string then its probably an appended entry.

These build hints are probably of the "required" type:

android.debug
android.release
android.installLocation
android.licenseKey
android.stack_size
android.statusbar_hidden
android.googleAdUnitId
android.includeGPlayServices
android.headphoneCallback
android.gpsPermission
android.asyncPaint
android.supportV4
android.theme
android.cusom_layout1
android.versionCode
android.captureRecord
android.removeBasePermissions
android.blockExternalStoragePermission
android.min_sdk_version
android.smallScreens
android.streamMode
android.enableProguard
android.targetSDKVersion
android.web_loading_hidden
facebook.appId
ios.keyboardOpen
ios.project_type
ios.newStorageLocation
ios.prerendered_icon
ios.application_exits
ios.themeMode
ios.xcode_version
javascript.inject_proxy
javascript.minifying
javascript.proxy.url
javascript.sourceFilesCopied
javascript.teavm.version
rim.askPermissions
google.adUnitId
ios.includePush
ios.headphoneCallback
ios.enableAutoplayVideo
ios.googleAdUnitId
ios.googleAdUnitIdPadding
ios.enableBadgeClear
ios.locationUsageDescription
ios.bundleVersion
ios.objC
ios.testFlight
desktop.width
desktop.height

```
desktop.adaptToRetina  
desktop.resizable  
desktop.fontSizes  
desktop.theme  
desktop.themeMac  
desktop.themeWin  
desktop.windowsOutput  
noExtraResources  
j2me.iconSize  
android.permission.<PERMISSION_NAME>
```

These build hints should probably be appended:

```
android.xapplication  
android.xpermissions  
android.xintent_filter  
android.facebook_permissions  
android.stringsXml  
android.style  
android.nonconsumable  
android.xapplication_attr  
android.xactivity  
android.pushVibratePattern  
android.proguardKeep  
android.sharedUserId  
android.sharedUserLabel  
ios.urlScheme  
ios.interface_orientation  
ios.plistInject  
ios.facebook_permissions  
ios.applicationDidEnterBackground  
ios.viewDidLoad  
ios.g AppDelegateHeader  
ios.g AppDelegateBody  
ios.beforeFinishLaunching  
ios.afterFinishLaunching  
ios.add_libs
```

cn1lib Structure/File Format

The cb1lib file format is quite simple, it's a zip file containing zip files within it with fixed names to support the various features.

The table below covers the files that can/should be a part of a cn1lib file:

Table 12. cn1lib structure

File Name	Required	Purpose
main.zip	□	Contains the bytecode and the library binary data. This is effectively the portable portion of the jar
stubs.zip	□	Stub source files (auto-generated) containing javadocs to provide code completion
manifest.properties	×	General properties of the library, this isn't used for much at the moment
codenameone_library_appended.properties	×	Discussed above
codenameone_library_required.properties	×	Discussed above
nativeios.zip	×	Native iOS sources if applicable
nativeand.zip	×	Native Android sources if applicable
nativejavascript.zip	×	Native JavaScript sources if applicable
nativerim.zip	×	Native RIM sources if applicable
nativese.zip	×	Native JavaSE sources if applicable
nativewin.zip	×	Native Windows sources if applicable
nativeme.zip	×	Native Java ME sources if applicable

16.7. Integrating Android 3rd Party Libraries & JNI

While its pretty easy to use native interfaces to write Android native code some things aren't necessarily as obvious. E.g. if you want to integrate a 3rd party library, specifically one that includes native C JNI code this process isn't as straightforward.

If you need to integrate such a library into your native calls you have the following options:

1. The first option (and the easiest one) is to just place a Jar file in the native/android directory. This will link your binary with the jar file. Just place the jar under the native/android and the build server will pick it up and will add it to the classpath.
Notice that Android release apps are obfuscated by default which might cause issues with such libraries if they reference API's that are unavailable on Android. You can workaround this by adding a build hint to the proguard obfuscation code that blocs the obfuscation of the

problematic classes using the build hint:

```
android.proguardKeep=-keep class com.mypackage.ProblemClass { *; }`
```

2. Another option is the **aar** file is a binary format Google introduced to represent an Android Library project (similarly to the cn1lib format). One of the problem with the Android Library projects was the fact that it required the project sources which made it difficult for 3rd party vendors to publish libraries.

As a result so android introduced the **aar** file which is a binary format that represents a Library project. To learn more about arr you can read [this](https://developer.android.com/studio/projects/android-library.html#aar-contents) [https://developer.android.com/studio/projects/android-library.html#aar-contents].

You can link an **aar** file by placing it under the native/android and the build server will link it to the project.

3. There is another **obsolete approach** that we are mentioning for legacy purposes (e.g. if you need to port code written with this legacy option). This predated the **aar** option from Google... Not all 3rd party tools can be packaged as a simple jar, some 3rd party tools need to declare activities add permissions, resources, assets, and/or even add native code (**.so** files).

To link a Library project to your Codename One project open the Library project in Eclipse or Android Studio and make sure the project builds, after the project was built successfully remove the bin directory from the project and zip the whole project.

Rename the extension from **.zip** to **.andlib** and place the andlib file under the **native/android** directory. The build server will pick it up and will link it to the project.

16.8. Drag & Drop

Unlike other platforms that tried to create overly generic catch all API's Codename One tried to make things as simple as possible.

In Codename One only components can be dragged and drop targets are always components. The logic of actually performing the operation indicated by the drop is the responsibility of the person implementing the drop.



Some platforms e.g. AWT allow dragging abstract concepts such as mime type elements. This allows dragging things like a text file into the app, but that use case isn't realistic in mobile

The code below allows you to rearrange the items based on a sensible order. Notice it relies on the default **Container** drop behavior.

```

Form hi = new Form("Rearrangeable Items", new BorderLayout());
String[] buttons = {"A Game of Thrones", "A Clash Of Kings", "A Storm Of Swords",
    "A Feast For Crows", "A Dance With Dragons", "The Winds of Winter", "A Dream of Spring" };

Container box = new Container(BoxLayout.y());
box.setScrollableY(true);
box.setDropTarget(true);
java.util.List<String> got = Arrays.asList(buttons);
Collections.shuffle(got);
for(String current : got) {
    MultiButton mb = new MultiButton(current);
    box.add(mb);
    mb.setDraggable(true);
}

hi.add(BorderLayout.NORTH, "Arrange The Titles").add(BorderLayout.CENTER, box);
hi.show();

```

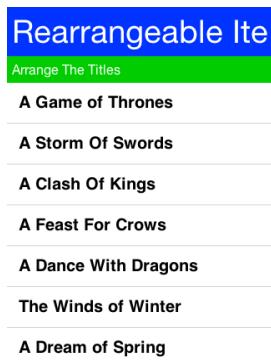


Figure 481. Drag and drop demo

To enable dragging a component it must be flagged as draggable using `setDraggable(true)`, to allow dropping the component onto another component you must first enable the drop target with `setDropTarget(true)` and override some methods (more on that later).

When dragging on top of a child component of a drop target the code recursively searches for a drop target parent. Dropping a component on the child will automatically find the right drop target, hence there is no need to make "everything" into a drop target.

You can override these methods in the draggable components:

- `getDragImage` - this generates an image preview of the component that will be dragged. This automatically generates a sensible default so you don't need to override it.
- `drawDraggedImage` - this method will be invoked to draw the dragged image at a given location, it might be useful to override it if you want to display some drag related information such an additional icon based on location etc. (e.g. a move/copy icon).

In the drop target you can override the following methods:

- `draggingOver` - returns true if a drop operation at this point is permitted. Otherwise releasing the component will have no effect.

- `dragEnter/Exit` - useful to track and cleanup state related to dragging over a specific component.
- `drop` - the logic for dropping/moving the component must be implemented here!

16.9. Continuous Integration & Release Engineering

Codename One was essentially built for continuous integration since the build servers are effectively a building block for such an architecture. However, there are several problems with that: the first of which is limited server capacity.

If all users would start sending builds with every commit the servers would instantly become unusable due to the heavy load. To circumvent this CI support is limited only on the Enterprise level which allows Codename One to stock more servers and cope with the rise in demand related to the feature.

To integrate with any CI solution just use the standard Ant targets such as `build-for-android-device`, `build-for-iphone-device` etc.

Normally, this would be a problem since the build is sent but since it isn't blocking you wouldn't get the build result and wouldn't be able to determine if the build passed or failed. To enable this just edit the build XML and add the attribute `automated="true"` to the `codeNameOne` tag in the appropriate targets.

This will deliver a `result.zip` file under the `dist` folder containing the binaries of a successful build. It will also block until the build is completed. This should be pretty easy to integrate with any CI system together with our automated testing solutions .

E.g. we can do a synchronous build like this:

```
<target name="build-for-javascript-sync" depends="clean,copy-javascript-override,copy-libs,jar,clean-override">
  <codeNameOne
    jarFile="${dist.jar}"
    displayName="${codename1.displayName}"
    packageName = "${codename1.packageName}"
    mainClassName = "${codename1.mainName}"
    version="${codename1.version}"
    icon="${codename1.icon}"
    vendor="${codename1.vendor}"
    subtitle="${codename1.secondaryTitle}"
    automated="true"
    targetType="javascript"
  />
</target>
```

This allows us to build a JavaScript version of the app automatically as part of a release build script.

16.10. Android Lollipop ActionBar Customization

When running on Android Lollipop (5.0 or newer) the native action bar will use the Lollipop design. This isn't applicable if you use the [Toolbar](https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html>] or [SideMenuBar](https://www.codenameone.com/javadoc/com/codename1/ui/SideMenuBar.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/SideMenuBar.html>]

this will be used only in the task switcher.

To customize the colors of the native `ActionBar` on Lollipop define a `colors.xml` file in the `native/android` directory of your project. It should look like this:

```
<resources>
    <color name="colorPrimary">#ff00ff00</color>
    <color name="colorPrimaryDark">#80ff0000</color>
    <color name="colorAccent">#800000ff</color>
</resources>
```

16.11. Intercepting URL's On iOS & Android

A common trick in mobile application development, is communication between two unrelated applications.

In Android we can use intents which are pretty elaborate and can be used via `Display.execute`, however what if you would like to expose the functionality of your application to a different application running on the device. This would allow that application to launch your application.

This isn't something we builtin to Codename One, however it does expose enough of the platform capabilities to enable that functionality rather easily on Android.

On Android we need to define an intent filter which we can do using the `android.xintent_filter` build hint, this accepts the XML to filter whether a request is relevant to our application:

```
android.xintent_filter=<intent-filter>  <action android:name="android.intent.action.VIEW" />  <category
    android:name="android.intent.category.DEFAULT" />  <category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="myapp" />  </intent-filter>
```

You can read more about it in [this stack overflow question](#) [<http://stackoverflow.com/questions/11421048/android-ios-custom-uri-protocol-handling>].

To bind the `myapp://` URL to your application. As a result typing `myapp://x` into the Android browser will launch the application.

16.11.1. Passing Launch Arguments To The App

You can access the value of the URL that launched the app using:

```
String arg = Display.getInstance().getProperty("AppArg");
```

This value would be null if the app was launched via the icon.

iOS is practically identical to Android with some small caveats, iOS's equivalent of the manifest is the plist.

You can inject more data into the plist by using the `ios.plistInject` build hint.

So the equivalent in the iOS side would be

```
ios.plistInject=<key>CFBundleURLTypes</key>    <array>          <dict>            <key>CFBundleURLName</key>
<string>com.yourcompany.myapp</string>      </dict>          <key>CFBundleURLSchemes</key>
<array>          <string>myapp</string>        </array>        </dict>    </array>
```

However, that can conflict with the Facebook integration if you use `FacebookConnect` which needs access to the schemes. To workaround it you can use the build hint `ios.urlScheme` e.g.:

```
ios.urlScheme=<string>myapp</string>
```

16.12. Native Peer Components

Many Codename One developers don't truly grasp the reason for the separation between peer (native) components and Codename One components. This is a crucial thing you need to understand especially if you plan on working with native widgets e.g. Web Browser, native maps, text input, media and native interfaces (which can return a `PeerComponent`).

Codename One draws all of its widgets on its own, this is a concept which was modeled in part after Swing. This allows functionality that can't be achieved in native widget platforms:

1. The Codename One GUI builder & simulator are almost identical to the device - notice that this also enables the build cloud, otherwise device specific bugs would overwhelm development and make the build cloud redundant.
2. Ability to override everything - paint, pointer, key events are all overridable and replaceable. Developers can also paint over everything e.g. glasspane and layered pane.
3. Consistency - provides identical functionality on all platforms for the most part.

This all contributes to our ease of working with Codename One and maintaining Codename One. More than 95% of Codename One's code is in Java hence its really portable and pretty easy to maintain!

16.12.1. Why does Codename One Need Native Widgets at all?

We need the native device to do input, html rendering etc. these are just too big and too complex tasks for Codename One to do from scratch.

They are sometimes impossible to perform without the native platform. E.g. the virtual keyboard input on the devices is tied directly to the native text input. It's impractical to implement everything from scratch for all languages, dictionaries etc. The result would be sub-par.

A web browser can't be implemented in this day and age without a JavaScript JIT and including a JIT within an iOS app is prohibited by Apple.

16.12.2. So what's the problems with native widgets?

Codename One does pretty much everything on the EDT (Event Dispatch Thread), this provides a lot of cool features e.g. modal dialogs, invokeAndBlock etc.

However native widgets have to be drawn on the devices native UI thread.
This means that drawing looks something like this:

1. Loop over all Codename One components and paint them.
2. Loop over all native peer components and paint them.

This effectively means that all peer components are drawn on top of the Codename One components.



This was also the case in AWT/Swing to one degree or another...

16.12.3. So how do we show dialogs on top of Peer Components?

Codename One grabs a screenshot of the peer, hide it and then we can just show the screenshot. Since the screenshot is static it can be rendered via the standard UI. Naturally we can't do that always since grabbing a screenshot is an expensive process on all platforms and must be performed on the native device thread.

16.12.4. Why can't we combine peer component scrolling and Codename One scrolling?

Since the form title/footer etc. are drawn by Codename One the peer component might paint itself on top of them. Clipping a peer component is often pretty difficult. Furthermore, if the user drags his finger within the peer component he might trigger the native scroll within the might collide with our scrolling?

16.12.5. Native Components In The First Form

There is also another problem that might be counter intuitive. iOS has screenshot images representing the first form. If your first page is an HTML or a native map (or other peer widget) the screenshot process on the build server will show fallback code instead of the real thing thus providing sub-par behavior.

Its impractical to support something like HTML for the screenshot process since it would also look completely different from the web component running on the device.



You can read more about the screenshot process [here](https://www.codenameone.com/manual/appendix-ios.html#section-ios-screenshots) [https://www.codenameone.com/manual/appendix-ios.html#section-ios-screenshots].

16.13. Integrating 3rd Party Native SDKs

The following is a description of the procedure that was used to create the [Codename One FreshDesk library](#) [http://shannah.github.io/cn1-freshdesk/]. This process can be easily adapted to wrap

any native SDK on Android and iOS.

16.13.1. Step 1 : Review the FreshDesk SDKs

Before we begin, we'll need to review the Android and iOS SDKs.

1. **FreshDesk Android SDK:** [Integration Guide](http://developer.freshdesk.com/mobihelp/android/integration_guide/) [http://developer.freshdesk.com/mobihelp/android/integration_guide/] | [API Docs](http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/package-summary.html) [<http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/package-summary.html>]
2. **FreshDesk iOS SDK:** [Integration Guide](http://developer.freshdesk.com/mobihelp/ios/integration_guide/) [http://developer.freshdesk.com/mobihelp/ios/integration_guide/] | [API Docs](http://developer.freshdesk.com/mobihelp/ios/api/) [<http://developer.freshdesk.com/mobihelp/ios/api/>]

In reviewing the SDKs, we are looking for answers to two questions:

1. What should my Codename One FreshDesk API look like?
2. What will be involved in integrating the native SDK in my app or lib?

16.13.2. Step 2: Designing the Codename One Public API

When designing the Codename One API, we should begin by looking at the [Javadocs](http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/package-summary.html) [<http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/package-summary.html>] for the native Android SDK. If the class hierarchy doesn't look too elaborate, we may decide to model our Codename One public API fairly closely on the Android API. On the other hand, if we only need a small part of the SDK's functionality, we may choose to create my abstractions around just the functionality that we need.

In the case of the FreshDesk SDK, it looks like most of the functionality is handled by one central class [Mobihelp](#), with a few other POJO classes for passing data to and from the service. This is a good candidate for a comprehensive Codename One API.

Before proceeding, we also need to look at the iOS API to see if there are any features that aren't included. While naming conventions in the iOS API are a little different than those in the Android API, it looks like they are functionally the same.

Therefore, I choose to create a class hierarchy and API that closely mirrors the Android SDK.

16.13.3. Step 3: The Architecture and Internal APIs

A Codename One library that wraps a native SDK, will generally consist of the following:

1. **Public Java API**, consisting of pure Java classes that are intended to be used by the outside world.
2. **Native Interface(s)**. The Native Interface(s) act as a conduit for the public Java API to communicate to the native SDK. Parameters in native interface methods are limited to primitive types, arrays of primitive types, and Strings, as are return values.
3. **Native code**. Each platform must include an implementation of the Native Interface(s). These implementations are written in the native language of the platform (e.g. Java for Android, and Objective-C for iOS).

4. **Native dependencies.** Any 3rd party libraries required for the native code to work, need to be included for each platform. On android, this may mean bundling .jar files, .aar files, or .andlib files. On iOS, this may mean bundling .h files, .a files, .framework, and .bundle files.
5. **Build hints.** Some libraries will require you to add some extra build hints to your project. E.g. On Android you may need to add permissions to the manifest, or define services in the `<Application>` section of the manifest. On iOS, this may mean specifying additional core frameworks for inclusion, or adding build flags for compilation.

The following diagram shows the dependencies in a native library:

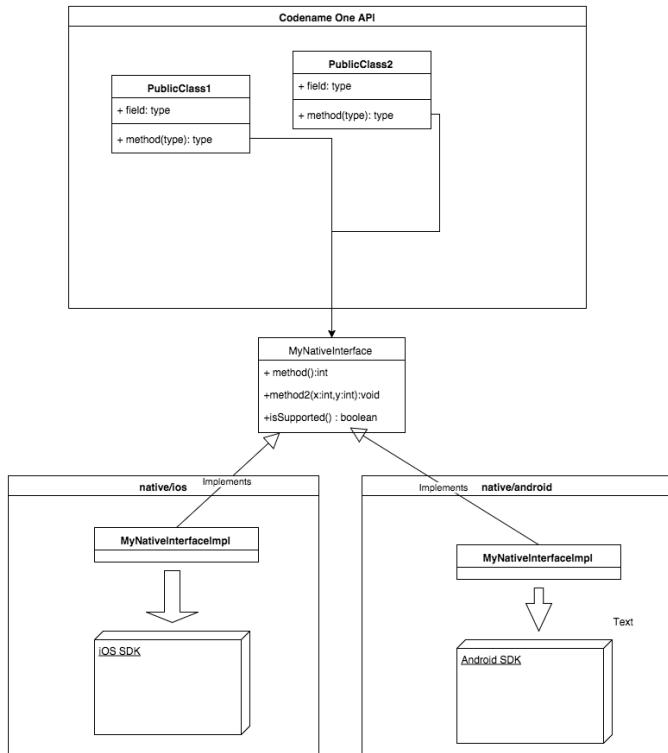


Figure 482. Relationship between native & Codename One API UML Diagram

In the specific case of our FreshDesk API, the public API and classes will look like:

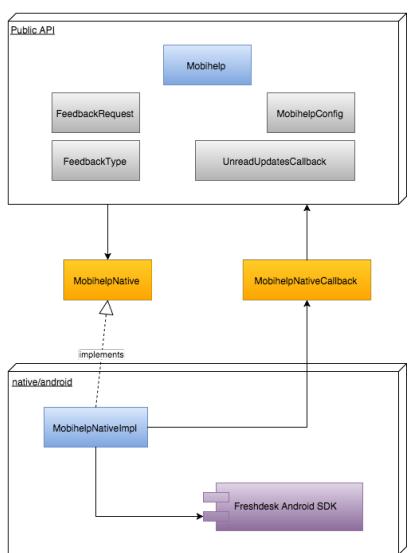


Figure 483. Freshdesk API Integration

Things to Notice

1. The public API consists of the main class ([Mobihelp](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/Mobihelp.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/Mobihelp.java]), and a few supporting classes ([FeedbackRequest](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/FeedbackRequest.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/FeedbackRequest.java], [FeedbackType](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/FeedbackType.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/FeedbackType.java], [MobihelpConfig](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpConfig.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpConfig.java], [MobihelpCallbackStatus](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpCallbackStatus.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpCallbackStatus.java]), which were copied almost directly from the Android SDK.
2. The only way for the public API to communicate with the native SDK is via the [MobihelpNative](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNative.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNative.java] interface.
3. We introduced the [MobihelpNativeCallback](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNativeCallback.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNativeCallback.java] class to facilitate native code calling back into the public API. This was necessary for a few methods that used asynchronous callbacks.

16.13.4. Step 4: Implement the Public API and Native Interface

We have already looked at the final product of the public API in the previous step, but let's back up and walk through the process step-by-step.

I wanted to model my API closely around the Android API, and the central class that includes all of the functionality of the SDK is the [com.freshdesk.mobihelp.Mobihelp](http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/Mobihelp.html) class [http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/Mobihelp.html], so we begin there.

We'll start by creating our own package (<com.codename1.freshdesk>) and our own [Mobihelp](#) class inside it.

Adapting Method Signatures

The [Context](#) parameter

In a first glance at the [com.freshdesk.mobihelp.Mobihelp API](http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/Mobihelp.html) [http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/Mobihelp.html] we see that many of the methods take a parameter of type [android.content.Context](http://developer.android.com/reference/android/content/Context.html) [http://developer.android.com/reference/android/content/Context.html]. This class is part of the core Android SDK, and will not be accessible to any pure Codename One APIs. Therefore, our public API cannot include any such references. Luckily, we'll be able to access a suitable context in the native layer, so we'll just omit this parameter from our public API, and inject them in our native implementation.

Hence, the method signature `public static final void setUserFullName (Context context, String name)` will simply become `public static final void setUserFullName (String name)` in our public API.

Non-Primitive Parameters

Although our public API isn't constrained by the same rules as our Native Interfaces with respect to parameter and return types, we need to be cognizant of the fact that parameters we pass to our public API will ultimately be funnelled through our native interface. Therefore, we should pay attention to any parameters or return types that can't be passed directly to a native interface, and start forming a strategy for them. E.g. consider the following method signature from the Android [Mobihelp](#) class:

```
public static final void showSolutions (Context activityContext, ArrayList<String> tags)
```

We've already decided to just omit the [Context](#) parameter in our API, so that's a non-issue. But what about the [ArrayList<String>](#) tags parameter? Passing this to our public API is no problem, but when we implement the public API, how will we pass this [ArrayList](#) to our native interface, since native interfaces don't allow us to arrays of strings as parameters?

I generally use one of three strategies in such cases:

1. Encode the parameter as either a single [String](#) (e.g. using JSON or some other easily parseable format) or a `byte[]` array (in some known format that can easily be parsed in native code).
2. Store the parameter on the Codename One side and pass some ID or token that can be used on the native side to retrieve the value.
3. If the data structure can be expressed as a finite number of primitive values, then simply design the native interface method to take the individual values as parameters instead of a single object. E.g. If there is a [User](#) [<https://www.codenameone.com/javadoc/com/codenname1/facebook/User.html>] class with properties `name` and `phoneNumber`, the native interface can just have `name` and `phoneNumber` [parameters rather than a single 'user' parameter](#).

In this case, because an array of strings is such a simple data structure, I decided to use a variation on strategy number 1: Merge the array into a single string with a delimiter.

In any case, we don't have to come up with the specifics right now, as we are still on the public API, but it will pay dividends later if we think this through ahead of time.

Callbacks

It is quite often the case that native code needs to call back into Codename One code when an event occurs. This may be connected directly to an API method call (e.g. as the result of an asynchronous method invocation), or due to something initiated by the operating system or the native SDK on its own (e.g. a push notification, a location event, etc..).

Native code will have access to both the Codename One API and any native APIs in your app, but on some platforms, accessing the Codename One API may be a little tricky. E.g. on iOS you'll be calling from Objective-C back into Java which requires knowledge of Codename One's java-to-objective C conversion process. In general, I have found that the easiest way to facilitate callbacks is to provide abstractions that involve static java methods (in Codename One space) that accept and return primitive types.

In the case of our `Mobihelp` class, the following method hints at the need to have a "callback plan":

```
public static final void getUnreadCountAsync (Context context, UnreadUpdatesCallback callback)
```

The interface definition for `UnreadUpdatesCallback` is:

```
public interface UnreadUpdatesCallback {  
    //This method is called once the unread updates count is available.  
    void onResult(MobihelpCallbackStatus status, Integer count);  
}
```

I.e. If we were to implement this method (which I plan to do), we need to have a way for the native code to call the `callback.onResult()` method of the passed parameter.

So we have two issues that will need to be solved here:

1. How to pass the `callback` object through the native interface.
2. How to call the `callback.onResult()` method from native code at the right time.

For the first issue, we'll use strategy #2 that we mentioned previously: (Store the parameter on the Codename One side and pass some ID or token that can be used on the native side to retrieve the value).

For the second issue, we'll create a static method that can take the token generated to solve the first issue, and call the stored `callback` object's `onResult()` method. We abstract both sides of this process using the `MobihelpNativeCallback` class [<https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNativeCallback.java>].

```

public class MobihelpNativeCallback {
    private static int nextId = 0;
    private static Map<Integer,UnreadUpdatesCallback> callbacks = new HashMap<Integer,UnreadUpdatesCallback>();

    static int registerUnreadUpdatesCallback(UnreadUpdatesCallback callback) {
        callbacks.put(nextId, callback);
        return nextId++;
    }

    public static void fireUnreadUpdatesCallback(int callbackId, final int status, final int count) {
        final UnreadUpdatesCallback cb = callbacks.get(callbackId);
        if (cb != null) {
            callbacks.remove(callbackId);
            Display.getInstance().callSerially(new Runnable() {

                public void run() {
                    MobihelpCallbackStatus status2 = MobihelpCallbackStatus.values()[status];
                    cb.onResult(status2, count);
                }
            });
        }
    }
}

```

Things to notice here:

1. This class uses a static `Map<Integer,UnreadUpdatesCallback>` member to keep track of all callbacks, mapping a unique integer ID to each callback.
2. The `registerUnreadUpdatesCallback()` method takes an `UnreadUpdatesCallback` object, places it in the `callbacks` map, and returns the integer **token** that can be used to fire the callback later. This method would be called by the public API inside the `getUnreadCountAsync()` method implementation to convert the `callback` into an integer, which can then be passed to the native API.
3. The `fireUnreadUpdatesCallback()` method would be called later from native code. Its first parameter is the token for the callback to call.
4. We wrap the `onResult()` call inside a `Display.callSerially()` invocation to ensure that the callback is called on the EDT. This is a general convention that is used throughout Codename One, and you'd be well-advised to follow it. **Event handlers** should be run on the EDT unless there is a good reason not to - and in that case your documentation and naming conventions should make this clear to avoid accidentally stepping into multithreading hell!

Initialization

Most Native SDKs include some sort of initialization method where you pass your developer and application credentials to the API. When I filled in FreshDesk's web-based form to create a new application, it generated an application ID, an app "secret", and a "domain". The SDK requires me to pass all three of these values to its `init()` method via the `MobihelpConfig` class.

Note, however, that FreshDesk (and most other service providers that have native SDKs) requires me to create different Apps for each platform. This means that my App ID and App secret will be

different on iOS than they will be on Android.

Therefore our public API needs to enable us to provide multiple credentials in the same app, and our API needs to know to use the correct credentials depending on the device that the app is running on.

There are many solutions to this problem, but the one I chose was to provide two different `init()` methods:

```
public final static void initIOS(MobihelpConfig config)
```

and

```
public final static void initAndroid(MobihelpConfig config)
```

Then I can set up the API with code like:

```
MobihelpConfig config = new MobihelpConfig();
config.setAppSecret("xxxxxxxx");
config.setAppId("freshdeskdemo-2-xxxxxx");
config.setDomain("codenameonetest1.freshdesk.com");
Mobihelp.initIOS(config);

config = new MobihelpConfig();
config.setAppSecret("yyyyyyyy");
config.setAppId("freshdeskdemo-1-yyyyyyyy");
config.setDomain("https://codenameonetest1.freshdesk.com");
Mobihelp.initAndroid(config);
```

The Resulting Public API

```
public class Mobihelp {

    private static char[] separators = new char[]{' ',',','|','/','@','#','%','!','^','&','*','=','+','*','<'};

    private static MobihelpNative peer;

    public static boolean isSupported() {
        ....
    }

    public static void setPeer(MobihelpNative peer) {
        ....
    }

    //Attach the given custom data (key-value pair) to the conversations/tickets.
    public final static void addCustomData(String key, String value) {
        ...
    }
    //Attach the given custom data (key-value pair) to the conversations/tickets with the ability to flag sensitive data.
    public final static void addCustomData(String key, String value, boolean isSensitive) {
        ...
    }
}
```

```

//Clear all breadcrumb data.
public final static void    clearBreadCrumbs() {
    ...
}

//Clear all custom data.
public final static void    clearCustomData() {
    ...
}

//Clears User information.
public final static void    clearUserData() {
    ...
}

//Retrieve the number of unread items across all the conversations for the user synchronously i.e.
public final static int getUnreadCount() {
    ...
}

//Retrieve the number of unread items across all the conversations for the user asynchronously, count is delivered to
//the supplied UnreadUpdatesCallback instance Note : This may return 0 or stale value when there is no network connectivity
etc
public final static void    getUnreadCountAsync(UnreadUpdatesCallback callback) {
    ...
}

//Initialize the Mobihelp support section with necessary app configuration.
public final static void    initAndroid(MobihelpConfig config) {
    ...
}

public final static void initIOS(MobihelpConfig config) {
    ...
}

//Attaches the given text as a breadcrumb to the conversations/tickets.
public final static void    leaveBreadcrumb(String crumbText) {
    ...
}

//Set the email of the user to be reported on the Freshdesk Portal
public final static void    setUserEmail(String email) {
    ...
}

//Set the name of the user to be reported on the Freshdesk Portal.
public final static void    setUserFullName(String name) {
    ...
}

//Display the App Rating dialog with option to Rate, Leave feedback etc
public static void    showAppRateDialog() {
    ...
}

//Directly launch Conversation list screen from anywhere within the application
public final static void    showConversations() {
    ...
}

//Directly launch Feedback Screen from anywhere within the application.
public final static void    showFeedback(FeedbackRequest feedbackRequest) {
    ...
}

//Directly launch Feedback Screen from anywhere within the application.
public final static void    showFeedback() {
    ...
}

//Displays the Support landing page (Solution Article List Activity) where only solutions tagged with the given tags
are displayed.
public final static void    showSolutions(ArrayList<String> tags) {
    ...
}

```

```

}

private static String findUnusedSeparator(ArrayList<String> tags) {
    ...
}

//Displays the Support landing page (Solution Article List Activity) from where users can do the following
//View solutions,
//Search solutions,
public final static void    showSolutions() {
    ...
}

//Displays the Integrated Support landing page where only solutions tagged with the given tags are displayed.
public final static void    showSupport(ArrayList<String> tags) {
    ...
}

//Displays the Integrated Support landing page (Solution Article List Activity) from where users can do the following
//View solutions,
//Search solutions,
// Start a new conversation,
// View existing conversations update/ unread count etc
public final static void    showSupport() {
    ...
}

}

```

The Native Interface

The final native interface is nearly identical to our public API, except in cases where the public API included non-primitive parameters.

```

public interface MobihelpNative extends NativeInterface {

    /**
     * @return the appId
     */
    public String config_getAppId();

    /**
     * @param appId the appId to set
     */
    public void config_setAppId(String appId);

    /**
     * @return the appSecret
     */
    public String config_getAppSecret();

    /**
     * @param appSecret the appSecret to set
     */
    public void config_setAppSecret(String appSecret);
    /**
     * @return the domain
     */
    public String config_getDomain();
    /**
     * @param domain the domain to set
     */
    public void config_setDomain(String domain) ;
}

```

```

/**
 * @return the feedbackType
 */
public int config_getFeedbackType() ;

/**
 * @param feedbackType the feedbackType to set
 */
public void config_setFeedbackType(int feedbackType);

/**
 * @return the launchCountForReviewPrompt
 */
public int config_getLaunchCountForReviewPrompt() ;
/**
 * @param launchCountForReviewPrompt the launchCountForReviewPrompt to set
 */
public void config_setLaunchCountForReviewPrompt(int launchCountForReviewPrompt);
/**
 * @return the prefetchSolutions
 */
public boolean config_isPrefetchSolutions();
/**
 * @param prefetchSolutions the prefetchOptions to set
 */
public void config_setPrefetchSolutions(boolean prefetchSolutions);
/**
 * @return the autoReplyEnabled
 */
public boolean config_isAutoReplyEnabled();

/**
 * @param autoReplyEnabled the autoReplyEnabled to set
 */
public void config_setAutoReplyEnabled(boolean autoReplyEnabled) ;

/**
 * @return the enhancedPrivacyModeEnabled
 */
public boolean config_isEnhancedPrivacyModeEnabled() ;

/**
 * @param enhancedPrivacyModeEnabled the enhancedPrivacyModeEnabled to set
 */
public void config_setEnhancedPrivacyModeEnabled(boolean enhancedPrivacyModeEnabled) ;

//Attach the given custom data (key-value pair) to the conversations/tickets.
public void addCustomData(String key, String value);
//Attach the given custom data (key-value pair) to the conversations/tickets with the ability to flag sensitive data.
public void addCustomDataWithSensitivity(String key, String value, boolean isSensitive);
//Clear all breadcrumb data.
public void clearBreadCrumbs() ;
//Clear all custom data.
public void clearCustomData();
//Clears User information.
public void clearUserData();
//Retrieve the number of unread items across all the conversations for the user synchronously i.e.
public int getUnreadCount();

//Retrieve the number of unread items across all the conversations for the user asynchronously, count is delivered to
//the supplied UnreadUpdatesCallback instance Note : This may return 0 or stale value when there is no network connectivity
//etc
public void getUnreadCountAsync(int callbackId);

```

```

public void initNative();

//Attaches the given text as a breadcrumb to the conversations/tickets.
public void leaveBreadCrumb(String crumbText);
//Set the email of the user to be reported on the Freshdesk Portal

public void setUserEmail(String email);

//Set the name of the user to be reported on the Freshdesk Portal.
public void setUserName(String name);

//Display the App Rating dialog with option to Rate, Leave feedback etc
public void showAppRateDialog();
//Directly launch Conversation list screen from anywhere within the application
public void showConversations();

//Directly launch Feedback Screen from anywhere within the application.
public void showFeedbackWithArgs(String subject, String description);
//Directly launch Feedback Screen from anywhere within the application.
public void showFeedback();

//Displays the Support landing page (Solution Article List Activity) where only solutions tagged with the given tags
are displayed.
public void showSolutionsWithTags(String tags, String separator);

//Displays the Support landing page (Solution Article List Activity) from where users can do the following
//View solutions,
//Search solutions,
public void showSolutions();
//Displays the Integrated Support landing page where only solutions tagged with the given tags are displayed.
public void showSupportWithTags(String tags, String separator);

//Displays the Integrated Support landing page (Solution Article List Activity) from where users can do the following
//View solutions,
//Search solutions,
// Start a new conversation,
//View existing conversations update/ unread count etc
public void showSupport();
}

```

Notice also, that the native interface includes a set of methods with names prefixed with `config_`. This is just a naming conventions I used to identify methods that map to the `MobihelpConfig` class. I could have used a separate native interface for these, but decided to keep all the native stuff in one class for simplicity and maintainability.

Connecting the Public API to the Native Interface

So we have a public API, and we have a native interface. The idea is that the public API should be a thin wrapper around the native interface to smooth out rough edges that are likely to exist due to the strict set of rules involved in native interfaces. We'll, therefore, use delegation inside the `Mobihelp` class to provide it a reference to an instance of `MobihelpNative`:

```

public class Mobihelp {
    private static MobihelpNative peer;
    //...
}

```

We'll initialize this `peer` inside the `init()` method of the `Mobihelp` class. Notice, though that `init()` is

private since we have provided abstractions for the Android and iOS apps separately:

```
//Initialize the Mobihelp support section with necessary app configuration.
public final static void initAndroid(MobihelpConfig config) {
    if ("and".equals(Display.getInstance().getPlatformName())) {
        init(config);
    }
}

public final static void initIOS(MobihelpConfig config) {
    if ("ios".equals(Display.getInstance().getPlatformName())) {
        init(config);
    }
}

private static void init(MobihelpConfig config) {
    peer = (MobihelpNative)NativeLookup.create(MobihelpNative.class);
    peer.config_setAppId(config.getAppId());
    peer.config_setAppSecret(config.getAppSecret());
    peer.config_setAutoReplyEnabled(config.isAutoReplyEnabled());
    peer.config_setDomain(config.getDomain());
    peer.config_setEnhancedPrivacyModeEnabled(config.isEnhancedPrivacyModeEnabled());
    if (config.getFeedbackType() != null) {
        peer.config_setFeedbackType(config.getFeedbackType().ordinal());
    }
    peer.config_setLaunchCountForReviewPrompt(config.getLaunchCountForReviewPrompt());
    peer.config_setPrefetchSolutions(config.isPrefetchSolutions());
    peer.initNative();
}
```

Things to Notice:

1. The `initAndroid()` and `initIOS()` methods include a check to see if they are running on the correct platform. Ultimately they both call `init()`.
2. The `init()` method, uses the `NativeLookup` [<https://www.codenameone.com/javadoc/com/codename1/system/NativeLookup.html>] class to instantiate our native interface.

Implementing the Glue Between Public API and Native Interface

For most of the methods in the `Mobihelp` class, we can see that the public API will just be a thin wrapper around the native interface. E.g. the public API implementation of `setUserFullName(String)` is:

```
public final static void setUserFullName(String name) {
    peer.setUserFullName(name);
}
```

For some other methods, the public API needs to break apart the parameters into a form that the

native interface can accept. E.g. the `init()` method, shown above, takes a `MobihelpConfig` object as a parameter, but it passed the properties of the `config` object individually into the native interface.

Another example, is the `showSupport(ArrayList<String> tags)` method. The corresponding native interface method that is wraps is `showSupport(String tags, 'String separator)` - i.e it needs to merge all tags into a single delimited string, and pass then to the native interface along with the delimiter used. The implementation is:`

```
public final static void showSupport(ArrayList<String> tags) {
    String separator = findUnusedSeparator(tags);
    StringBuilder sb = new StringBuilder();
    for (String tag : tags) {
        sb.append(tag).append(separator);
    }
    peer.showSupportWithTags(sb.toString().substring(0, sb.length()-separator.length()), separator);
}
```

The only other non-trivial wrapper is the `getUnreadCountAsync()` method that we discussed before:

```
public final static void getUnreadCountAsync(UnreadUpdatesCallback callback) {
    int callbackId = MobihelpNativeCallback.registerUnreadUpdatesCallback(callback);
    peer.getUnreadCountAsync(callbackId);
}
```

16.13.5. Step 5: Implementing the Native Interface in Android

Now that we have set up our public API and our native interface, it is time to work on the native side of things. You can generate stubs for all platforms in your IDE (Netbeans in my case), by right clicking on the `MobihelpNative` class in the project explorer and selecting "Generate Native Access".

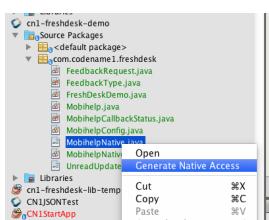


Figure 484. Generate Native Access Menu Item

This will generate a separate directory for each platform inside your project's `native` directory:

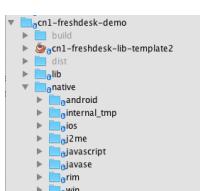


Figure 485. Native generated sources directory view

Inside the `android` directory, this generates a `com/codename1/freshdesk/MobihelpNativeImpl` class with stubs for each method.

Our implementation will be a thin wrapper around the native Android SDK. See the source [here](https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/android/com/codename1/freshdesk/MobihelpNativeImpl.java) [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/android/com/codename1/freshdesk/MobihelpNativeImpl.java].

Some highlights:

1. **Context** : The native API requires us to pass a **context** object as a parameter on many methods. This should be the context for the current activity. It will allow the FreshDesk API to know where to return to after it has done its thing. Codename One provides a class called **AndroidNativeUtil** that allows us to retrieve the app's Activity (which includes the Context). We'll wrap this with a convenience method in our class as follows:

```
private static Context context() {  
    return com.codename1.impl.android.AndroidNativeUtil.getActivity().getApplicationContext();  
}
```

This will enable us to easily wrap the freshdesk native API. E.g.:

```
public void clearUserData() {  
    com.freshdesk.mobihelp.Mobihelp.clearUserData(context());  
}
```

2. **runOnUiThread()** - Many of the calls to the FreshDesk API may have been made from the Codename One EDT. However, Android has its own event dispatch thread that should be used for interacting with native Android UI. Therefore, any API calls that look like they initiate some sort of native Android UI process should be wrapped inside Android's **runOnUiThread()** method which is similar to Codename One's **Display.callSerially()** method. E.g. see the **showSolutions()** method:

```
public void showSolutions() {  
    activity().runOnUiThread(new Runnable() {  
        public void run() {  
            com.freshdesk.mobihelp.Mobihelp.showSolutions(context());  
        }  
    });  
}
```

(Note here that the **activity()** method is another convenience method to retrieve the app's current **Activity** from the **AndroidNativeUtil** class).

3. **Callbacks**. We discussed, in detail, the mechanisms we put in place to enable our native code to perform callbacks into Codename One. You can see the native side of this by viewing the **getUnreadCountAsync()** method implementation:

```

public void getUnreadCountAsync(final int callbackId) {
    activity().runOnUiThread(new Runnable() {
        public void run() {
            com.freshdesk.mobihelp.Mobihelp.getUnreadCountAsync(context(), new
com.freshdesk.mobihelp.UnreadUpdatesCallback() {
                public void onResult(com.freshdesk.mobihelp.MobihelpCallbackStatus status, Integer count) {
                    MobihelpNativeCallback.fireUnreadUpdatesCallback(callbackId, status.ordinal(), count);
                }
            });
        }
    });
}

```

16.13.6. Step 6: Bundling the Native SDKs

The last step (at least on the Android side) is to bundle the FreshDesk SDK. For Android, there are a few different scenarios you'll run into for embedding SDKs:

- The SDK includes only Java classes** - NO XML UI files, assets, or resources that aren't included inside a simple .jar file. In this case, you can just place the .jar file inside your project's `native/android` directory.
- The SDK includes some XML UI files, resources, and assets.** In this case, the SDK is generally distributed as an Android project folder that can be imported into an Eclipse or Android studio workspace. In general, in this case, you would need to zip the entire directory and change the extension of the resulting .zip file to ".andlib", and place this in your project's `native/android` directory.
- The SDK is distributed as an .aar file** - In this case you can just copy the .aar file into your `native/android` directory.

The FreshDesk SDK

The FreshDesk (aka Mobihelp) SDK is distributed as a project folder (i.e. scenario 2 from the above list). Therefore, our procedure is to download the SDK ([download link](https://s3.amazonaws.com/assets.mobihelp.freshpo.com/sdk/mobihelp_sdk_android.zip) [https://s3.amazonaws.com/assets.mobihelp.freshpo.com/sdk/mobihelp_sdk_android.zip]), and rename it from `mobihelp_sdk_android.zip` to `mobihelp_sdk_android.andlib`, and copy it into our `native/android` directory.

Dependencies

Unfortunately, in this case there's a catch. The Mobihelp SDK includes a dependency:

Mobihelp SDK depends on AppCompat-v7 (Revision 19.0+) Library. You will need to update `project.properties` to point to the Appcompat library.

If we look inside the `project.properties` file (inside the Mobihelp SDK directory--- i.e. you'd need to extract it from the zip to view its contents), you'll see the dependency listed:

```
android.library.reference.1=../appcompat_v7
```

I.e. it is expecting to find the `appcompat_v7` library located in the same parent directory as the Mobihelp SDK project. After a little bit of research (if you're not yet familiar with the Android AppCompat support library), we find that the `AppCompat_v7` library is part of the Android Support library, which can be installed into your local Android SDK using Android SDK Manager. [Installation process specified here](https://developer.android.com/tools/support-library/setup.html) [<https://developer.android.com/tools/support-library/setup.html>].

After installing the support library, you need to retrieve it from your Android SDK. You can find that `.aar` file inside the `ANDROID_HOME/sdk/extras/android/m2repository/com/android/support/appcompat-v7/19.1.0/` directory (for version 19.1.0). The contents of that directory on my system are:

```
appcompat-v7-19.1.0.aar      appcompat-v7-19.1.0.pom  
appcompat-v7-19.1.0.aar.md5  appcompat-v7-19.1.0.pom.md5  
appcompat-v7-19.1.0.aar.sha1  appcompat-v7-19.1.0.pom.sha1
```

There are two files of interest here:

1. `appcompat-v7-19.1.0.aar` - This is the actual library that we need to include in our project to satisfy the Mobisdk dependency.
2. `appcompat-v7-19.1.0.pom` - This is the Maven XML file for the library. It will show us any dependencies that the `appcompat` library has. We will also need to include these dependencies:

```
<dependencies>  
  <dependency>  
    <groupId>com.android.support</groupId>  
    <artifactId>support-v4</artifactId>  
    <version>19.1.0</version>  
    <scope>compile</scope>  
  </dependency>  
</dependencies>
```

i.e. We need to include the `support-v4` library version 19.1.0 in our project. This is also part of the Android Support library. If we back up a couple of directories to: `ANDROID_HOME/sdk/extras/android/m2repository/com/android/support`, we'll see it listed there:

```
appcompat-v7          palette-v7  
cardview-v7          .recyclerview-v7  
gridlayout-v7         support-annotations  
leanback-v17          support-v13  
mediarouter-v7        support-v4  
multidex              test  
multidex-instrumentation
```

+ And if we look inside the appropriate version directory of `support-v4` (in `ANDROID_HOME/sdk/extras/android/m2repository/com/android/support/support-v4/19.1.0`), we see:

```

support-v4-19.1.0-javadoc.jar      support-v4-19.1.0.jar
support-v4-19.1.0-javadoc.jar.md5  support-v4-19.1.0.jar.md5
support-v4-19.1.0-javadoc.jar.sha1 support-v4-19.1.0.jar.sha1
support-v4-19.1.0-sources.jar     support-v4-19.1.0.pom
support-v4-19.1.0-sources.jar.md5 support-v4-19.1.0.pom.md5
support-v4-19.1.0-sources.jar.sha1 support-v4-19.1.0.pom.sha1

```

Looks like this library is pure Java classes, so we only need to include the [support-v4-19.1.0.jar](#) file into our project. Checking the [.pom](#) file we see that there are no additional dependencies we need to add.

So, to summarize our findings, we need to include the following files in our [native/android](#) directory:

1. appcompat-v7-19.1.0.aar
2. support-v4-19.1.0.jar

And since our MobiHelp SDK lists the appcompat_v7 dependency path as "../appcompat_v7" in its project.properties file, we are going to rename [appcompat-v7-19.1.0.aar](#) to [appcompat_v7.aar](#).

When all is said and done, our [native/android](#) directory should contain the following:

```

appcompat_v7.aar    mobihelp.andlib
com                 support-v4-19.1.0.jar

```

16.13.7. Step 7 : Injecting Android Manifest and Proguard Config

The final step on the Android side is to inject necessary permissions and services into the project's `AndroidManifest.xml` file.

We can find the manifest file injections required by opening the [AndroidManifest.xml](#) file from the MobiHelp SDK project. Its contents are as follows:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.freshdesk.mobihelp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="10" />

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

    <application>
        <activity>

```

```
    android:name="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
    android:configChanges="orientation|screenSize"
    android:theme="@style/Theme.Mobihelp"
    android:windowSoftInputMode="adjustPan" >
</activity>
<activity
    android:name="com.freshdesk.mobihelp.activity.FeedbackActivity"
    android:configChanges="keyboardHidden|orientation|screenSize"
    android:theme="@style/Theme.Mobihelp"
    android:windowSoftInputMode="adjustResize|stateVisible" >
</activity>
<activity
    android:name="com.freshdesk.mobihelp.activity.InterstitialActivity"
    android:configChanges="orientation|screenSize"
    android:theme="@style/Theme.AppCompat">
</activity>
<activity
    android:name="com.freshdesk.mobihelp.activity.TicketListActivity"
    android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
    android:theme="@style/Theme.Mobihelp" >

    <!-- Parent activity meta-data to support 4.0 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity" />
</activity>
<activity
    android:name="com.freshdesk.mobihelp.activity.SolutionArticleActivity"
    android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
    android:theme="@style/Theme.Mobihelp"
    android:configChanges="orientation|screenSize|keyboard|keyboardHidden" >

    <!-- Parent activity meta-data to support 4.0 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity" />
</activity>
<activity
    android:name="com.freshdesk.mobihelp.activity.ConversationActivity"
    android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
    android:theme="@style/Theme.Mobihelp"
    android:windowSoftInputMode="adjustResize|stateHidden" >

    <!-- Parent activity meta-data to support 4.0 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity" />
</activity>
<activity
    android:name="com.freshdesk.mobihelp.activity.AttachmentHandlerActivity"
    android:configChanges="keyboardHidden|orientation|screenSize"
    android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
    android:theme="@style/Theme.Mobihelp" >

    <!-- Parent activity meta-data to support 4.0 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
```

```

        android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity" />
    </activity>

    <service android:name="com.freshdesk.mobihelp.service.MobihelpService" />

    <receiver android:name="com.freshdesk.mobihelp.receiver.ConnectivityReceiver" >
        <intent-filter>
            <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
        </intent-filter>
    </receiver>
</application>

</manifest>

```

We'll need to add the `<uses-permission>` tags and all of the contents of the `<application>` tag to our manifest file. Codename One provides the following build hints for these:

1. `android.xpermissions` - For your `<uses-permission>` directives. Add a build hint with name `android.xpermissions`, and for the value, paste the actual `<uses-permission>` XML tag.
2. `android.xapplication` - For the contents of your `<application>` tag.

Proguard Config

For the release build, we'll also need to inject some proguard configuration so that important classes don't get stripped out at build time. The FreshDesk SDK instructions state:

If you use Proguard, please make sure you have the following included in your project's proguard-project.txt

```

-keep class android.support.v4.** { *; }
-keep class android.support.v7.** { *; }

```

In addition, if you look at the `proguard-project.txt` file inside the Mobihelp SDK, you'll see the rules:

```

-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.app.Activity
-keep public class * extends android.preference.Preference
-keep public class com.freshdesk.mobihelp.exception.MobihelpComponentNotFoundException

-keepclassmembers class * implements android.os.Parcelable {
    public static final android.os.Parcelable$Creator *;
}

```

We'll want to merge this and then paste them into the build hint `android.proguardKeep` of our project.

Troubleshooting Android Stuff

If, after doing all this, your project fails to build, you can enable the "Include Source" option of the build server, then download the source project, open it in Eclipse or Android Studio, and debug from there.

16.14. Part 2: Implementing the iOS Native Code

Part 1 of this tutorial focused on the Android native integration. Now we'll shift our focus to the iOS implementation.

After selecting "Generate Native Interfaces" for our "MobihelpNative" class, you'll find a `native/ios` directory in your project with the following files:

1. `com_codename1_freshdesk_MobihelpNativeImpl.h` [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/ios/com_codename1_freshdesk_MobihelpNativeImpl.h]
2. `com_codename1_freshdesk_MobihelpNativeImpl.m` [https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/ios/com_codename1_freshdesk_MobihelpNativeImpl.m]

These files contain stub implementations corresponding to our `MobihelpNative` class.

We make use of the [API docs](#) [<http://developer.freshdesk.com/mobihelp/ios/api/>] to see how the native SDK needs to be wrapped. The method names aren't the same. E.g. instead of a method `showFeedback()`, it has a message `-presentFeedback:`

We more-or-less just follow the [iOS integration guide](#) [http://developer.freshdesk.com/mobihelp/ios/integration_guide/#getting-started] for wrapping the API. Some key points include:

1. Remember to import the `Mobihelp.h` file in your header file:

```
#import "Mobihelp.h"
```

2. Similar to our use of `runOnUiThread()` in Android, we will wrap all of our API calls in either `dispatch_async()` or `dispatch_sync()` calls to ensure that we are interacting with the Mobihelp API on the app's main thread rather than the Codename One EDT.
3. Some methods/messages in the Mobihelp SDK require us to pass a `UIViewController` as a parameter. In Codename One, the entire application uses a single UIViewController: `CodenameOne_GLViewController`. You can obtain a reference to this using the `[CodenameOne_GLViewController instance]` message. We need to import its header file:

```
#import "CodenameOne_GLViewController.h"
```

As an example, let's look at the `showFeedback()` method:

```

-(void)showFeedback{
    dispatch_async(dispatch_get_main_queue(), ^{
        [[Mobihelp sharedInstance] presentFeedback:[CodenameOne_GLViewController instance]];
    });
}

```

16.14.1. Using the MobihelpNativeCallback

We described earlier how we created a static method on the `MobihelpNativeCallback` class so that native code could easily fire a callback method. Now let's take a look at how this looks from the iOS side of the fence. Here is the implementation of `getUnreadCountAsync()`:

```

-(void)getUnreadCountAsync:(int)param{
    dispatch_async(dispatch_get_main_queue(), ^{
        [[Mobihelp sharedInstance]
            unreadCountWithCompletion:^(NSInteger count){
                com_codename1_freshdesk_MobihelpNativeCallback_fireUnreadUpdatesCallback___int_int_int(
                    CN1_THREAD_GET_STATE_PASS_ARG param, 3 /*SUCCESS*/, count);
            }];
    });
}

```

In our case the iOS SDK version of this method is `+unreadCountWithCompletion:` which takes a block (which is like an anonymous function) as a parameter.

The callback to our Codename One function occurs on this line:

```

com_codename1_freshdesk_MobihelpNativeCallback_fireUnreadUpdatesCallback___int_int_int(
    CN1_THREAD_GET_STATE_PASS_ARG param, 3 /*SUCCESS*/, count);

```

Some things worth mentioning here:

1. The method name is the result of taking the FQN (`MobihelpNativeCallback.fireUpdateUnreadUpdatesCallback(int, int, int)`) in the package `com.codename1.freshdesk`) and replacing all `.` characters with underscores, suffixing two underscores after the end of the method name, then appending `_int` once for each of the `int` arguments.
2. We also need to import the header file for this class:

```
#import "com_codename1_freshdesk_MobihelpNativeCallback.h"
```

16.14.2. Bundling Native iOS SDK

Now that we have implemented our iOS native interface, we need to bundle the Mobihelp iOS SDK

into our project. There are a few different scenarios you may face when looking to include a native SDK:

1. The SDK includes `.bundle` resource files. In this case, just copy the `.bundle` file(s) into your `native/ios` directory.
2. The SDK includes `.h` header files. In this case, just copy the `.h` file(s) into your `native/ios` directory.
3. The SDK includes `.a` files. In this case, just copy the `.a` file(s) into your `native/ios` directory.
4. The SDK includes `.framework` files. In this case, you'll need to zip up the framework, and copy it into your `native/ios` directory. E.g. If the framework is named, `MyFramework.framework`, then the zip file should be named `MyFramework.framework.zip`, and should be located at `native/ios/MyFramework.framework.zip`.

The FreshDesk SDK doesn't include any `.framework` files, so we don't need to worry about that last scenario. We simply [download the iOS SDK](https://s3.amazonaws.com/assets.mobihelp.freshpo.com/sdk/mobihelp_sdk_ios.zip) [https://s3.amazonaws.com/assets.mobihelp.freshpo.com/sdk/mobihelp_sdk_ios.zip], copy the `libFDMobihelpSDK.a`, `Mobihelp.h`, `MHModel.bundle`, `MHResources.bundle`, and `MHLocalization/en.proj/MHLocalizable.strings` into `native/ios`.

16.14.3. Troubleshooting iOS

If you run into problems with the build, you can select "Include Sources" in the build server to download the resulting Xcode Project. You can then debug the Xcode project locally, make changes to your iOS native implementation files, and copy them back into your project once it is building properly.

16.14.4. Adding Required Core Libraries and Frameworks

The iOS integration guide for the FreshDesk SDK lists the following core frameworks as dependencies:

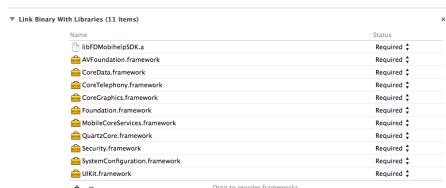


Figure 486. iOS link options

We can add these dependencies to our project using the `ios.add_libs` build hint. E.g.

```
|ios.add_libs
```

```
CoreData.framework;CoreTelephony.framework
```

Figure 487. iOS's "add libs" build hint

I.e. we just list the framework names separated by semicolons. Notice that my list in the above image doesn't include all of the frameworks that they list because many of the frameworks are already included by default (I obtained the default list by simply building the project with "include sources" checked, then looked at the frameworks that were included).

16.15. Part 3 : Packaging as a cn1lib

During the initial development, I generally find it easier to use a regular Codename One project so that I can run and test as I go. But once it is stabilized, and I want to distribute the library to other developers, I will transfer it over to a Codename One library project. This general process involves:

1. Create a Codename One Library project.
2. Copy the .java files from my original project into the library project.
3. Copy the `native` directory from the original project into the library project.
4. Copy the **relevant** build hints from the original project's `codenameone_settings.properties` file into the library project's `codenameone_library_appended.properties` file.

In the case of the FreshDesk .cn1lib, I modified the original project's build script to generate and build a library project automatically. But that is beyond the scope of this tutorial.

16.16. Building Your Own Layout Manager

A `Layout` [<https://www.codenameone.com/javadoc/com/codename1/ui/layouts/Layout.html>] contains all the logic for positioning Codename One components. It essentially traverses a Codename One `Container` [<https://www.codenameone.com/javadoc/com/codename1/ui/Container.html>] and positions components absolutely based on internal logic.

When we build the layout we need to take margin into consideration and make sure to add it into the position/size calculations. Building a layout manager involves two simple methods: `layoutContainer` & `getPreferredSize`.

`layoutContainer` is invoked whenever Codename One decides the container needs rearranging, Codename One tries to avoid calling this method and only invokes it at the last possible moment. Since this method is generally very expensive (imagine the recursion with nested layouts). Codename One just marks a flag indicating layout is "dirty" when something important changes and tries to avoid "reflows".

`getPreferredSize` allows the layout to determine the size desired for the container. This might be a difficult call to make for some layout managers that try to provide both flexibility and simplicity.

Most of `FlowLayout` bugs stem from the fact that this method is just impossible to implement correctly & efficiently for all the use cases of a deeply nested `FlowLayout`. The size of the final layout won't necessarily match the requested size (it probably won't) but the requested size is taken into consideration, especially when scrolling and also when sizing parent containers.

This is a layout manager that just arranges components in a center column aligned to the middle. We then show the proper usage of margin to create a stair like effect with this layout manager:

```

class CenterLayout extends Layout {
    public void layoutContainer(Container parent) {
        int components = parent.getComponentCount();
        Style parentStyle = parent.getStyle();
        int centerPos = parent.getLayoutWidth() / 2 + parentStyle.getMargin(Component.LEFT);
        int y = parentStyle.getMargin(Component.TOP);
        boolean rtl = parent.isRTL();
        for (int iter = 0; iter < components; iter++) {
            Component current = parent.getComponentAt(iter);
            Dimension d = current.getPreferredSize();
            Style currentStyle = current.getStyle();
            int marginRight = currentStyle.getMarginRight(rtl);
            int marginLeft = currentStyle.getMarginLeft(rtl);
            int marginTop = currentStyle.getMarginTop();
            int marginBottom = currentStyle.getMarginBottom();
            current.setSize(d);
            int actualWidth = d.getWidth() + marginLeft + marginRight;
            current.setX(centerPos - actualWidth / 2 + marginLeft);
            y += marginTop;
            current.setY(y);
            y += d.getHeight() + marginBottom;
        }
    }

    public Dimension getPreferredSize(Container parent) {
        int components = parent.getComponentCount();
        Style parentStyle = parent.getStyle();
        int height = parentStyle.getMargin(Component.TOP) + parentStyle.getMargin(Component.BOTTOM);
        int marginX = parentStyle.getMargin(Component.RIGHT) + parentStyle.getMargin(Component.LEFT);
        int width = marginX;
        for (int iter = 0; iter < components; iter++) {
            Component current = parent.getComponentAt(iter);
            Dimension d = current.getPreferredSize();
            Style currentStyle = current.getStyle();
            width = Math.max(d.getWidth() + marginX + currentStyle.getMargin(Component.RIGHT)
                + currentStyle.getMargin(Component.LEFT), width);
            height += currentStyle.getMargin(Component.TOP) + d.getHeight()
                + currentStyle.getMargin(Component.BOTTOM);
        }
        Dimension size = new Dimension(width, height);
        return size;
    }
}

Form hi = new Form("Center Layout", new CenterLayout());
for(int iter = 1 ; iter < 10 ; iter++) {
    Label l = new Label("Label: " + iter);
    l.getUnselectedStyle().setMarginLeft(iter * 3);
    l.getUnselectedStyle().setMarginRight(0);
    hi.add(l);
}
hi.add(new Label("Really Wide Label Text!!!"));
hi.show();

```

Center Layout

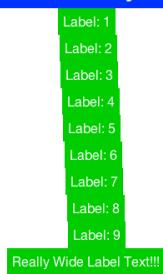


Figure 488. Center layout staircase effect with margin

16.16.1. Porting a Swing/AWT Layout Manager

The [GridLayout](https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html) [https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html] was ported to Codename One relatively easily considering the complexity of that specific layout manager. Here are some tips you should take into account when porting a Swing/AWT layout manager:

1. Codename One doesn't have [Insets](#), we added some support for them in order to port GridBag but components in Codename One have a margin they need to consider instead of the [Insets](#) (the padding is in the preferred size and is thus hidden from the layout manager).
2. AWT layout managers also synchronize a lot on the AWT thread. This is no longer necessary since Codename One is single threaded, like Swing.
3. AWT considers the top left position of the [Container](#) to be 0,0 whereas Codename One considers the position based on its parent [Container](#). The top left position in Codename One is [getX\(\)](#), [getY\(\)](#).

Other than those things it's mostly just fixing method and import statements, which are slightly different. Pretty trivial stuff.

16.17. Port a Language to Codename One

As you may have already read, we have just added support for Kotlin in Codename One. This is something that you can achieve without the help of Codename One. You could port a 3rd party language like Scala, Ruby, Python etc. to Codename One.

16.17.1. What is a JVM Language?

A JVM Language is any programming language that can be compiled to byte-codes that will run on the JVM (Java Virtual Machine). Java was the original JVM language, but many others have sprung up over the years. [Kotlin](https://kotlinlang.org/) [https://kotlinlang.org/], [Scala](https://www.scala-lang.org/) [https://www.scala-lang.org/], [Groovy](http://groovy-lang.org/) [http://groovy-lang.org/], and [JRuby](http://jruby.org/) [http://jruby.org/] come to mind as well-established and mature languages, but there are [many others](https://en.wikipedia.org/wiki/List_of_JVM_languages) [https://en.wikipedia.org/wiki/List_of_JVM_languages].

16.17.2. How Hard is it to Port a JVM Language to Codename One?

The difficulty of porting a particular language to Codename One will vary depending on such factors as:

1. Does it require a runtime library?
 - a. How complex is the runtime library? (E.g. Does it require classes that aren't currently offered in Codename One's subset of the java standard libraries?)
2. Does it need reflection?
 - a. Codename One doesn't support reflection because it would result in a very large application size. If a JVM language requires reflection just to get off the ground then adding it to Codename one would be tricky.
3. Does it perform any runtime byte-code manipulation?
 - a. Some dynamic languages may perform byte-code manipulation at runtime. This is problematic on iOS (and possibly other platforms) which prohibits such runtime behavior.

Step 1: Assess the Language

The more similar a language, and its build outputs are to Java, the easier it will be to port (probably). Most JVM languages have two parts:

1. A compiler, which compiles source files to JVM byte-code (usually as .class files).
2. A runtime library.

Currently I'm only aware of one language (other than Java) that doesn't require a runtime library, and that is [Mirah](http://www.mirah.org/) [<https://www.codenameone.com/blog/mirah-for-codename-one.html>].



Codename One also supports [Mirah](https://www.codenameone.com/blog/mirah-for-codename-one.html) [<https://www.codenameone.com/blog/mirah-for-codename-one.html>]

Assessing the Byte-Code

The first thing I do is take a look at the byte-code that is produced by the compiler. I use `javap` to print out a nice version.

Consider this sample Kotlin class:

```

package com.codename1.hellokotlin2

import com.codename1.ui.Button
import com.codename1.ui.Form
import com.codename1.ui.Label
import com.codename1.ui.layouts.BoxLayout

/**
 * Created by shannah on 2017-07-10.
 */
class KotlinForm : Form {

    constructor() : super("Hello Kotlin", BoxLayout.y()) {
        val label = Label("Hello Kotlin")
        val clickMe = Button("Click Me")
        clickMe.addActionListener {
            label.setText("You Clicked Me");
            revalidate();
        }
        add(label).add(clickMe);
    }

}

```

}

Let's take a look at the bytecode that Kotlin produced for this class:

```
$ javap -v com/codename1/hellokotlin2/KotlinForm.class
```

```

Last modified 10-Jul-2017; size 1456 bytes
MD5 checksum 1cb00f6e63b918bb5a9f146ca8b0b78e
Compiled from "KotlinForm.kt"
public final class com.codename1.hellokotlin2.KotlinForm extends com.codename1.ui.Form
SourceFile: "KotlinForm.kt"
InnerClasses:
    static final #31; //class com/codename1/hellokotlin2/KotlinForm$1
RuntimeVisibleAnnotations:
    0: #56(#57=[I#58,I#58,I#59],#60=[I#58,I#61,I#58],#62=I#58,#63=[s#64],#65=[s#55,s#66,s#6,s#67])
minor version: 0
major version: 50
flags: ACC_PUBLIC, ACC_FINAL, ACC_SUPER
Constant pool:
#1 = Utf8                  com/codename1/hellokotlin2/KotlinForm
#2 = Class                 #1          //  com/codename1/hellokotlin2/KotlinForm
#3 = Utf8                  com/codename1/ui/Form
#4 = Class                 #3          //  com/codename1/ui/Form
#5 = Utf8                  <init>
#6 = Utf8                  ()V
#7 = Utf8                  Hello Kotlin
#8 = String                #7          //  Hello Kotlin
#9 = Utf8                  com/codename1/ui/layouts/BoxLayout

```

```

#10 = Class          #9           // com/codename1/ui/layouts/BoxLayout
#11 = Utf8          y
#12 = Utf8          ()Lcom/codename1/ui/layouts/BoxLayout;
#13 = NameAndType   #11:#12      // y:()Lcom/codename1/ui/layouts/BoxLayout;
#14 = Methodref     #10.#13      // com/codename1/ui/layouts/BoxLayout.y:()Lcom/codename1/ui/layouts/BoxLayout;
#15 = Utf8          com/codename1/ui/layouts/Layout
#16 = Class          #15          // com/codename1/ui/layouts/Layout
#17 = Utf8          (Ljava/lang/String;Lcom/codename1/ui/layouts/Layout;)V
#18 = NameAndType   #5:#17      // "<init>":(Ljava/lang/String;Lcom/codename1/ui/layouts/Layout;)V
#19 = Methodref     #4.#18      //
com/codename1/ui/Form."<init>":(Ljava/lang/String;Lcom/codename1/ui/layouts/Layout;)V
#20 = Utf8          com/codename1/ui/Label
#21 = Class          #20          // com/codename1/ui/Label
#22 = Utf8          (Ljava/lang/String;)V
#23 = NameAndType   #5:#22      // "<init>":(Ljava/lang/String;)V
#24 = Methodref     #21.#23      // com/codename1/ui/Label."<init>":(Ljava/lang/String;)V
#25 = Utf8          com/codename1/ui/Button
#26 = Class          #25          // com/codename1/ui/Button
#27 = Utf8          Click Me
#28 = String         #27          // Click Me
#29 = Methodref     #26.#23      // com/codename1/ui/Button."<init>":(Ljava/lang/String;)V
#30 = Utf8          com/codename1/hellokotlin2/KotlinForm$1
#31 = Class          #30          // com/codename1/hellokotlin2/KotlinForm$1
#32 = Utf8          (Lcom/codename1/hellokotlin2/KotlinForm;Lcom/codename1/ui/Label;)V
#33 = NameAndType   #5:#32      // "<init>":(Lcom/codename1/hellokotlin2/KotlinForm;Lcom/codename1/ui/Label;)V
#34 = Methodref     #31.#33      //
com/codename1/hellokotlin2/KotlinForm$1."<init>":(Lcom/codename1/hellokotlin2/KotlinForm;Lcom/codename1/ui/Label;)V
#35 = Utf8          com/codename1/ui/events/ActionListener
#36 = Class          #35          // com/codename1/ui/events/ActionListener
#37 = Utf8          addActionListener
#38 = Utf8          (Lcom/codename1/ui/events/ActionListener;)V
#39 = NameAndType   #37:#38      // addActionListener:(Lcom/codename1/ui/events/ActionListener;)V
#40 = Methodref     #26.#39      //
com/codename1/ui/Button.addActionListener:(Lcom/codename1/ui/events/ActionListener;)V
#41 = Utf8          com/codename1/ui/Component
#42 = Class          #41          // com/codename1/ui/Component
#43 = Utf8          add
#44 = Utf8          (Lcom/codename1/ui/Component;)Lcom/codename1/ui/Container;
#45 = NameAndType   #43:#44      // add:(Lcom/codename1/ui/Component;)Lcom/codename1/ui/Container;
#46 = Methodref     #2.#45      //
com/codename1/hellokotlin2/KotlinForm.add:(Lcom/codename1/ui/Component;)Lcom/codename1/ui/Container;
#47 = Utf8          com/codename1/ui/Container
#48 = Class          #47          // com/codename1/ui/Container
#49 = Methodref     #48.#45      //
com/codename1/ui/Container.add:(Lcom/codename1/ui/Component;)Lcom/codename1/ui/Container;
#50 = Utf8          clickMe
#51 = Utf8          Lcom/codename1/ui/Button;
#52 = Utf8          label
#53 = Utf8          Lcom/codename1/ui/Label;
#54 = Utf8          this
#55 = Utf8          Lcom/codename1/hellokotlin2/KotlinForm;
#56 = Utf8          Lkotlin/Metadata;
#57 = Utf8          mv
#58 = Integer        1
#59 = Integer        6
#60 = Utf8          bv
#61 = Integer        0
#62 = Utf8          k
#63 = Utf8          d1
#64 = Utf8          \n\n\204"
#65 = Utf8          d2
#66 = Utf8          Lcom/codename1/ui/Form;
#67 = Utf8          HelloKotlin2
#68 = Utf8          KotlinForm.kt
#69 = Utf8          Code
#70 = Utf8          LocalVariableTable

```

```

#71 = Utf8           LineNumberTable
#72 = Utf8           SourceFile
#73 = Utf8           InnerClasses
#74 = Utf8           RuntimeVisibleAnnotations
{
    public com.codename1.hellokotlin2.KotlinForm();
        descriptor: ()V
        flags: ACC_PUBLIC
        Code:
            stack=5, locals=3, args_size=1
            0: aload_0
            1: ldc      #8           // String Hello Kotlin
            3: invokestatic #14        // Method
com/codename1/ui/layouts/BoxLayout.y:()Lcom/codename1/ui/layouts/BoxLayout;
            6: checkcast   #16        // class com/codename1/ui/layouts/Layout
            9: invokespecial #19       // Method
com/codename1/ui/Form."<init>":(Ljava/lang/String;Lcom/codename1/ui/layouts/Layout;)V
            12: new     #21          // class com/codename1/ui/Label
            15: dup
            16: ldc      #8           // String Hello Kotlin
            18: invokespecial #24       // Method com/codename1/ui/Label."<init>":(Ljava/lang/String;)V
            21: astore_1
            22: new     #26          // class com/codename1/ui/Button
            25: dup
            26: ldc      #28          // String Click Me
            28: invokespecial #29       // Method com/codename1/ui/Button."<init>":(Ljava/lang/String;)V
            31: astore_2
            32: aload_2
            33: new     #31          // class com/codename1/hellokotlin2/KotlinForm$1
            36: dup
            37: aload_0
            38: aload_1
            39: invokespecial #34       // Method
com/codename1/hellokotlin2/KotlinForm$1."<init>":(Lcom/codename1/hellokotlin2/KotlinForm;Lcom/codename1/ui/Label;)V
            42: checkcast   #36        // class com/codename1/ui/events/ActionListener
            45: invokevirtual #40        // Method
com/codename1/ui/Button.addActionListener:(Lcom/codename1/ui/events/ActionListener;)V
            48: aload_0
            49: aload_1
            50: checkcast   #42        // class com/codename1/ui/Component
            53: invokevirtual #46        // Method add:(Lcom/codename1/ui/Component;)Lcom/codename1/ui/Container;
            56: aload_2
            57: checkcast   #42        // class com/codename1/ui/Component
            60: invokevirtual #49        // Method
com/codename1/ui/Container.add:(Lcom/codename1/ui/Component;)Lcom/codename1/ui/Container;
            63: pop
            64: return
    LocalVariableTable:
        Start  Length  Slot  Name   Signature
            32      32     2  clickMe  Lcom/codename1/ui/Button;
            22      42     1  label    Lcom/codename1/ui/Label;
            0       65     0  this     Lcom/codename1/hellokotlin2/KotlinForm;
    LineNumberTable:
        line 13: 0
        line 14: 12
        line 15: 22
        line 16: 32
        line 21: 48
}

```

That's a big mess of stuff, but it's pretty easy to pick through it when you know what you're looking for. The layout of this output is pretty straight forward. The beginning shows that this is a class definition:

```
public final class com.codename1.hellokotlin2.KotlinForm extends com.codename1.ui.Form {  
    //...  
}
```

Even just comparing this line with the class definition from the source file we have learned something about the Kotlin compiler. It has made the class `final` by default. That observation shouldn't affect our assessment here, but it is kind of interesting.

After the class definition, it shows the internal classes:

```
InnerClasses:  
    static final #31; //class com/codename1/hellokotlin2/KotlinForm$1
```

The Constant Pool

And the constants that are used in the class:

Constant pool:

#1 = Utf8	com/codename1/hellokotlin2/KotlinForm
#2 = Class	#1 // com/codename1/hellokotlin2/KotlinForm
#3 = Utf8	com/codename1/ui/Form
#4 = Class	#3 // com/codename1/ui/Form
#5 = Utf8	<init>
#6 = Utf8	()V
#7 = Utf8	Hello Kotlin
#8 = String	#7 // Hello Kotlin
#9 = Utf8	com/codename1/ui/layouts/BoxLayout
... etc...	

The constant pool will consist of class names, and strings mostly. You'll want to peruse this list to see if the compiler has added any classes that aren't in the source code. In the example above, it looks like Kotlin is pretty faithful to the original source's dependencies. It didn't inject any classes that aren't in the original source.

Even if the compiler does inject other dependencies into the bytecode, it might not be a problem. It is only a problem if those classes aren't supported by Codename One. Keep your eyes peeled for anything in the `java.lang.reflect` package or unsolicited use of `java.net`, `java.nio`, or any other package that aren't part of the Codename One standard library. If you're not sure if a class or package is available in the Codename One standard library, check [the javadocs](https://www.codenameone.com/javadoc/) [<https://www.codenameone.com/javadoc/>].

The ByteCode Instructions:

After the constant pool, we see each of the methods of the class written out as a list of bytecode instructions. E.g.

```

public com.codename1.hellokotlin2.KotlinForm();
descriptor: ()V
flags: ACC_PUBLIC
Code:
stack=5, locals=3, args_size=1
  0: aload_0
  1: ldc           #8           // String Hello Kotlin
  3: invokestatic #14          // Method
com/codename1/ui/layouts/BoxLayout.y:()Lcom/codename1/ui/layouts/BoxLayout;
  6: checkcast    #16          // class com/codename1/ui/layouts/Layout
  9: invokespecial #19         // Method
com/codename1/ui/Form.<init>:(Ljava/lang/String;Lcom/codename1/ui/layouts/Layout;)V
 12: new           #21          // class com/codename1/ui/Label
 15: dup
 16: ldc           #8           // String Hello Kotlin
etc...

```

In the above snippet, the first instruction is `aload_0` (which adds `this` to the stack). The 2nd instruction is `ldc`, (which loads constant #8—the string "Hello Kotlin" to the stack). The 3rd instruction is `invokestatic` which calls the static method define by Constant #14 from the constant pool, with the two parameters that had just been added to the stack.



You don't need to understand what all of these instructions do. You just need to look for instructions that may be problematic.

The only instruction that I **think** might be problematic is "invokedynamic". All other instructions should work fine in Codename One. (I don't know for a fact that invokedynamic won't work - I just suspect it might not work on some platforms).

Summary of Byte-code Assessment

So to summarize, the byte-code assessment phase, we're basically just looking to make sure that the compiler doesn't tend to add dependencies to parts of the JDK that Codename One doesn't currently support. And we want to make sure that it doesn't use invokedynamic.

If you find that the compiler does use invokedynamic or add references to classes that Codename One doesn't support, don't give up just yet. You might be able to create your own "porting" runtime library that will provide these dependencies at runtime.

Assessing the Runtime Library

The process for assessing the runtime library is pretty similar to the process for the bytecodes. You'll want to get your hands on the language's runtime library, and use `javap` to inspect the .class files. You're looking for the same things as you were looking for in the compiler's output: "invokedynamic" and classes that aren't supported in Codename One.

Step 2: Convert the Runtime Library into a CN1Lib

Once you have assessed the language and are optimistic that it is a good candidate for porting, you can proceed to port the runtime library into Codename One. Usually that language's runtime library will be distributed in .jar format. You need to convert this into a cn1lib so that it can be used in a Codename One project. If you can get your hands on the source code for the runtime library

then the best approach is to paste the source files into a Codename One Library project, and try to build it. This has the advantage that it will validate the source during compile to ensure that it doesn't depend on any classes that Codename One doesn't support.

If you can't find the sources of the runtime library or they don't seem to be easily "buildable", then the next best thing is to just get the binary distribution's jar file and convert it to a cn1lib. This is what we did for the [Kotlin runtime library](https://github.com/shannah/codenameone-kotlin) [<https://github.com/shannah/codenameone-kotlin>].

This procedure exploits the fact that a cn1lib file is just a zip file with a specific file structure inside it. The cross-platform Java .class files are all contained inside a file named "main.zip", inside the zip file. This is the only **mandatory** file that must be inside a cn1lib.

To make the library easier to use the cn1lib file can also contain a file named "stubs.zip" which includes stubs of the Java sources. When you build a cn1lib using a Codename One Library project, it will automatically generate stubs of the source so that the IDE will have access to nice things like Javadoc when using the library. The kotlin distribution includes a separate jar file with the runtime sources, named "kotlin-runtime-sources.jar", so we used this as the "stubs". It contains full sources, which isn't necessary, but it also doesn't hurt.

So now that we had my two jar files: kotlin-runtime.jar and kotlin-runtime-sources.jar, I created a new empty directory, and copied them inside. I renamed the jars "main.zip" and "stubs.zip" respectively. Then I zipped up the directory and renamed the zip file "kotlin-runtime.cn1lib".

Building cn1libs manually in this way is a **very** bad habit, as it bypasses the API verification step that normally occurs when building a library project. It is possible, even likely, that the jar files that you convert depend on classes that aren't in the Codename One library, so your library will fail at runtime in unexpected ways. The only reason we could do this with kotlin's runtime (with some confidence) is because I already analyzed the bytecodes to ensure that they didn't include anything problematic.



Step 3: Hello World

For our "Hello World" test we will need to create a separate project in our JVM language and produce class files that we will **manually** copy into an appropriate location of our project. We'll want to use the **normal** tools for the language and not worry about how it integrates with Codename One. For Kotlin, I just followed the getting started tutorial on the Kotlin site to create a new Kotlin project in IntelliJ. When Steve ported Mirah, he just used a text editor and the mirahc command-line compiler to create my Hello World class. The tools and process will depend on the language.

Here is the "hello world" we created in Kotlin:

```
package com.mycompany.myapp

class HelloKotlin {

    fun hello() {
        System.out.println("Hello from Kotlin");
    }
}
```

After building this, I have a directory that contains "com/mycompany/myapp/HelloKotlin.class".

It also produced a .jar file that contains this class.

The easiest way to integrate external code into a Codename One project, is just to wrap it as a cn1lib file and place it into my Codename One project's lib directory. That way you don't have to mess with any of the build files. So, using roughly the same procedure as we used to create the kotlin-runtime.cn1lib, I wrap my hellokotlin.jar as a cn1lib to produce "hellokotlin.cn1lib" and copy it to the "lib" directory of a Codename One project.



Remember to select "Codename One" → "Refresh CN1Libs" after placing the cn1lib in your lib directory or it won't get picked up.

Finally, I call my library from the start() method of my app:

```
HelloKotlin hello = new HelloKotlin();
hello.hello();
```

If we run this in the Simulator, it should print "Hello from Kotlin" in the output console. If we get an error, then we can dig in and try to figure out what went wrong using my standard debugging techniques. **EXPECT** an error on the first run. Hopefully it will just be a missing import or something simple.

Step 4: A More Complex Hello World

In the case of Kotlin, the hello world example app would actually run without the runtime library because it was so simple. So it was necessary to add a more complex example to prove the need for the runtime library. It doesn't matter what you do with your more complex example, as long as it doesn't require classes that aren't in Codename One.

If you want to use the Codename One inside your project, you should add the CodenameOne.jar (found inside any Codename One project) to your classpath so that it will compile.

Step 5: Automation and Integration

At this point we already have a manual process for incorporating files built with our alternate language into a Codename One project. The process looks like:

1. Use standard tools for your JVM language to write your code.

2. Use the JVM language's standard build tools (e.g. command-line compiler, etc..) to compile your code so that you have .class files (and optionally a .jar file).
3. Wrap your .class files in a cn1lib.
4. Add the cn1lib to the lib directory of a Codename One project.
5. Use your library from the Codename One project.

When Steve first developed Mirah support he automated this process using an [ANT script](#) [<https://github.com/shannah/CN1MirahNBM/blob/master/src/ca/weblite/codename1/mirah/build.xml>]. He also automatically generated some bootstrap code so that he could develop the whole app in Mirah and he wouldn't have to write any Java. However, this level of integration has limitations.

For example, with this approach alone, you couldn't have two-way dependencies between Java source and Mirah source. Yes, Mirah code could use Java libraries (and it did depend on CodenameOne.jar), and my Java code could use my Mirah code. However, Mirah **source** code could not depend on the Java **source** code in my project. This has to do with the order in which code is compiled. It's a bit of a chicken and egg issue. If we are building a project that has Java source code and Mirah source code, we are using two different compilers: mirahc to compile the Mirah files, and javac to compile the Java files. If we are starting from a clean build, and we run mirahc first, then the .java files haven't yet been compiled to .class files - and thus mirahc can't **reference** them - and any mirah code that depends on those uncompiled Java classes will fail. If we compile the .java files first, then we have the opposite problem.

Steve worked around this problem in Mirah by writing [my own pseudo-compiler](#) [<https://github.com/shannah/mirah-ant/blob/master/src/ca/weblite/asm/JavaExtendedStubCompiler.java>] that produced stub class files for the java source that would be referenced by mirahc when compiling the Mirah files. In this way he was able to have two-way dependencies between Java and Mirah in the same project.

Kotlin also supports two-way dependencies, probably using a similar mechanism.

How Seamless Can You Make It?

For both the Kotlin and Mirah support, we wanted integration to be seamless. We didn't want users to have to create a separate project for their Kotlin/Mirah code. We wanted them to simply add a Kotlin/Mirah file into their project and have it **just work**. Achieving this level of integration in Kotlin was quite easy, since they provide an [ANT plugin](#) [<https://kotlinlang.org/docs/reference/using-ant.html>] that essentially allowed me to just add one tag inside my `<javac/>` tags:

```
<withKotlin/>
```

And it would automatically handle Kotlin and Java files together: Seamlessly. There are a few places in a Codename One's build.xml file where we call "javac" so we just needed to inject these tags in those places. This injection is performed automatically by the Codename One IntelliJ plugin.

For Mirah, Steve developed his own [ANT plugins](#) [<https://github.com/shannah/mirah-ant>] and [Netbeans module](#) [<https://github.com/shannah/mirah-nbm>] that do something similar in Netbeans.

16.18. Update Framework

When we launched Codename One in 2012 we needed a way to ship updates and fixes faster than the plugin update system. So we built the client lib update system. Then we needed a way to update the designer tool (resource editor), the GUI builder & the skins... We also needed a system to update the builtin builder code (`CodeNameOneBuildClient.jar` so we built a tool for that too).

The [Update Framework](#) [<https://github.com/codenameone/UpdateCodenameOne>] solves several problems in the old systems:

- Download once - if you have multiple projects the library will only download once to the `.codenameone` directory. All the projects will update from local storage
- Skins update automatically - this is hugely important. When we change a theme we need to update it in the skins and if you don't update the skin you might see a difference between the simulator and the device
- Update of settings/designer without IDE plugin update - The IDE plugin update process is slow and tedious. This way we can push out a bug fix for the GUI builder without going through the process of releasing a new plugin version

For the most part this framework should be seamless. You should no longer see the "downloading" message whenever we push an update after your build client is updated. Your system would just poll for a new version daily and update when new updates are available.

You can also use the usual method of **Codename One Settings** → **Basic** → **Update Client Libs** which will force an update check. Notice that the UI will look a bit different after this update.

16.18.1. How does it Work?

You can see the full code [here](#) [<https://github.com/codenameone/UpdateCodenameOne>] the gist of it is very simple. We create a jar called `UpdateCodenameOne.jar` under `~/.codenameone` (~ represents the users home directory).

An update happens by running this tool with a path to a Codename One project e.g.:

```
java -jar ~/.codenameone/UpdateCodenameOne.jar path_to_my_codenameone_project
```

E.g.:

```
java -jar ~/.codenameone/UpdateCodenameOne.jar ~/dev/AccordionDemo
Checking: JavaSE.jar
Checking: CodeNameOneBuildClient.jar
Checking: CLDC11.jar
Checking: CodenameOne.jar
Checking: CodenameOne_SRC.jar
Checking: designer_1.jar
Checking: guibuilder_1.jar
Updating the file: /Users/shai/dev/AccordionDemo/JavaSE.jar
Updating the file: /Users/shai/dev/AccordionDemo/CodeNameOneBuildClient.jar
Updating the file: /Users/shai/dev/AccordionDemo/lib/CLDC11.jar
Updating the file: /Users/shai/dev/AccordionDemo/lib/CodenameOne.jar
Updated project files
```

Notice that no download happened since the files were up to date. You can also force a check against the server by adding the force argument as such:

```
java -jar ~/.codenameone/UpdateCodenameOne.jar path_to_my_codenameone_project
```

The way this works under the hood is thought a `Versions.properties` within your directory that lists the versions of local files. That way we know what should be updated.



Exclude `Versions.properties` from Git

Under the `~/.codenameone` directory we have a more detailed `UpdateStatus.properties` file that includes versions of the locally downloaded files. Notice you can delete this file and it will be recreated as all the jars get downloaded over again.

16.18.2. What isn't Covered

You will notice 3 big things that aren't covered in this unified framework:

- We don't update cn1libs - I'm not sure if this is something we would like to update automatically
- Versioned builds - there is a lot of complexity in the versioned build system. This might be something we address in the future but for now I wanted to keep the framework simple.
- Offline builds - Offline builds work through manual download and aren't subjected to this framework

[10] The old Codename One VM

17. Signing, Certificates & Provisioning

In this section we attempt to explain how to acquire certificates for the various platforms and how to set them up.

The good news is that this is usually a "one time issue" and once it's done the work becomes easier (except for the case of iOS where a provisioning profile should be maintained).

17.1. Common Terms In Signing & Distribution

This section uses some terms that you might not be familiar with if you haven't used cryptography or built mobile apps before. Here is a quick "FAQ" covering some common terms/concepts in this section.

17.1.1. What Is A Certificate?

Certificates use cryptographic principals to "sign" data (e.g. an application). Think of them as you would think of a company stamp, you use them to sign an app so users know who it's from.

17.1.2. What Is Provisioning?

Provisioning provides the hints/guidelines for the application install. E.g. if an application needs some service from the OS such as push it can usually request that with provisioning.

In iOS provisioning is separate from the app and you need to also define the devices supported by the app during development.

17.1.3. What's a Signing Authority?

Normally certificates are issued by a signing authority which is a body that certifies that you are who you say you are. Apple issues certificates for iOS and is in effect a signing authority.

Android uses self signed certificates which don't use a signing authority so anyone can ship an Android app.

The logic with the Android approach is that a signature indicates that you are the same person who previously shipped the app. Hence an app will be updated only with the exact same certificate.

17.1.4. What is UDID?

The UDID is the Universal Device Identifier. It identifies mobile devices uniquely, notice that some operating systems e.g. iOS block access to this value due to privacy concerns.

You need the iOS device UDID value during development to add the device into the list of allowed devices.



Don't use an app to get the UDID!

Most return the wrong value, the official way to get the UDID is thru itunes. We can also recommend trying <http://get.udid.io/> which seems to work rather well

17.1.5. Should I Reuse the Same Certificate for All Apps?

For iOS yes. It's designed to work in that way.

We would recommend it for all platforms for simplicity but some developers prefer creating per-application certificates for Android. The advantage here is that you can transfer ownership of the application later on without giving away what is effectively "you house keys".

17.2. iOS Signing Wizard

Codename One features a wizard to generate certificates/provisioning for iOS without requiring a Mac or deep understanding of the signing process for iOS. There is still support for manually generating the P12/provisioning files when necessary but for most intents and purposes using the wizard will simplify this error prone process significantly.

To generate your certificates and profiles, open project's properties and click on "iOS" in the left menu. This will show the "iOS Signing" panel that includes fields to select your certificates and mobile provisioning profiles.

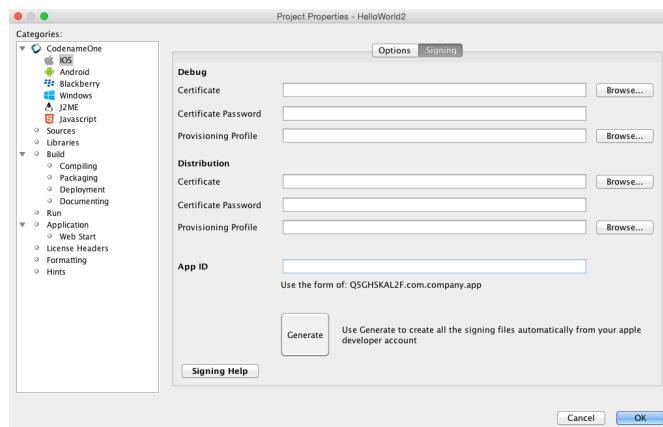


Figure 489. Netbeans iOS Signing properties panel

If you already have valid certificates and profiles, you can just enter their locations here. If you don't, then you can use the wizard by clicking the **Generate** button in the lower part of the form.

17.2.1. Logging into the Wizard

After clicking **Generate** you'll be shown a login form. Log into this form using your **iTunes Connect** user ID and password. **NOT YOUR CODENAME ONE LOGIN**.

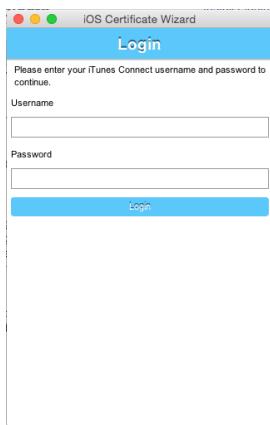


Figure 490. Wizard login form

17.2.2. Selecting Devices

Once you are logged in you will be shown a list of all of the devices that you currently have registered on your Apple developer account.

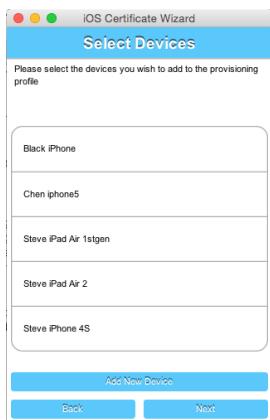


Figure 491. Devices form

Select the ones that you want to include in your provisioning profile and click next.



Apple supports up to 100 devices for testing purposes so if you want to install the built app on your device you need to add it here

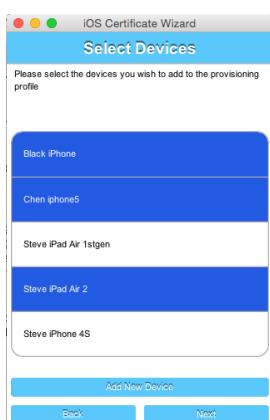


Figure 492. After selecting devices

If you don't have any devices registered yet, you can click the "Add New Device" button, which will prompt you to enter the UDID for your device.

17.2.3. Decisions & Edge Cases



If you already have iOS P12 development/distribution certificates you should reuse them for all your apps from that account and you shouldn't regenerate them

After you click **Next** on the device form, the wizard checks to see if you already have a valid certificate. If your project already has a valid certificate and it matches the one that is currently active in your apple developer account, then it will just use the same certificate. If the certificate doesn't match the currently-active one, or you haven't provided a certificate, you will be prompted to overwrite the old certificate with a new one.

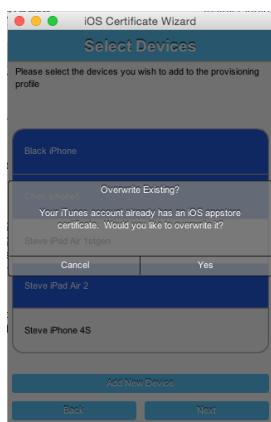


Figure 493. Prompt to overwrite existing certificate



Figure 494. Prompt to overwrite other certificate

The same decision need to be made twice: Once for the development certificate, and once for the Appstore certificate.



You can revoke a certificate when you have an application in the store shipping with said certificate!

This won't affect the shipping app see [this](http://stackoverflow.com/questions/6320255/if-i-revoke-an-existing-distribution-certificate-will-it-mess-up-anything-with) [http://stackoverflow.com/questions/6320255/if-i-revoke-an-existing-distribution-certificate-will-it-mess-up-anything-with].

Why Two Certificates?

A typical iOS app has at least two certificates:

- Development - this is used during development and can't be shipped to 3rd parties. An application signed with this certificate can only be installed on one of the up to 100 devices listed above.
- Distribution - this is used when you are ready to upload your app to iTunes whether for final shipping or beta testing. Notice that you can't install a distribution build on your own device. You need to upload it to iTunes.
- There are two push certificates, they are separate from the signing certificates. Don't confuse them!

They are used to authenticate with Apple when sending push messages.

17.2.4. App IDs and Provisioning Profiles

The next form in the wizard asks for your app's bundle ID. This should have been pre-filled, but you can change the app ID to a wildcard ID if you prefer.

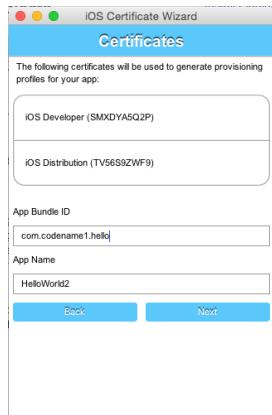


Figure 495. Enter the app bundle ID

Wildcard Card Provisioning

Wildcard ids such as com.mycompany.* or even * allow you to create one generic certificate to use with all applications. This is remarkably useful for the global settings dialog and allows you to create an app without launching the wizard. Notice that wildcards apps can't use features such as push etc.

You can set the global defaults for the IDE by going to IDE settings/preferences and setting default values e.g.:

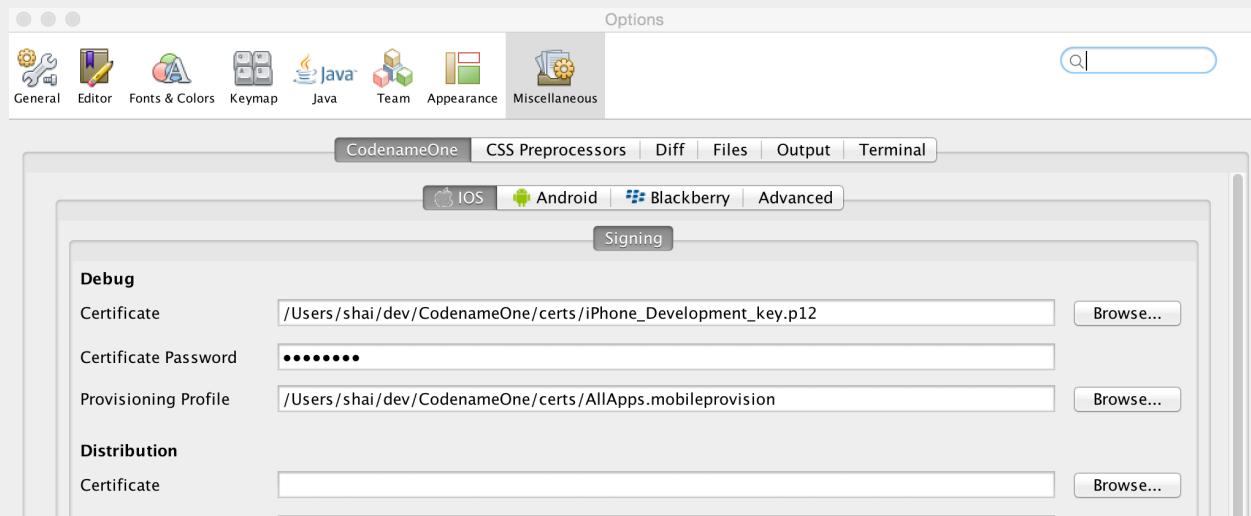


Figure 496. Setting the development certificate and a global * provisioning profile allows you to create a new app and build it to device without running the certificate wizard. Notice that you will need to run it when going into production

17.2.5. Installing Files Locally

Once the wizard is finished generating your provisioning profiles, you should click "Install Locally", which will open a file dialog for you to navigate to a folder in which to store the generated files.

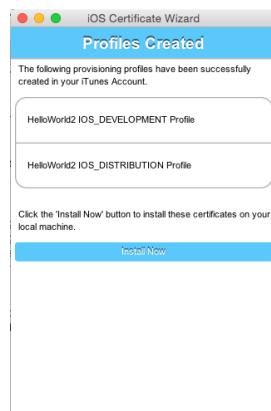


Figure 497. Install files locally

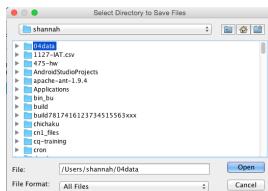


Figure 498. Select directory to save files in



Figure 499. Final Done Message



You can see the password for the P12 files in the `codenameone_settings.properties` file. You can use the values defined there when creating a new application

17.2.6. Building Your App

After selecting your local install location, and closing the wizard, you should see the fields of the "iOS Signing" properties panel filled in correctly. You should now be able to send iOS debug or Appstore builds without the usual hassles.

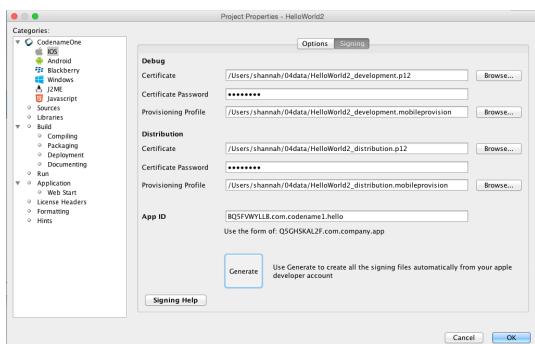


Figure 500. Filled in signing panel after wizard complete

17.3. Advanced iOS Signing



You should use the certificate wizard, especially if you don't have a Mac. This section is here mostly for reference and edge cases that don't work with the certificate wizard

iOS signing has two distinct modes: App Store signing which is only valid for distribution via iTunes (you won't be able to run the resulting application without submitting it to Apple) and development mode signing.

You have two major files to keep track of:

1. **Certificate** - your signature
2. **Provisioning Profile** - details about the application and who is allowed to execute it

You need two versions of each file (4 total files) one pair is for development and the other pair is for uploading to the iTunes App Store.



You need to use a Mac in order to create a certificate file for iOS

The first step you need to accomplish is signing up as a developer to [Apple's iOS development program](http://developer.apple.com/) [<http://developer.apple.com/>], even for testing on a device this is required!

This step requires that you pay Apple on an annual basis.

The Apple website will guide you through the process of applying for a certificate at the end of this process you should have a distribution and development certificate pair. After that point you can login to the [iOS provisioning portal](https://developer.apple.com/ios/manage/overview/index.action) [<https://developer.apple.com/ios/manage/overview/index.action>] where there are plenty of videos and tutorials to guide you through the process. Within the iOS provisioning portal you need to create an application ID and register your development devices.

You can then create a provisioning profile which comes in two flavors:

- Distribution - for building the release version of your application
- Development - the development provisioning profile needs to contain the devices on which you want to test.

You can then configure the 4 files in the IDE and start sending builds to the Codename One cloud.

17.4. Provisioning Profile & Certificates Visual Guide

One of the hardest parts in developing for iOS is the certificate & provisioning process. In this step by step guide we walk over the manual certificate generation process. Notice that the UI for the Apple website changes occasionally but the basic process remains the same...

Start by logging in to the iOS-provisioning portal

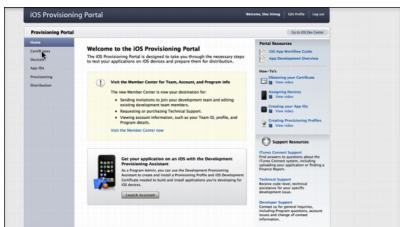


Figure 501. Login for the iOS provisioning portal

In the certificates section you can download your development and distribution certificates.

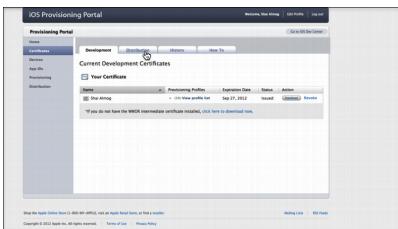


Figure 502. Download development provisioning profile

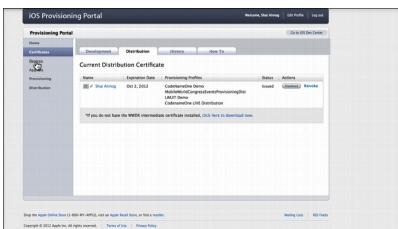


Figure 503. Download distribution provisioning profile

In the devices section add device ids for the development devices you want to support. Notice no more than 100 devices are supported!

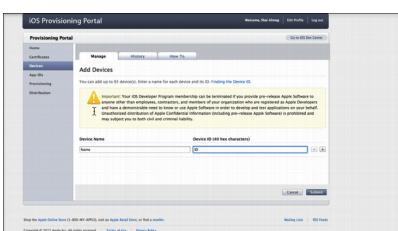


Figure 504. Add devices

Create an application id; it should match the package identifier of your application perfectly!

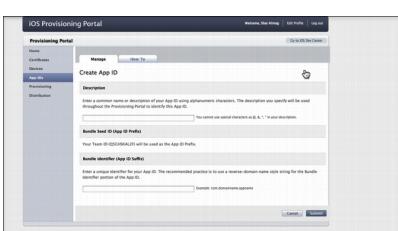


Figure 505. Create application id

Create a provisioning profile for development, make sure to select the right app and make sure to add the devices you want to use during debug.

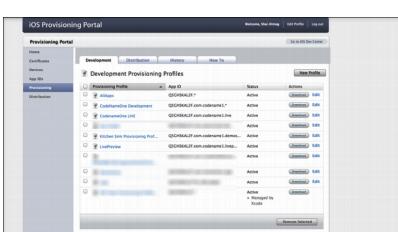


Figure 506. Create provisioning profile step 1

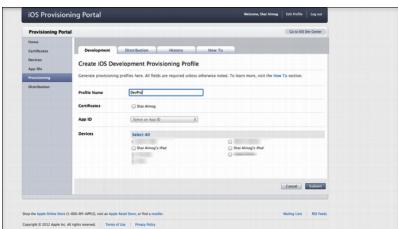


Figure 507. Create provisioning profile step 2

Refresh the screen to see the profile you just created and press the download button to download your development provisioning profile.

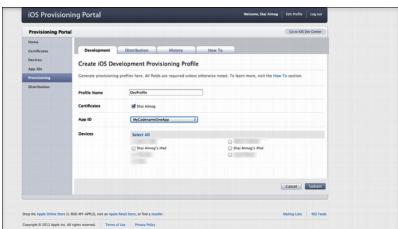


Figure 508. Create provisioning profile step 3

Create a distribution provisioning profile; it will be used when uploading to the app store. There is no need to specify devices here.

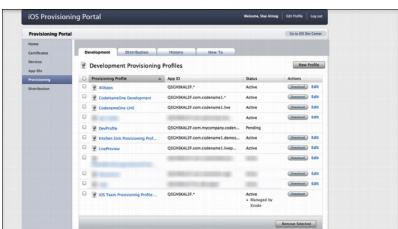


Figure 509. Create distribution provisioning profile

Download the distribution provisioning profile.

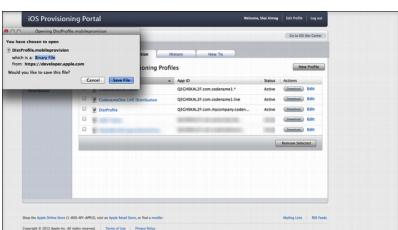


Figure 510. Download distribution provisioning profile

We can now import the cer files into the key chain tool on a Mac by double clicking the file, on Windows the process is slightly more elaborate

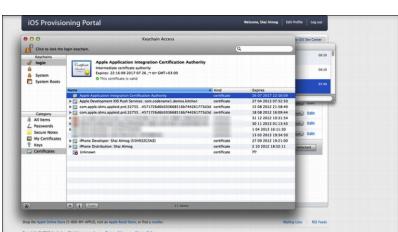


Figure 511. Import cer files

We can export the p12 files for the distribution and development profiles through the keychain tool

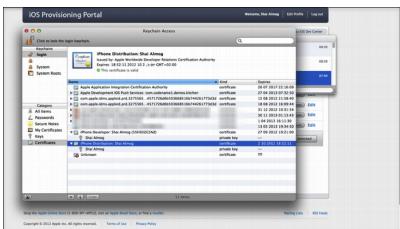


Figure 512. Export p12 files

In the IDE we enter the project settings, configure our provisioning profile, the password we typed when exporting and the p12 certificates. It is now possible to send the build to the server.

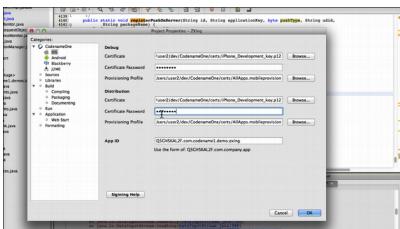


Figure 513. IOS Project Settings

17.4.1. iOS Code Signing Failure Checklist

Below is a list of common issues when signing and a set of suggestions for things to check. Notice that some of these signing failures will sometimes manifest themselves during build and sometimes will manifest during the install of the application.



Most of these issues aren't applicable when using the wizard e.g. a Mac isn't required for the certificate wizard as it uses the Codename One cloud

- You must use a Mac to generate P12 certificates manually.** The only workaround we found is the certificate wizard!

Notice that this is something you need to do once a year (generate P12), you will also need a Mac to upload your final app to the store at this time.

- When exporting the P12 certificate **make sure that you selected BOTH the public and the private keys** as illustrated here. If you only see one entry (no private key) then you created the CSR (singing request) on a different machine than the one where you imported the resulting CER file.

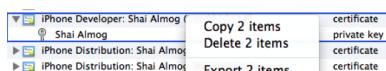


Figure 514. p12 export

- Make sure the package matches between the main preferences screen in the IDE and the iOS settings screen.



Figure 515. Package ID matching App ID

- Make sure the prefix for the app id in the iOS section of the preferences matches the one you have from Apple



Figure 516. App prefix

- Make sure your provisioning profile's app id matches your package name or is a * provisioning profile. Both are sampled in the pictures below, notice that you would need an actual package name for push/in-app-purchase support as well as for app store distribution.

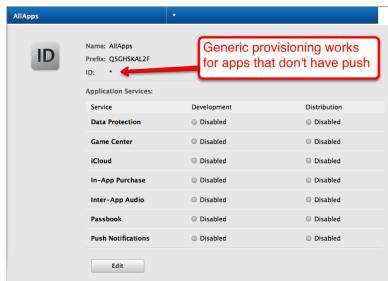


Figure 517. The star (*) Provisioning Profile

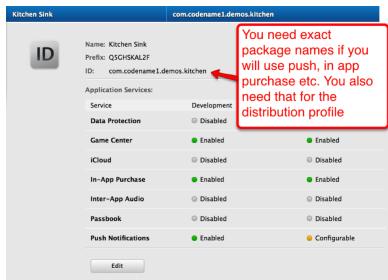


Figure 518. Provisioning Profile with app id

- Make sure the certificate and provisioning profile are from the same source (if you work with multiple accounts), notice that provisioning profiles and certificates expire so you will need to regenerate provisioning when your certificate expires or is revoked.
- If you declare push in the provisioning profile then `ios.includePush` (in the build arguments) MUST be set to true, otherwise it MUST be set to false (see pictures below). Notice that this should be configured via the UI in the iOS section.

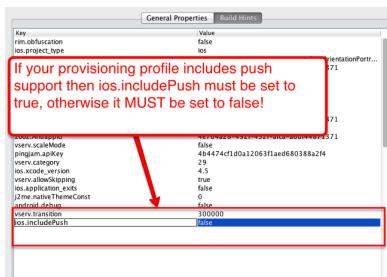


Figure 519. Include push build hint

17.5. Android

Signing Android applications is trivial when compared to the pain of iOS signing.

The NetBeans and Eclipse plugins have a simple wizard to generate the certificate that you can launch by pressing this button:

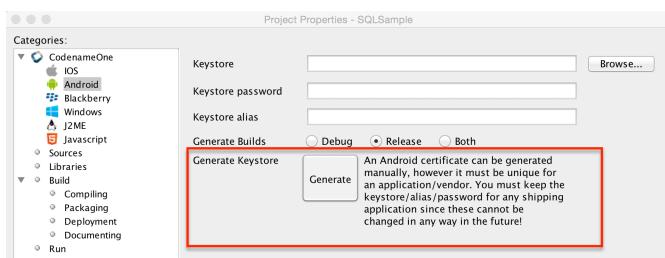


Figure 520. Android keystore generation wizard

Then fill out your details and password in the form:

The screenshot shows a 'Generate Android Key' dialog. It has fields for 'Key File Location' (set to '/Users/shai/keystore.ks'), 'Password', 'Verify Password', 'Alias', 'Full Name', 'Organization Unit', 'Company', 'City', 'State', and '2 Letter Country Code'. At the bottom are 'Cancel' and 'Generate' buttons. A red box highlights the 'Generate' button.

Figure 521. UI for Android certificate details

This will seamlessly generate a certificate for your project, you can reuse it for other projects as well.

17.5.1. Generating an Android Certificate Manually

You need the JDK's keytool executable (it should be under the JDK's bin directory) and execute the following command:

```
keytool -genkey -keystore Keystore.ks -alias [alias_name] -keyalg RSA -keysize 2048 -validity 15000 -dname "CN=[full name], OU=[ou], O=[comp], L=[City], S=[State], C=[Country Code]" -storepass [password] -keypass [password]
```

The elements in the brackets should be filled up based on this:

```
Alias: [alias_name] (just use your name/company name without spaces)
Full name: [full name]
Organizational Unit: [ou]
Company: [comp]
City: [City]
State: [State]
CountryCode: [Country Code]
Password: [password] (we expect both passwords to be identical)
```

Executing the command will produce a Keystore.ks file in that directory which you need to keep since if you lose it you will no longer be able to upgrade your applications! Fill in the appropriate details in the project properties or in the CodenameOne section in the Netbeans preferences dialog.

For more details see <http://developer.android.com/guide/publishing/app-signing.html>

17.6. RIM/BlackBerry

You can now get signing keys for free from Blackberry by going [here](https://www.blackberry.com/SignedKeys/) [<https://www.blackberry.com/SignedKeys/>]. Once you obtain the certificates you need to install them on your machine (you will need the Blackberry development environment for this). You will have two files: **sigtool.db** and **sigtool.csk** on your machine (within the JDE directory hierarchy). We need them and their associated password to perform the signed build for Blackberry application.

17.7. J2ME

Currently signing J2ME applications isn't supported. You can use tools such as the Sprint WTK to sign the resulting **jad/jar** produced by Codename One.

18. Working with iOS

18.1. Troubleshooting iOS Debug Build installs

In 9 cases out of 10 if you have a problem installing an app make sure your device is a 64 bit device. If not you will need to add the build hint `ios.debug.archs=armv7`. Notice that a 32 bit app will still work on a 64 bit device it will just display a performance warning.

If you have access to a Mac you can connect a cable and open xcode where you can use the device explorer console to look at messages which sometimes give a clue about what went wrong. If not here is a laundry list of a few things that might fail:

- Make sure you built the debug version and not the appstore version. The appstore version won't install on the device and can only be distributed via Apple's store or testflight
- Verify that you are sending a 32 bit build in the build hints using the build hint `ios.debug.archs=armv7`. It's only necessary if you have an older 32 bit device, see [this](https://www.codenameone.com/blog/moving-to-64bit-by-default.html) [https://www.codenameone.com/blog/moving-to-64bit-by-default.html]. Notice that this only applies for debug builds, release builds include both 32 and 64 bit versions



Devices prior to iPad Air & iPhone 5s were 32 bit devices so iPhone 5s won't need that flag but iPhone 5 or iPhone 5c will need it

- Check the the UDID is correct - if you got the UDID from an app then it's probably wrong as apps don't have access to the device UDID anymore. The way to get the UDID is either thru iOS Settings app or itunes
- Make sure the device isn't locked for installing 3rd party apps. I've had this when trying to install on my kids tablet which I configured to be child safe. This is configured in the settings as parental controls
- Check that you "own" the package name. E.g. if you previously installed an app with the same package name but a different certificate a new install will fail (this is true for Android too). So if you installed the kitchen sink from the store then built one of your own and installed it there will be a collision.

Notice that this might be problematic if you use overly generic package names as someone else might have used them which is why you must always use your own domain

- Make sure the device has a modern enough version of iOS for the dependencies. I think the current minimum for hello world is 6.0.1 but some apps might require a newer version e.g. Intercom requires OS 8 or newer
- Verify that you are using Safari when installing on the device (if you tried via cable that's not a problem), some developers had issues with firefox not launching the install process
- Check that the build hint `ios.includePush` is set in a way that matches your iOS provisioning. So it must be false if you don't have push within the provisioning profile

18.2. The iOS Screenshot/Splash Screen Process



As of version 5.0 (September 2018), screenshot generation is no longer enabled by default. Instead, the launch storyboard is used. Therefore much of the following section is no longer relevant.

iOS apps seem to start almost instantly in comparison to Android apps.

There is a trick to that, iOS applications have a file traditionally called `Default.png` that includes a **320x480** pixel image of the first screen of the application. This creates an "illusion" of the application instantly coming to life and filling up with data, this is rather clever but is a source trouble as the platform grows ^[11].



You can disable the screenshot process entirely with the `ios.fastBuild=true` build hint. This will only apply for debug builds so you don't need to worry about accidentally forgetting it in production.

The screenshot process was a pretty clever workaround but as Apple introduced the retina display **640x960** it required a higher resolution `Default@2x.png` file, then it added the iPad, iPad Retina and iPhone 5 ^[12] iPhone 6 & 6+ all of which required images of their own.

To make matters worse iPad apps (and iPhone 6+ apps) can be launched in landscape mode so that's two more resolutions for the horizontal orientation iPad. Overall as of this writing (or until Apple adds more resolutions) we need 16 screenshots for a typical iOS app:

Table 13. iOS Device Screenshot Resolutions

Resolution	File Name	Devices
320x480	<code>Default.png</code>	iPhone 3gs
640x960	<code>Default@2x.png</code>	iPhone 4x
640x1136	<code>Default-568h@2x.png</code>	iPhone 5x
1024x768	<code>Default-Portrait.png</code>	Non-retina ipads in portrait mode
768x1024	<code>Default-Landscape.png</code>	Non-retina ipads in landscape mode
2048x1536	<code>Default-Portrait@2x.png</code>	Retina ipads in portrait mode
1536x2048	<code>Default-Landscape@2x.png</code>	Retina ipads in landscape mode
750x1334	<code>Default-667h@2x.png</code>	iPhone 6
1242x2208	<code>Default-736h@3x.png</code>	iPhone 6 Plus Portrait
2208x1242	<code>Default-736h-Landscape@3x.png</code>	iPhone 6 Plus Landscape
2048x2732	<code>Default-iPadPro@2.png</code>	iPad Pro Portrait
2732x2048	<code>Default-iPadPro-Landscape@2.png</code>	iPad Pro Landscape
1668x2224	<code>Default-iPadProSmall@2.png</code>	10.5" iPad Pro Portrait

Resolution	File Name	Devices
2224x1668	Default-iPadProSmall-Landscape@2.png	10.5" iPad Pro Landscape
1125x2436	Default-iPhoneX@3.png	iPhone X Portrait
2436x1125	Default-iPhoneX-Landscape@3.png	iPhone X Landscape



You can predefine any of these files within the `native/ios` directory in your project. If the build server sees a file matching that exact name it will not generate a screenshot for that resolution

Native iOS developers can run their applications 16 times with blank data to grab these screenshots every time they change something in the first view of their application!

With Codename One this will not be feasible since the fonts and behavior might not match the device. Thus Codename One runs the application 16 times in the build servers, grabs the right sized screenshots in the simulator and then builds the app!

This means the process of the iPhone splash screen is almost seamless to the developer, however like every abstraction this too leaks.

The biggest problem developers have with this approach is for apps that use a web browser or native maps in the first screen of their app. This won't work well as those are native widgets. They will look different during the screenshot process.

Another problem is with applications that require a connection immediately on startup, this can fail for the build process.

A solution to both problems is to create a special case for the first launch of the app where no data exists. This will setup the screenshot process correctly and let you proceed with the app as usual.

18.2.1. Size

One of the first things we ran into when building one of our demos was a case where an app that wasn't very big in terms of functionality took up 30mb!

After inspecting the app we discovered that the iPad retina PNG files were close to 5mb in size... Since we had 2 of them (landscape and portrait) this was the main problem.

The iPad retina is a 2048x1536 device and with the leather theme the PNG images are almost impossible to compress because of the richness of details within that theme. This produced the huge screenshots that ballooned the application.

18.2.2. Mutable first screen

A very common use case is to have an application that pops up a login dialog on first run. This doesn't work well since the server takes a picture of the login screen and the login screen will appear briefly for future loads and will never appear again.

18.2.3. Unsupported component

One of the biggest obstacles is with heavyweight components, e.g. if you use a browser or maps on the first screen of the app you will see a partially loaded/distorted [MapComponent](https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html) [https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html] and the native webkit browser obviously can't be rendered properly by our servers.

The workaround for such issues is to have a splash screen that doesn't include any of the above. Its OK to show it for a very brief amount of time since the screenshot process is pretty fast.

18.3. Launch Screen Storyboards

With the shift to Xcode 9, which is the default version on the Codename One build servers as of [February 2018](https://www.codenameone.com/blog/xcode-9-on-by-default.html) [https://www.codenameone.com/blog/xcode-9-on-by-default.html], it is now possible to use a launch-screen storyboard as the splash screen instead of launch images. This will potentially solve the issue of the proliferation of screenshots, as you can supply a single storyboard which will work on all devices. Launch screen storyboards are enabled by default (as of version 5.0/September 2018). You can disable them by adding the `ios.generateSplashScreens=true` build hint.

18.3.1. Launch Storyboard vs Launch Images

The key benefit of using a launch storyboard right now is that it allows your app to be used in split-screen mode. Storyboards, however, work a little bit differently than launch images. They don't show a screenshot of the first page of your app. The default Codename One launch storyboard simply shows your app's icon in the middle of the screen. You can customize the launch screen by providing one or more of the following files in your project's native/ios directory

1. `Launch.Foreground.png` - Will be shown instead of your app's icon in the center of the screen.
2. `Launch.Background.png` - Will fill the background of the screen.
3. `LaunchScreen.storyboard` - A custom storyboard developed in Xcode, that will be used instead of the default storyboard.



Make sure to add the `ios.multitasking=true` build hint, or your launch storyboard will not be used.



Changes to the launch screen will not take effect until the device has been restarted. I.e. If you install your app on a device, then you make changes to the launch screen and update the app, the launch screen won't change until the device is restarted.

18.4. Local Notifications on iOS and Android

Local notifications are similar to push notifications, except that they are initiated locally by the app, rather than remotely. They are useful for communicating information to the user while the app is running in the background, since they manifest themselves as pop-up notifications on supported devices.



To set the notification icon on Android place a 24x24 icon named `ic_stat_notify.png` under the `native/android` folder of the app. The icon can be white with transparency areas

18.4.1. Sending Notifications

The process for sending a notification is:

1. Create a `LocalNotification` [<https://www.codenameone.com/javadoc/com/codename1/notifications/LocalNotification.html>] object with the information you want to send in the notification.
2. Pass the object to `Display.scheduleLocalNotification()`.

Notifications can either be set up as one-time only or as repeating.

Example Sending Notification

```
LocalNotification n = new LocalNotification();
n.setId("demo-notification");
n.setAlertBody("It's time to take a break and look at me");
n.setAlertTitle("Break Time!");
n.setAlertSound("beep-01a.mp3");

Display.getInstance().scheduleLocalNotification(
    n,
    System.currentTimeMillis() + 10 * 1000, // fire date/time
    LocalNotification.REPEAT_MINUTE // Whether to repeat and what frequency
);
```

The resulting notification will look like



Figure 522. Resulting notification in iOS

18.4.2. Receiving Notifications

The API for receiving/handling local notifications is also similar to push. Your application's main lifecycle class needs to implement the `com.codename1.notifications.LocalNotificationCallback`

interface which includes a single method:

```
public void localNotificationReceived(String notificationId)
```

The `notificationId` parameter will match the `id` value of the notification as set using `LocalNotification.setId()`.

Example Receiving Notification

```
public class BackgroundLocationDemo implements LocalNotificationCallback {  
    //...  
  
    public void init(Object context) {  
        //...  
    }  
  
    public void start() {  
        //...  
    }  
  
    public void stop() {  
        //...  
    }  
  
    public void destroy() {  
        //...  
    }  
  
    public void localNotificationReceived(String notificationId) {  
        System.out.println("Received local notification "+notificationId);  
    }  
}
```

 `localNotificationReceived()` is only called when the user responds to the notification by tapping on the alert. If the user doesn't click on the notification, then this event handler will never be fired.

18.4.3. Canceling Notifications

Repeating notifications will continue until they are canceled by the app. You can cancel a single notification by calling:

```
Display.getInstance().cancelLocalNotification(notificationId);
```

Where `notificationId` is the string id that was set for the notification using

`LocalNotification.setId()`.

18.5. iOS Beta Testing (Testflight)

Apple provides the ability to distribute beta versions of your application to beta testers using testflight. This allows you to recruit up to 1000 beta testers without the typical UDID limits a typical Apple account has.



This is supported for pro users as part of the crash protection feature.

To take advantage of that capability use the build hint `ios.testFlight=true` and then submit the app to the store for beta testing. Make sure to use a release build target.

18.6. Accessing Insecure URL's

Due to security exploits Apple blocked some access to insecure URL's which means that http code that worked before could stop working for you on iOS 9+. This is generally a good move, you should use https and avoid http as much as possible but that's sometimes impractical especially when working with an internal or debug environment.

You can disable the strict URL checks from Apple by using the venerable `ios.plistInject` build hint and setting it to:

```
<key>NSAppTransportSecurity</key><dict><key>NSAllowsArbitraryLoads</key><true/></dict>
```

However, it seems that Apple will reject your app if you just include that and don't have a good reason.

18.7. Using Cocoapods

CocoaPods [<https://cocoapods.org/>] is a dependency manager for Swift and Objective-C Cocoa projects. It has over eighteen thousand libraries and can help you scale your projects elegantly. Cocoapods can be used in your Codename One project to include native iOS libraries without having to go through the hassle of bundling the actual library into your project. Rather than bundling .h and .a files in your ios/native directory, you can specify which "pods" your app uses via the `ios.pods` build hint. (There are other build hints also if you need more advanced features).

Examples

Include the [AFNetworking](https://github.com/AFNetworking/AFNetworking) [<https://github.com/AFNetworking/AFNetworking>] library in your app:

```
ios.pods=AFNetworking
```

Include the [AFNetworking](https://github.com/AFNetworking/AFNetworking) [<https://github.com/AFNetworking/AFNetworking>] version 3.0.x library in your app:

```
ios.pods=AFNetworking ~> 3.0
```

For full versioning syntax specifying pods see the [Podfile spec for the "pod" directive](#) [<https://guides.cocoapods.org/syntax/podfile.html#pod>].

18.7.1. Including Multiple Pods

Multiple pods can be separated by either commas or semi-colons in the value of the `ios.pods` build hint. E.g. To include GoogleMaps and AFNetworking, you could:

```
ios.pods=GoogleMaps,AFNetworking
```

Or specifying versions:

```
ios.pods=AFNetworking ~> 3.0,GoogleMaps
```

18.7.2. Other Pod Related Build Hints

`ios.pods.platform` : The minimum platform to target. In some cases, Cocoapods require functionality that is not in older version of iOS. For example, the GoogleMaps pod requires iOS 7.0 or higher, so you would need to add the `ios.pods.platform=7.0` build hint.

`ios.pods.sources` : Some pods require that you specify a URL for the source of the pod spec. This may be optional if the spec is hosted in the central CocoaPods source (<https://github.com/CocoaPods/Specs.git>).

18.7.3. Converting PodFile To Build Hints

Most documentation for Cocoapods "pods" provide instructions on what you need to add to your Xcode project's PodFile. Here is an example from the GoogleMaps cocoapod to show you how a PodFile can be converted into equivalent build hints in a Codename One project.

The GoogleMaps cocoapod directs you to add the following to your PodFile:

```
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '7.0'
pod 'GoogleMaps'
```

This would translate to the following build hints in your Codename One project:

```
ios.pods.sources=https://github.com/CocoaPods/Specs.git
ios.pods.platform=7.0
ios.pods=GoogleMaps
```

(Note that the `ios.pods.sources` directive is optional).

18.8. Including Dynamic Frameworks

If you need to use a dynamic framework (e.g. `SomeThirdPartySDK.framework`), and it isn't available via cocoapods, then you can add it to your project by simply zipping up the framework and copying it to your native/ios directory.

e.g. `native/ios/SomeThirdPartySDK.framework.zip`

There are no build hints necessary for this approach. The build server will automatically detect the framework and link it into your app.

[11] Apple provided another trick with XIB files starting with iOS 8 but that doesn't apply to games or Codename One. It has its own set of problems

[12] slightly larger screen and different aspect ratio

19. Working with JavaScript

This section covers the Codename One Javascript port, which allows you to compile your app as native javascript and run it inside a browser. This is different from the [BrowserComponent](https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html>] and other methods of displaying HTML/Javascript inside a Codename One app.

19.1. Limitations of the Javascript Port

19.1.1. No Multithreaded Code inside Static Initializers



This section pertains to Codename One 3.6 and older. Newer versions of Codename One support multithreaded code inside static initializers now.

Codename One's Javascript port uses [TeaVM](http://teavm.org/) [<http://teavm.org/>] to compile your application directly to Javascript so that it can run inside modern web browsers without the need for any plugins (i.e. NOT as an applet). One of the revolutionary features that TeaVM provides is the ability to run multi-threaded code (i.e. it has full support for `Object.wait()`, `Object.notify()`, `Object.notifyAll()`, and the `synchronized` keyword). The one caveat to be aware of is that **you cannot use any threading primitives inside static initializers**. This is due to technical limitations in the browser environment and the way that TeaVM compiles class definitions into Javascript. The workaround for this issue is to do lazy initialization in cases where you need to use multithreaded code.

Example

The following code will result in a build error when deploying a Javascript build:

Class1.java

```
import com.codename1.io.Log;
class Class1 {
    public static int getValue() {
        Log.p("Hello world");
        return 1;
    }
}
```

Class2.java

```
class Class2 {
    public static int value = Class1.getValue();
}
```

This fails because `Class2` calls `Class1.getValue()` in its static initializer, and `getValue()` calls `Log.p()`, which, underneath the covers, writes to Storage - which involves some synchronous network access

in the Javascript port (i.e. it uses `wait()` and `notify()` under the hood).

If we simply remove the call to `Log.p()` from `getValue()`, as follows:

```
public static int getValue() {  
    return 1;  
}
```

Then everything would be fine.

But How do we Know if A method includes `wait()/notify` somewhere along the line?

When you try to build your app as a Javascript app, it will fail (if code in your static initializers uses `wait()/notify()` somewhere along the line).

How to Work Around this Issue

Use lazy initialization wherever you can. You don't need to worry about this for setting static variables to literal values. E.g.: `static int someVal = 20;` will always be fine. But `static int someVal = OtherClass.calculateSomeVal();` may or may not be fine, because you don't know whether `calculateSomeVal()` uses a `wait/notify`. So instead of initializing `someVal` in the static initializer, create a static accessor that lazily initializes it. Or initialize it inside your app's `init()` method. Or initialize it inside the class constructor.

19.2. Troubleshooting Build Errors

If your Javascript build fails, you should download the error log and see what the problem is. The most common errors are:

1. "[ERROR] Method XXX.<clinit>()V is claimed to be synchronous, but it has invocations of asynchronous methods"

This error will occur if you have static initializers that use multithreaded code (e.g. `wait/notify/sleep`, etc...). See [Static Initializers](#) for information about troubleshooting this error.

In some cases TeaVM may give a false-positive here (i.e. it **thinks** you are doing some multithreaded stuff, but you're really not), then you can force the build to "succeed" by adding the `javascript.stopOnErrors=false` build hint.

2. "Method XXX was not found"

TeaVM uses its own Java runtime library. It is mostly complete, but you may occasionally run into methods that haven't been implemented. If you run into errors saying that certain classes or methods were not found, please post them to the [Codename One issue tracker](#) [<https://github.com/codenameone/CodenameOne/issues>]. You can also work around these by changing your own code to not use such functions. If this missing method doesn't fall on a critical path on your app, you can also force the app to still build despite this error by adding the `javascript.stopOnErrors=false` build hint.

19.3. ZIP, WAR, or Preview. What's the difference?

The **Javascript** build target will result in up to three different bundles being generated:

1. YourApp-1.0.war
2. YourApp-1.0.zip
3. YourApp-1.0-Preview.html

YourApp-1.0.war is a self contained application bundle that can be installed in any JavaEE servlet container. If you haven't customized any [proxy settings](#), then the application will be configured to use a proxy servlet that is embedded into the .war file.

As an example, the PropertyCross .war file contains the following files:

```
$ jar tvf PropertyCross-1.0.war
 0 Thu Apr 30 15:57:38 PDT 2015 META-INF/
132 Thu Apr 30 15:57:36 PDT 2015 META-INF/MANIFEST.MF
 0 Thu Apr 30 15:57:36 PDT 2015 assets/
 0 Thu Apr 30 15:57:36 PDT 2015 assets/META-INF/
 0 Thu Apr 30 15:57:36 PDT 2015 js/
 0 Thu Apr 30 15:57:36 PDT 2015 teavm/
 0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/
 0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/
 0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/com/
 0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/com/codename1/
 0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/com/codename1/corsproxy/
 0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/lib/
27568 Thu Apr 30 15:57:12 PDT 2015 assets/CN1Resource.res
306312 Thu Apr 30 15:57:12 PDT 2015 assets/iOS7Theme.res
427737 Thu Apr 30 15:57:12 PDT 2015 assets/iPhoneTheme.res
 350 Thu Apr 30 15:57:12 PDT 2015 assets/META-INF/MANIFEST.MF
92671 Thu Apr 30 15:57:12 PDT 2015 assets/theme.res
23549 Thu Apr 30 15:57:14 PDT 2015 icon.png
 2976 Thu Apr 30 15:57:14 PDT 2015 index.html
30695 Thu Apr 30 15:57:12 PDT 2015 js/fontmetrics.js
84319 Thu Apr 30 15:57:12 PDT 2015 js/jquery.min.js
13261 Thu Apr 30 15:57:12 PDT 2015 progress.gif
 2816 Thu Apr 30 15:57:12 PDT 2015 style.css
1886163 Thu Apr 30 15:57:36 PDT 2015 teavm/classes.js
359150 Thu Apr 30 15:57:36 PDT 2015 teavm/classes.js.map
1147502 Thu Apr 30 15:57:36 PDT 2015 teavm/classes.js.teavmdbg
 30325 Thu Apr 30 15:57:36 PDT 2015 teavm/runtime.js
 1011 Thu Apr 30 15:57:18 PDT 2015 WEB-INF/classes/com/codename1/corsproxy/CORSProxy.class
232771 Wed Nov 05 17:35:12 PST 2014 WEB-INF/lib/commons-codec-1.6.jar
 62050 Wed Apr 15 14:35:56 PDT 2015 WEB-INF/lib/commons-logging-1.1.3.jar
590004 Wed Apr 15 14:35:58 PDT 2015 WEB-INF/lib/httpclient-4.3.4.jar
282269 Wed Apr 15 14:35:56 PDT 2015 WEB-INF/lib/httpcore-4.3.2.jar
 14527 Wed Apr 15 14:35:56 PDT 2015 WEB-INF/lib/smiley-http-proxy-servlet-1.6.jar
 903 Thu Apr 30 15:57:12 PDT 2015 WEB-INF/web.xml
9458 Thu Apr 30 15:57:14 PDT 2015 META-INF/maven/com.propertycross/PropertyCross/pom.xml
 113 Thu Apr 30 15:57:36 PDT 2015 META-INF/maven/com.propertycross/PropertyCross/pom.properties
```

Some things to note in this file listing:

1. The **index.html** file is the entry point to the application.
2. **CORSProxy.class** is the proxy servlet for making network requests to other domains.
3. The **assets** directory contains all of your application's **jar** resources. All resource files in your app will end up in this directory.
4. The **teavm** directory contains all of the generated javascript for your application. Notice that there are some debugging files generated (**classes.js.map** and **classes.js.teavmdbg**). These are not normally loaded by the browser when your app is run, but they can be used by Chrome when you are doing debugging.
5. The **jar** files in the **WEB-INF/lib** directory are dependencies of the proxy servlet. They are not required for your app to run - unless you are using the proxy.

YourApp-1.0.zip is appropriate for deploying the application on any web server. It contains all of the same files as the .war file, excluding the WEB-INF directory (i.e. it doesn't include any servlets, class files, or Java libraries - it contains purely client-side javascript files and HTML).

As an example, this is a listing of the files in the zip distribution of the PropertyCross demo:

```
$ unzip -vl PropertyCross-1.0.zip
Archive: /path/to/PropertyCross-1.0.zip
      Length   Method    Size  Ratio Date  Time    CRC-32     Name
----- -----
    27568 Defl:N    26583   4% 04-30-15 15:57 9dc91739 assets/CN1Resource.res
  306312 Defl:N   125797  59% 04-30-15 15:57 0b5c1c3a assets/iOS7Theme.res
  427737 Defl:N   218975  49% 04-30-15 15:57 3de499c8 assets/iPhoneTheme.res
      350 Defl:N     241  31% 04-30-15 15:57 7e7e3714 assets/META-INF/MANIFEST.MF
  92671 Defl:N    91829   1% 04-30-15 15:57 004ad9d7 assets/theme.res
  23549 Defl:N   23452   0% 04-30-15 15:57 acd79066 icon.png
   2903 Defl:N     1149  60% 04-30-15 15:57 e5341de1 index.html
  30695 Defl:N     7937  74% 04-30-15 15:57 2e008f6c js/fontmetrics.js
  84319 Defl:N   29541  65% 04-30-15 15:57 15b91689 js/jquery.min.js
  13261 Defl:N   11944  10% 04-30-15 15:57 51b895c7 progress.gif
   2816 Defl:N     653  77% 04-30-15 15:57 a12159c7 style.css
1886163 Defl:N   315437  83% 04-30-15 15:57 2b34c50f teavm/classes.js
  359150 Defl:N   92874  74% 04-30-15 15:57 30abdf13 teavm/classes.js.map
 1147502 Defl:N   470472  59% 04-30-15 15:57 e5c456f7 teavm/classes.js.teavmdbg
   30325 Defl:N     5859  81% 04-30-15 15:57 46651f06 teavm/runtime.js
----- -----
  4435321                1422743 68%
                                         15 files
```

You'll notice that it has many of the same files as the .war distribution. It is just missing the the proxy servlet and dependencies.

YourApp-1.0-Preview.html is a single-page HTML file with all of the application's resources embedded into a single page. This is generated for convenience so that you can preview your application on the build server directly. While you could use this file in production, you are probably better to use the ZIP or WAR distribution instead as some mobile devices have file size

limitations that may cause problems for the "one large single file" approach. If you do decide to use this file for your production app (i.e. copy the file to your own web server), you will need to change the proxy settings, as it is configured to use the proxy on the Codename One build server - which won't be available when the app is hosted on a different server.

19.4. Setting up a Proxy for Network Requests

The Codename One API includes a network layer (the [NetworkManager](#) [<https://www.codenameone.com/javadoc/com/codename1/io/NetworkManager.html>] and [ConnectionRequest](#) [<https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html>] classes) that allows you to make HTTP requests to arbitrary destinations. When an application is running inside a browser as a Javascript app, it is constrained by the same origin policy. You can only make network requests to the same host that served the app originally.

E.g. If your application is hosted at <http://example.com/myapp/index.html>, then your app will be able to perform network requests to retrieve other resources under the **example.com** domain, but it won't be able to retrieve resources from **example2.com**, **foo.net**, etc..



The HTTP standard does support cross-origin requests in the browser via the **Access-Control-Allow-Origin** HTTP header. Some web services supply this header when serving resources, but not all. The only way to be make network requests to arbitrary resources is to do it through a proxy.

Luckily there is a solution. The .war javascript distribution includes an embedded proxy servlet, and your application is configured, by default, to use this servlet. If you intend to use the .war distribution, then it **should just work**. You shouldn't need to do anything to configure the proxy.

If, however, you are using the .zip distribution or the single-file preview, you will need to set up a Proxy servlet and configure your application to use it for its network requests.

19.4.1. Step 1: Setting up a Proxy



This section is only relevant if you are using the .zip or single-file distributions of your app. You shouldn't need to set up a proxy for the .war distribution since it includes a proxy built-in.

The easiest way to set up a proxy is to use the Codename One [cors-proxy](#) [<https://github.com/shannah/cors-proxy>] project. This is the open-source project from which the proxy in the .war distribution is derived. Simply download and install the cors-proxy .war file in your JavaEE compatible servlet container.

If you don't want to install the .war file, but would rather just copy the proxy servlet into an existing web project, you can do that also. [See the cors-proxy wiki for more information about this](#) [<https://github.com/shannah/cors-proxy/wiki/Embedding-Servlet-into-Existing-Project>].

19.4.2. Step 2: Configuring your Application to use the Proxy

There are three ways to configure your application to use your proxy.

1. Using the `javascript.proxy.url` build hint.

E.g.:

```
javascript.proxy.url=http://example.com/myapp/cn1-cors-proxy?_target=
```

2. By modifying your app's `index.html` file after the build.

E.g.:

```
<script type="text/javascript">
    window.cn1CORSProxyURL='http://example.com/myapp/cn1-cors-proxy?_target=';
</script>
```

3. By setting the `javascript.proxy.url` property in your Java source. Generally you would do this inside your `init()` method, but it just has to be executed before you make a network request that requires the proxy.

```
Display.getInstance().setProperty(
    "javascript.proxy.url",
    "http://example.com/myapp/cn1-cors-proxy?_target="
);
```

The method you choose will depend on the workflow that you prefer. Options #1 and #3 will almost always result in fewer changes than #2 because you only have to set them up once, and the builds will retain the settings each time you build your project.

19.5. Using the CORS Proxy for Same Origin Requests

By default, the CORS proxy is only used for HTTP requests to URLs at a different domain than the one that the app is running in. There are some circumstances where you may want to **even** use the proxy for same domain requests. You can do this by setting the `javascript.useProxyForSameDomain` display property to `true`. E.g.

```
Display.getInstance().setProperty("javascript.useProxyForSameDomain", "true");
```

Why would you want to do this?

The browser shields some HTTP headers (e.g. "Set-Cookie") from Javascript so that your app cannot access them. Going through the proxy works around this limitation by copying and encoding such headers in a format that the browser will allow, and then decoding them client-side to make them available to your app seamlessly.

Using Apache as a Proxy

If you are hosting your application on an Apache 2 web server with mod_proxy installed, and you only need to make CORS requests to a single domain (or a limited set of domains), you can use Apache to serve as your proxy. One sample configuration (which you would place either in your VirtualHost definition or your .htaccess file) is as follows:

```
SSLProxyEngine on  
ProxyPass /app https://www.myexternaldomain.com  
ProxyPassReverse /app https://www.myexternaldomain.com
```

This tells Apache to proxy all requests for '/app' to the domain <https://www.myexternaldomain.com>. You would then need to set your CORS proxy URL in your CN1 app to "/app/".

The syntax is the same if you have multiple domains, but keep attention to the order of the lines to make the proxy working correctly. For example:

```
SSLProxyEngine on  
ProxyPass /app https://www.myexternaldomain1.com  
ProxyPassReverse /app https://www.myexternaldomain1.com  
ProxyPass /storage https://www.myexternaldomain2.com  
ProxyPassReverse /storage https://www.myexternaldomain2.com
```

This tells Apache to proxy all requests for '/app' to the domain <https://www.myexternaldomain1.com> and all requests for '/storage' to the domain <https://www.myexternaldomain2.com>

19.6. Customizing the Splash Screen

Since your application may include many resource files, videos, etc., the build-server will generate a splash screen for your app to display while it is loading. This basically shows a progress indicator with your app's icon.

You can customize this splash screen by simply modifying the HTML source inside the **cn1-splash** **div** tag of your app's index.html file:

```
<div id="cn1-splash">  
      
  
      
    <p>...Loading...</p>  
</div>
```

19.7. Debugging

If you run into problems with your app that only occur in the Javascript version, you may need to

do a little bit of debugging. There are many debugging tools for Javascript, but the preferred tool for debugging Codename One apps is Chrome's debugger.

If your application crashes and you don't have a clue where to begin, follow these steps:

1. Load your application in Chrome.
2. Open the Chrome debugger.
3. Enable the "Pause on Exceptions" feature, then click the "Refresh" button to reload your app.
4. Step through each exception until you reach the one you are interested in. Chrome will then show you a stack trace that includes the name of the Java source file and line numbers.

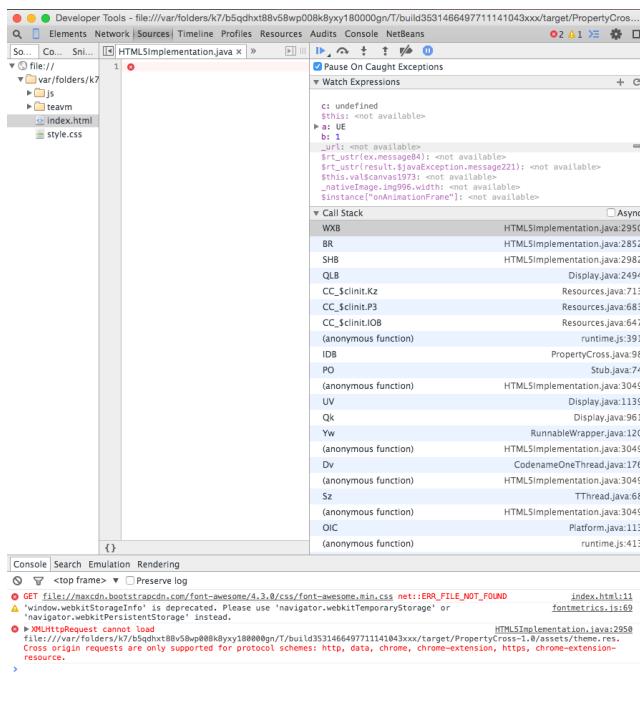


Figure 523. Debugging using Chrome tools

19.8. Including Third-Party Javascript Libraries

Codename One allows you to interact directly with Javascript using native interfaces. Native interfaces are placed inside your project's `native/javascript` directory using a prescribed naming convention. If you want to, additionally, include third-party Javascript libraries in your application you should also place these libraries inside the `native/javascript` directory but you must specify which files should be treated as "libraries" and which files are treated as "resources". You can do this by adding a file with extension `.cn1mf.json` file either the root of your `native/javascript` directory or the root level of the project's `src` directory.

19.8.1. Libraries vs Resources

A **resource** is a file whose contents can be loaded by your application at runtime using `Display.getInstance().getResourceAsStream()`. In a typical Java environment, resources would be stored on the application's classpath (usually inside a Jar file). On iOS, resources are packaged inside the application bundle. In the Javascript port, resources are stored inside the `APP_ROOT/assets` directory. Historically, javascript files have always been treated as resources in Codename One, and

many apps include HTML and Javascript files for use inside the [BrowserComponent](https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html) [<https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html>].

With the Javascript port, it isn't quite so clear whether a Javascript file is meant to be a resource or a library that the application itself uses. Most of the time you probably want Javascript files to be used as libraries, but you might also have Javascript files in your app that are meant to be loaded at runtime and displayed inside a Web View - these would be considered resources.

19.8.2. The Javascript Manifest File

In order to differentiate libraries from resources, you should provide a `cn1mf.json` file inside your `native/javascript` directory that specifies any files or directories that should be treated as libraries. This file can be named anything you like, as long as its name ends with `cn1mf.json`. Any files or directories that you list in this manifest file will be packaged inside your app's `includes` directory instead of the `assets` directory. Additionally it adds appropriate `<script>` tags to include your libraries as part of the `index.html` page of your app.



If you include the `cn1mf.json` file in your project's `src` directory it could potentially be used to add configuration parameters to platform's other than Javascript (although currently no other platforms use this feature). If you place it inside your `native/javascript` directory, then only the Javascript port will use the configuration contained therein.

A simple manifest file might contain the following JSON:

```
{  
  "javascript" : {  
    "libs" : [  
      "mylib1.js"  
    ]  
  }  
}
```

I.e. It contains an object with key **libs** whose value is a list of files that should be treated as libraries. In the above example, we are declaring that the file `native/javascript/mylib1.js` should be treated as a library. This will result in the following `<script>` tag being added to the `index.html` file:

```
<script src="includes/mylib1.js"></script>
```



This also caused the `mylib1.js` file to be packaged inside the `includes` directory instead of the `assets` directory.



A project may contain more than one manifest file. This allows you to include manifest files with your cn1libs also. You just need to make sure that each manifest file has a different name.

How to NOT generate the `<script>` tag

In some cases you may want a Javascript file to be treated as a library (i.e. packaged in the `includes` directory) but not automatically included in the `index.html` page. Rather than simply specifying the name of the file in the `libs` list, you can provide a structure with multiple options about the file. E.g.

```
{  
  "javascript" : {  
    "libs" : [  
      "mylib1.js",  
      {  
        "file" : "mylib2.js",  
        "include" : false  
      }  
    ]  
  }  
}
```

In the above example, the `mylib2.js` file will be packaged inside the `includes` directory, but the build server won't insert its `<script>` tag in the `index.html` page.

Library Directories

You can also specify directories in the manifest file. In this case, the entire directory will be packaged inside the `includes` directory of your app.



If you are including Javascript files in your app that are contained inside a directory hierarchy, you should specify the root directory of the hierarchy in your manifest file and use the sub "includes" property of the directory entry to specify which files should be included with `<script>` tags. Specifying the file directly inside the "libs" list will result in the file being packed directly in the your app's `includes` directory. This may or may not be what you want.

E.g.

```
{
  "javascript" : {
    "libs" : [
      "mylib1.js",
      {
        "file" : "mylib2.js",
        "include" : false
      },
      {
        "file" : "mydir1",
        "includes" : ["subfile1.js", "subfile2.js"]
      }
    ]
  }
}
```

In this example the entire `mydir1` directory would be packed inside the app's `includes` directory, and the following `script` tags would be inserted into the `index.html` file:

```
<script src="includes/mydir1/subfile1.js"></script>
<script src="includes/mydir1/subfile2.js"></script>
```



Libraries included from a directory hierarchy may not work correctly with the single file preview that the build server generates. For that version, it will embed the contents of each included Javascript file inside the `index.html` file, but the rest of the directory contents will be omitted. If your library depends on the directory hierarchy and supporting files and you require the single-file preview to work, then you may consider hosting the library on a separate server, and including the library directly from there, rather than embedding it inside your project's "native/javascript" directory.

Including Remote Libraries

The examples so far have only demonstrated the inclusion of libraries that are part of the app bundle. However, you can also include libraries over the network by specifying the URL to the library directly. This is handy for including common libraries that are hosted by a CDN.

E.g. The Google Maps library requires the Google maps API to be included. This is accomplished with the following manifest file contents:

```
{
  "javascript" : {
    "libs" : [
      "//maps.googleapis.com/maps/api/js?v=3.exp"
    ]
  }
}
```

 This example uses the "://" prefix for the URL instead of specifying the protocol directly. This allows the library to work for both http and https hosting. You could however specify the protocol as well:

+

```
{
  "javascript" : {
    "libs" : [
      "https://maps.googleapis.com/maps/api/js?v=3.exp"
    ]
  }
}
```

Including CSS Files

CSS files can be included using the same mechanism as is used for Javascript files. If the file name ends with ".css", then it will be treated as a CSS file (and included with a [link](#) tag instead of a [script](#) tag. E.g.

```
{
  "javascript" : {
    "libs" : [
      "mystyles.css"
    ]
  }
}
```

or

```
{
  "javascript" : {
    "libs" : [
      "https://example.com/mystyles.css"
    ]
  }
}
```

Embedding Variables in URLs

In some cases the URL for a library may depend on the values of some build hints in the project. For example, in the Google Maps cn1lib, the API key must be appended to the URL for the API as a GET parameter. E.g. https://maps.googleapis.com/maps/api/js?v=3.exp&key=SOME_API_KEY, but the developer of the library doesn't want to put his own API key in the manifest file for the library. It would be better for the API key to be supplied by the developer of the actual app that uses the library and not the library itself.

The solution for this is to add a **variable** into the URL as follows:

```
{  
    "javascript" : {  
        "libs" : [  
            "//maps.googleapis.com/maps/api/js?v=3.exp&key={{javascript.googlemaps.key}}"  
        ]  
    }  
}
```

The `{{javascript.googlemaps.key}}` variable will be replaced with the value of the `javascript.googlemaps.key` build hint by the build server, so the resulting include you see in the index.html page will be something like:

```
<script src="//maps.googleapis.com/maps/api/js?v=3.exp&key=XYZ"></script>
```

19.9. Browser Environment Variables

Native interfaces allow you to interact with the Javascript environment in unlimited ways, but Codename One provides a simpler method of obtaining some common environment information from the browser via the `Display.getInstance().getProperty()` method. The following environment variables are currently available:

Table 14. Property hints for the JavaScript port

Name	Description
<code>browser.window.location.href</code>	A String, representing the entire URL of the page, including the protocol (like http://)
<code>browser.window.location.search</code>	A String, representing the querystring part of a URL, including the question mark (?)
<code>browser.window.location.host</code>	A String, representing the domain name and port number, or the IP address of a URL
<code>browser.window.location.hash</code>	A String, representing the anchor part of the URL, including the hash sign (#)

Name	Description
<code>browser.window.location.origin</code>	A String, representing the protocol (including ://), the domain name (or IP address) and port number (including the colon sign :) of the URL. For URL's using the "file:" protocol, the return value differs between browsers
<code>browser.window.location.pathname</code>	A String, representing the pathname
<code>browser.window.location.protocol</code>	A String, representing the protocol of the current URL, including the colon (:)
<code>browser.window.location.port</code>	A String, representing the port number of a URL. + Note: If the port number is not specified or if it is the scheme's default port (like 80 or 443), an empty string is returned
<code>browser.window.location.hostname</code>	A String, representing the domain name, or the IP address of a URL
<code>User-Agent</code>	The User-agent string identifying the browser, version etc..
<code>browser.language</code>	The language code that the browser is currently set to. (e.g. en-US)
<code>browser.name</code>	the name of the browser as a string.
<code>Platform</code>	a string that must be an empty string or a string representing the platform on which the browser is executing. + For example: "MacIntel", "Win32", "FreeBSD i386", "WebTV OS"
<code>browser.codeName</code>	the internal name of the browser
<code>browser.version</code>	the version number of the browser
<code>javascript.deployment.type</code>	Specifies the deployment type of the app. This will be "file" for the single-file preview, "directory" for the zip distribution, and "war" for the war distribution.

19.10. Changing the Native Theme

Since a web application could potentially be run on any platform, it isn't feasible to bundle all possible themes into the application (at least it wouldn't be efficient for most use cases). By default we have bundled the iOS7 theme for javascript applications. This means that the app will look like iOS7 on all devices: desktop, iOS, Android, WinPhone, etc...

You can override this behavior dynamically by setting the `javascript.native.theme` `Display` [<https://www.codenameone.com/javadoc/com/codename1/ui/Display.html>] property to a theme that you have included in your app. All of the native themes are available on GitHub, so you can easily copy these into your application. The best place to add the theme is in your `native/javascript` directory - so that they won't be included for other platforms.

19.10.1. Example: Using Android Theme on Android



As of Codename One 6.0, apps will automatically use the Android theme when run on an Android device, so this example is not necessary. However the technique of changing the native theme at runtime is still applicable.

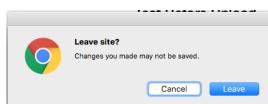
First, download `androidTheme.res` [<https://github.com/codenameone/CodenameOne/raw/master/Ports/Android/src/androidTheme.res>] from the Android port on GitHub, and copy it into your app's `native/javascript` directory.

Then in your app's `init()` method, add the following:

```
Display d = Display.getInstance();
if (d.getProperty("User-Agent", "Unknown").indexOf("Android") != -1) {
    d.setProperty("javascript.native.theme", "/androidTheme.res");
}
```

19.11. Disabling the 'OnBeforeUnload' Handler

By default, apps will display warning/confirm dialog when the user attempts to leave the page.



You can explicitly enable or disable this behaviour by setting the "platformHint.javascript.beforeUnloadMessage" display property. Setting the property to `null` will disable this behaviour, so that users will not be harassed by this dialog when they navigate away from the app. Setting it to a string value, like "leaving so soon?", will re-enable this behaviour.



Some browsers don't allow you to specify the message that is displayed in this dialog. In those browsers, this property can be viewed as boolean: A null value will result in no prompt being shown, and a non-null value will result in a prompt being shown.

19.11.1. Example: Toggling the BeforeUnload Prompt On/Off

```
Form f = new Form("Test Before Unload", BoxLayout.y());
CheckBox enableBeforeUnload = new CheckBox("Enable Before Unload");
enableBeforeUnload.setSelected(true);
enableBeforeUnload.addActionListener(e->{
    if (enableBeforeUnload.isSelected()) {
        CN.setProperty("platformHint.javascript.beforeUnloadMessage", "Are you sure you want to leave this page? It
might be bad!");
    } else {
        CN.setProperty("platformHint.javascript.beforeUnloadMessage", null);
    }
});
f.add(enableBeforeUnload);
f.show();
```

19.12. Deploying as a Progressive Web App

Out of the box, your app is ready to be deployed as a progressive web app (PWA). That means that users can access the app directly in their browser, but once the browser determines that the user is frequenting the app, it will "politely" prompt the user to install the app on their home screen. Once installed on the home screen, the app will behave just like a native app. It will continue to work while offline, and if the user launches the app, it will open without the browser's navigation bar. If you were to install the native and PWA versions of your app side by side, you would be hard pressed to find the difference - especially on newer devices.

Below is a screenshot from Chrome for Android where the browser is prompting the user to add the app to their home screen.

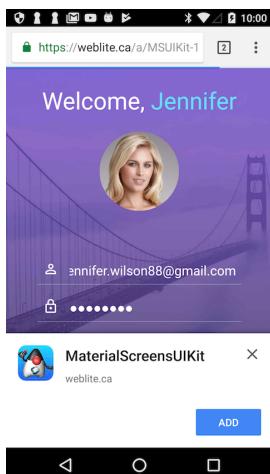


Figure 524. Add app to homescreen banner

If the app is available as a native app, in the Play store, you can indicate this using the `javascript.manifest.related_applications` and `javascript.manifest.prefer_related_applications` build hints. Then, instead of prompting the user to add the web app to their home screen, they'll be prompted to install the native app from the Play store, as shown below.

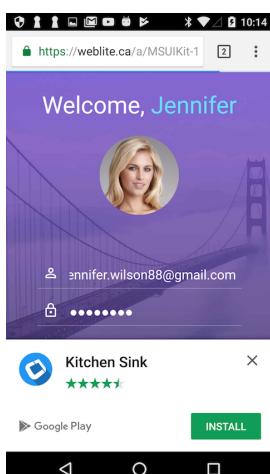


Figure 525. Add native app banner



The PWA standard requires that you host your app on over HTTPS. For testing purposes, it will also work when accessed at a **localhost** address. You can use the [Lighthoust PWA analysis tool](https://developers.google.com/web/ilt/pwa/lighthouse-pwa-analysis-tool) [<https://developers.google.com/web/ilt/pwa/lighthouse-pwa-analysis-tool>] to ensure compliance.

For more information about Progressive Web Apps see [Google's introduction to the subject](https://developers.google.com/web/introduction) [<https://developers.google.com/web/progressive-web-apps/>].

19.12.1. Customizing the App Manifest File

At the heart of a progressive web app is the [web app manifest](https://developer.mozilla.org/en-US/docs/Web/Manifest) [<https://developer.mozilla.org/en-US/docs/Web/Manifest>]. It specifies things like the app's name, icons, description, preferred orientation, display mode (e.g. whether to display browser navigation or to open with the full screen like a native app), associated native apps, etc.. The Codename One build server will automatically generate a manifest file for your app but you can (and should) customize this file via build hints.

Build hints of the form **javascript.manifest.XXX** will be injected into the app manifest. E.g. To set the app's description, you could add the build hint:

```
javascript.manifest.description=An app for doing cool stuff
```

You can find a full list of available manifest keys [here](https://developer.mozilla.org/en-US/docs/Web/Manifest) [<https://developer.mozilla.org/en-US/docs/Web/Manifest>]. The build server will automatically generate all of the icons so you don't need to worry about those. The "name" and "short_name" properties will default to the app's display name, but they can be overridden via the **javascript.manifest.name** and **javascript.manifest.short_name** build hints respectively.



The **javascript.manifest.related_applications** build hint expects a JSON formatted list, just like in the raw manifest file.

19.12.2. Related Applications

One nice feature (discussed above) of progressive web apps, is the ability to specify related applications in the app manifest. Browsers that support the PWA standard use some heuristics to "offer" the user to install the associated native app when it is clear that the user is using the app on a regular basis. Use the **javascript.manifest.related_applications** build hint to specify the location of the native version of your app. E.g.

```
javascript.manifest.related_applications=[{"platform":"play", "id":"my.app.id"}]
```

You can declare that the native app is the preferred way to use the app by setting the **javascript.manifest.prefer_related_applications** build hint to "true".



According to the [app manifest documentation](https://developer.mozilla.org/en-US/docs/Web/Manifest) [<https://developer.mozilla.org/en-US/docs/Web/Manifest>], this should only be used if the related native apps really do offer something that the web application can't do.

19.12.3. Device/Browser Support for PWAs

Chrome and Firefox both support PWAs on desktop and on Android. iOS doesn't support the PWA standard, however, many aspects of it are supported. E.g. On iOS you can add the app to your home screen, after which time it will appear and behave like a native app - and it will continue to work while offline. However, many other nice features of PWA like "Install this app on your home screen" banners, push notifications, and invitations to install the native version of the app, are not supported. It is unclear when, or even, whether Apple will ever add full support; but most experts predict that they will join the rest of the civilized world and add PWA support in the near future.

On the desktop, Chrome provides an analogous feature to "add to your homescreen": "Add to shelf". If it looks like the user is using the app on a regular basis, and it isn't yet installed, it will show a banner at the top of the page asking the user if they want to add to their shelf.

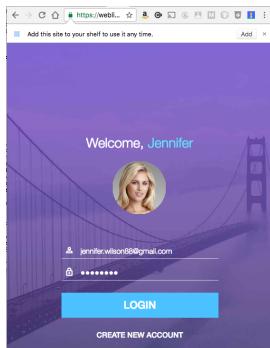


Figure 526. Add to shelf banner

Clicking the "Add button" prompts the user for the name they wish the app to appear as:

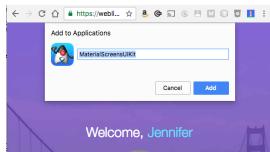


Figure 527. Add to shelf prompt

Upon submission, Chrome will generate a real application (on Mac, it will be a ".app", on Windows, an "exe", etc..) which the user can double click to open the app directly in the Chrome. And, importantly, the app will still work when the user is offline.

The app will also appear in their "Shelf" which you can always access at <chrome://apps>, or by opening the "Chrome App Launcher" app (on OS X this is located in "~/Applications/Chrome Apps/Application Launcher").

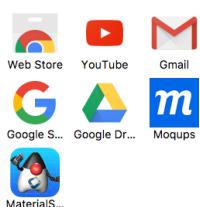


Figure 528. Chrome App Launcher



The Chrome App Launcher lists apps installed both via the Chrome Web Store and via the "Add to Shelf" feature that we discuss here. The features we describe in this article are orthogonal to the Chrome Web Store and will not be affected by its closure.

19.13. Playing Media and Opening Links

People don't like it when the browser automatically starts playing sounds, or opening links without their permission. For this reason, modern browsers generally restrict your ability to programmatically do these things, unless they are in response to a user action, like a mouse click.

If your app needs to play media (e.g. `Media.play()`), or open a link (e.g. `Display.execute("…")`) without the user actually interacting physically (e.g. key press or pointer press), then it will display a popup dialog confirming that the user actually wants to perform this action.

In some cases this dialog may affect the utility of the app. For example, suppose you want to play a video in response to a voice command. Having to press an "OK" button after the command, may be annoying. For such cases, you can use the `platformHint.javascript.backsideHooksInterval` property to **poll** for media play requests on an authorized event.

For example:

```
CN.setProperty("platformHint.javascript.backsideHooksInterval", "1000");

// Now your app will process media.play() and Display.execute(...) calls
// once per second (1000ms). If play() or execute() has been called anytime
// in that second (since the last poll), it will seamlessly process the
// request.

// To disable polling, just set it to an interval 0 or lower.
// e.g.
CN.setProperty("platformHint.javascript.backsideHooksInterval", "0");
```



Do not abuse this feature. You should enable this polling only when necessary. E.g. If your app enables the user to listen for voice commands, only enable polling for the period of time that it is listening. When the user wants to stop listening, you should also stop the polling by setting the interval to "0".

20. Working with UWP

UWP Apps may distributed in 2 different ways:

1. **In the Windows App Store.** (This should be used for deployment of any production app).
2. **Outside of the Windows App Store** via sideloading directly onto a device. This should only be used for development.

20.1. Deploying Outside of the Windows App Store (Sideloading)

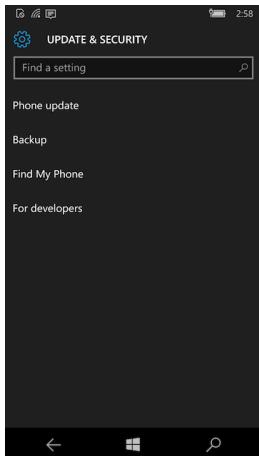
UWP apps may be deployed directly to Windows 10 desktop and mobile devices without any need to involve the Windows Store. This process is only realistic, though, for testing and debugging your app during development because it requires you to enable "Development" mode on the device. In addition, installation is a little bit more complicated than simply downloading an app over the internet. The process for deploying to Windows 10 **desktop** devices is different than the process for **mobile** devices. These are described in the following sections.

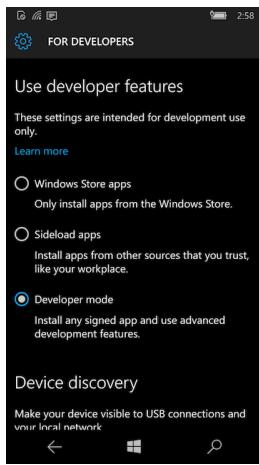
20.1.1. Side-loading to Windows 10 Mobile Devices

Enabling Developer Mode on Device

Before you can side-load apps onto your phone, you'll need to set up your phone for development.

1. In "Settings", select "Update & Security" > "For developers"

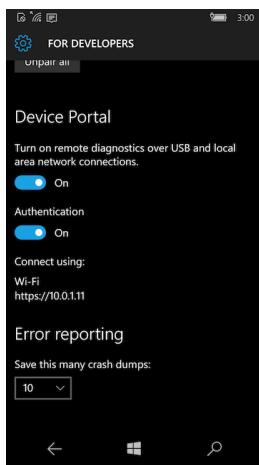




2. Select "Developer mode"
3. Under "Device Discovery", make sure that the "Make your device visible to USB connections and your local network" is set to "On".



4. Make sure that "Device Portal" is set to "On"



5. When you switch "Device Portal" to "On" it should show you an address that you can access the Phone at via wifi. (E.g. <https://10.0.1.11>). **Remember this address**, you're going to use it to install all of your apps onto the device.



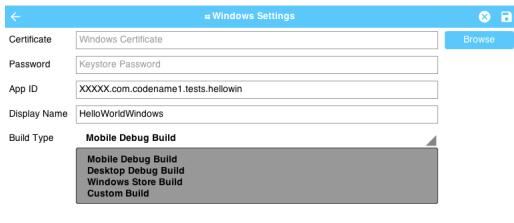
This will be a local address within your local network. It **won't** be available to the outside world.

At this point, your phone should be ready to receive "Side-loaded" apps. This was a one-time setup, so you shouldn't have to do it again, until you set up another device.

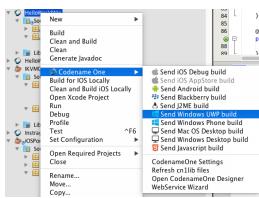
Building App for UWP

Now that your device is set up for development, you can proceed to build your app.

1. Select the "Mobile Debug Build" option in the UWP Codename One Settings.



2. Select the "Send Windows UWP Build" option in the Codename One menu of your IDE. This will initiate the build on the Codename One build server.



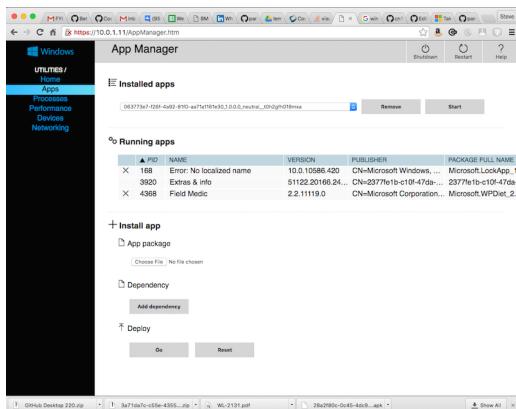
3. Log into the Codename One dashboard to watch the build progress. When it is complete, you'll be able to download the ".appxbundle" file to your desktop.



You cannot simply download the .appxbundle file directly to your Windows Phone 10 mobile device and install it. It will indeed allow you to download it, and will give you an option to install it, but the install will silently fail.

Installing App On Device

1. Point your computer's web browser to the address for your mobile device. (This is the address listed when you turned on the "Device Portal" in the "Enabling Developer Mode on Device" section above. This will open the App Manager page.)
2. Click on the "Apps" item in the left menu.



3. If this is the first time installing a UWP (debug) app on your device, you will need to install the dependencies. You can find the dependencies for mobile/ARM apps [here](https://github.com/codenameone/cn1-binaries/tree/master/uwp/Dependencies/ARM) [https://github.com/codenameone/cn1-binaries/tree/master/uwp/Dependencies/ARM]. You'll need to install both **Microsoft.NET.CoreRuntime.1.0.appx** and **Microsoft.VCLibs.ARM.Debug.14.00.appx**. **If this is not the first time installing a UWP app, you can skip to the next step.**
 - a. Under the "Install App" section, click the "Choose File" button and navigate through the file chooser to select the "Microsoft.NET.CoreRuntime.1.0.appx" file. Then click "Go".
 - b. Do the same for the "Microsoft.VCLibs.ARM.Debug.14.00.appx" file.
4. Under the "Install App" section, click the "Choose File" button and navigate through the file chooser to select the .appxbundle file for your app.
5. Once you have the appxbundle selected, you should press "Go" under the "Deploy" subheading. This will install the app and, if all went well, your app will appear in the "Recently Added" section in the apps list of the phone.

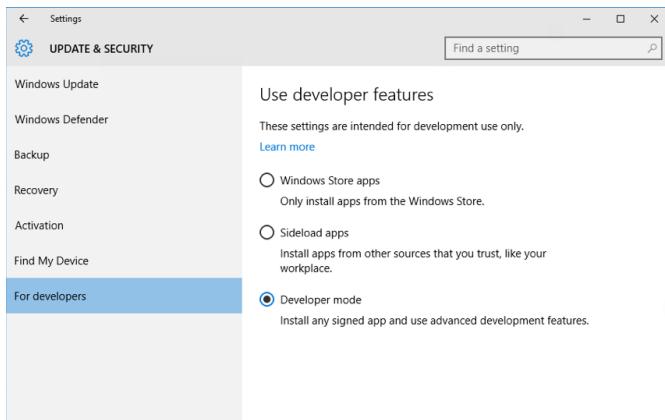


20.1.2. Side-loading to Windows 10 Desktop Devices

Enabling Developer Mode on PC

The easiest way to be able to run your development apps on a Windows 10 PC is to enable developer mode. This will allow you to install any app even if it is just "self-signed".

To enable developer mode, open "Settings", then select "Updates and Security". Under the "For Developers" menu item, select "Developer Mode" as shown below:

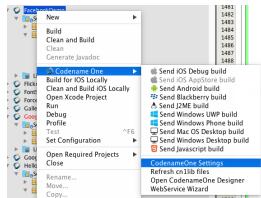


Building the App

Before building the app, you'll need to ensure that the build target is set to "Debug Desktop" in the Codename One Settings panel for Windows apps.

Steps:

1. Open Codename One Settings (steps vary by IDE). On Netbeans you will find "Codename One Settings" by right clicking your project's node in the project explorer, and look in the "Codename One" submenu:



2. Click on "Windows Settings":



3. Under "Build Type", make sure that "Desktop Debug Build" is selected, as shown below:

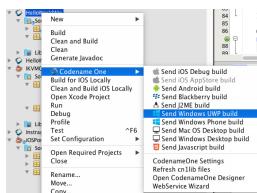


4. Save the changes by clicking the "Disk" icon in the upper right:



Now you can proceed to send the build to the build server.

1. Select the "Send Windows UWP Build" option in the Codename One menu of your IDE. This will initiate the build on the Codename One build server.



- Log into the Codename One dashboard to watch the build progress. When it is complete, you'll be able to download the ".zip" file to the Windows 10 PC on which you wish to install the app.



Installing the App

Start by extracting the .zip file. (Navigate to the folder where the zip was downloaded, right click it, and select "Extract all" as shown below:

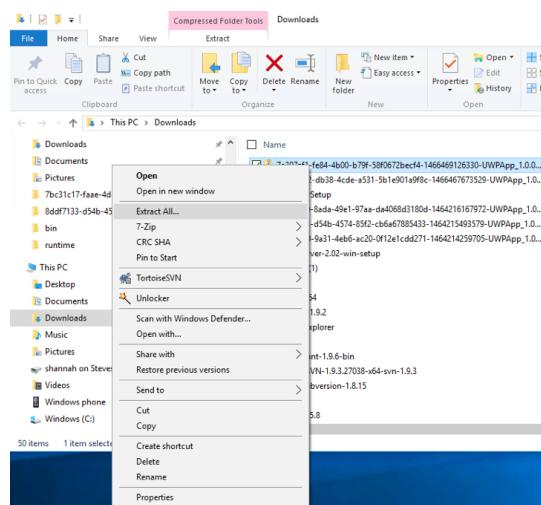


Figure 529. Extract UWP zip file

After extraction, open the resulting directory. You should see contents similar to the following:

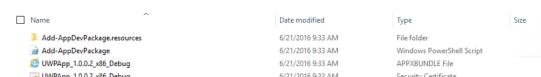


Figure 530. UWP Zip file contents

Downloading Dependencies

If this is your first time installing a UWP app on this PC, you may need to add the dependencies before you can install. You can download the dependencies [here](https://github.com/codenameone/cn1-binaries/raw/master/uwp/Dependencies.zip) [<https://github.com/codenameone/cn1-binaries/raw/master/uwp/Dependencies.zip>]. Extract "Dependencies.zip" and copy the resulting "Dependencies" directory into the app install directory. Your app install directory should now look like:



Figure 531. UWP Zip file contents with dependencies

Running the Powershell Script

We are **finally** at the point where we can run the installer.

Right-click on the "Add-AppDevPackage" icon, and select "Run in Powershell", as shown:

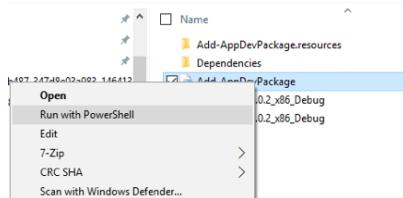


Figure 532. UWP Install by run in powershell

You may be prompted that you need to change the execution policy, in Powershell:

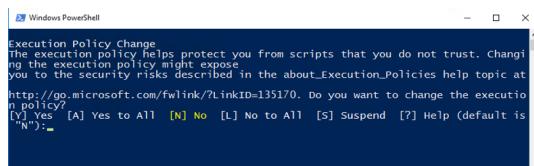


Figure 533. UWP Powershell change execution policy

Enter "Y" at the prompt to allow this.

If all goes well, you should see a message saying that the app was successfully installed.

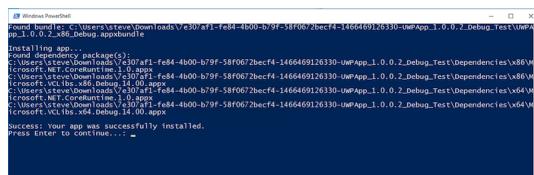


Figure 534. UWP Powershell app successfully installed

And if you look in your "Windows Menu" under "All Apps", you should see your app listed there:

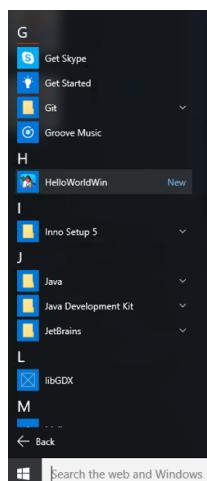


Figure 535. Windows 10 App listed in All Apps

20.1.3. Building for the Windows Store

If you want to be able to distribute your app to the public, the Windows Store is your best channel.

Building for the Windows store involves roughly 3 steps:

1. Reserve a name for your app in the Windows Store
2. Build your app using the "Windows Store Upload" build type.
3. Upload the resulting .appxupload file to the Windows store.

Let's go through these steps in more detail. Start [here](https://developer.microsoft.com/en-us/windows/publish) [https://developer.microsoft.com/en-us/windows/publish].

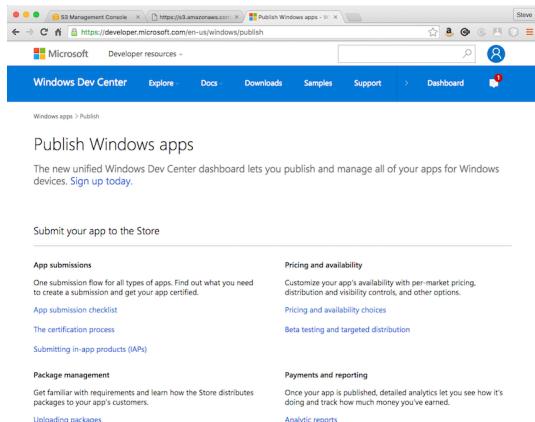


Figure 536. Publish windows apps webpage

If you don't already have an account, sign up for one. Then log in. Once logged in, you can click the "Dashboard" link on the toolbar.

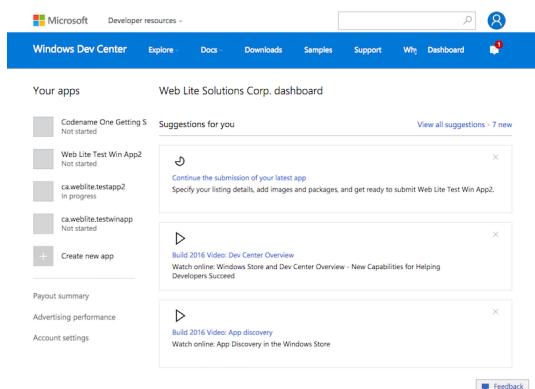


Figure 537. Windows Store dashboard

Under the "Your apps" section (on the left in the above screenshot), click the "Create new app" button.

Create your app by reserving a name

Once you reserve a name, your app will be provisioned for services like push notifications and you can start defining IAPs (in-app products).

Make sure you have the rights to use any name you reserve. You must submit this app to the Store within one year, or you'll lose your name reservation. [Learn more](#)

A screenshot of a "Reserve app name" form. A text input field contains "My First CN1 App" with a green checkmark icon to its right. Below the input field are two buttons: "Reserve app name" (highlighted in blue) and "Cancel".

Figure 538. Reserve name form

Enter a name for your app, and click "Reserve app name".

If the name was available, it should take you to the app overview page for your new app. There's quite a few options there to play with, but we're not going to worry about any of them for now. All we need to know is:

1. Your App's ID
2. Your App's Publisher ID

You can get this information by scrolling down to the bottom of the "App overview" page and clicking the "View app identity details" under the "App Management" > "App identity" section:

The screenshot shows the "App management" section with the "App identity" tab selected. It displays two main sections: "App identity" and "App names". The "App identity" section contains a note about unique identities and a link to "View app identity details". The "App names" section allows reserving names for other languages or changing the app's name, with a link to "Manage app names".

Figure 539. App Identity section

You'll see a page with the information we need shown below:

The screenshot shows the "App identity" details page. It includes a note about including identity details in the package manifest and a table with three rows of data:

Package/Identity/Name	CCC233E.MyFirstCN1App
Package/Identity/Publisher	CN=E88E4AF1-6731-4B5F-8629-B5488FFD73C4
Package/Properties/PublisherDisplayName	Web Lite Solutions Corp.

Figure 540. App identity details

The next step is to copy this information into your Codename One project.

Open up the Codename One Settings for your project and go to the "Windows Settings" section.

Copy and paste the "Package/Identity/Name" and "Package/Properties/PublisherDisplayName" values from the windows store into the "App ID" and "Publisher Display Name" fields respectively.



It is important that your App ID and Publisher Display Name match exactly what you have in the store, or your app will fail at the validation stage when you try to upload your app to the store.

Next, click on the "Generate" button next to the Certificate field.

This will open a dialog titled "Certificate Generator". Paste the value from the "Package/Identity/Publisher" listed in the Windows Store into the Publisher ID field as shown below:



Figure 541. Certificate Generator

Then click OK. This will generate a .pfx file inside your project folder.

The "Display Name" must also match that app name in the store.

Finally, make sure that "Windows Store Upload" is selected in the "Build Type" field. For the example above, my settings form looks like the following screenshot when I am done.



Figure 542. Settings for Windows Store uploads

When you are done, hit the "Save" icon in the upper right corner of the window to save your changes.

Finally, select "Codename One" > "Send Windows UWP Build" in your IDE's project explorer.

This will produce an .appxupload file that you can upload to the Windows Store.

See the [Microsoft's documentation on uploading app packages](https://msdn.microsoft.com/en-us/windows/uwp/publish/upload-app-packages) [<https://msdn.microsoft.com/en-us/windows/uwp/publish/upload-app-packages>] for more information on the remaining steps.

20.2. Debugging UWP Apps

On most platforms, there is a device log that records errors, exceptions, and messages written to STDOUT. UWP, unfortunately, doesn't seem to provide this. If you are running on Windows Phone 10, there doesn't seem to be any device log at all. There is a separate program called "Field Medic" that you can use to do some logging, but it doesn't capture application errors or STDOUT messages.

The best way to debug apps on device is to enable crash protection in your app. This can be enabled by adding the following to your app's `init()` method:

```
Log.bindCrashProtection(true);
```

With crash protection enabled, you'll receive an email whenever an exception is thrown that isn't caught in your application code. The email will include the stack trace of the error along with any output you had previously provided using the `com.codename1.io.Log` class (e.g. `Log.p()` and `Log.e()`).



A Pro account (or higher) is required to receive crash protection emails.

20.2.1. No Line Numbers in Stack Traces

One major annoyance of UWP is that it doesn't provide line numbers in its stack traces. Here is what you can expect to see in a stack trace:

```

[EDT] 0:0:7,683 - Results null
[EDT] 0:0:7,799 - Exception: java.lang.NullPointerException - null
    at com.codename1.ui.Display.invokeAndBlock(Runnable r, Boolean dropEvents)[EDT] 0:0:7,815 - Exception in AppName
version 1.1
[EDT] 0:0:7,830 - OS win
[EDT] 0:0:7,836 - Error java.lang.NullPointerException
[EDT] 0:0:7,836 - Current Form null
[EDT] 0:0:7,836 - Exception: java.lang.NullPointerException - null
at System.Environment.GetStackTrace(Exception e, Boolean needFileInfo)
    at System.Environment.get_StackTrace()
    at UWPApp.IKVMReflectionHelper.getCurrentStackTrace()
    at java.lang.ThrowableHelper.getCurrentStackTrace()
    at java.lang.Throwable..ctor()
    at java.lang.Exception..ctor()
    at java.lang.RuntimeException..ctor()
    at java.lang.NullPointerException..ctor()
    at java.lang.Throwable.___mapImpl(Exception )
    at IKVM.Internal.ExceptionHelper.MapException[T](Exception x, Boolean remap, Boolean unused)
    at IKVM.Runtime.ByteCodeHelper.MapException[T](Exception x, MapFlags mode)
    at com.codename1.ui.Display.invokeAndBlock(Runnable r, Boolean dropEvents)
    at com.codename1.ui.Display.invokeAndBlock(Runnable r)
    at com.codename1.impl.SilverlightImplementation.editString(Component n1, Int32 n2, Int32 n3, String n4, Int32 n5)
    at com.codename1.impl.CodenameOneImplementation.editStringImpl(Component cmp, Int32 maxSize, Int32 constraint, String
text, Int32 initiatingKeyCode)
    at com.codename1.ui.Display.editString(Component cmp, Int32 maxSize, Int32 constraint, String text, Int32
initiatingKeyCode)
        at com.codename1.ui.Display.editString(Component cmp, Int32 maxSize, Int32 constraint, String text)
        at com.codename1.ui.TextArea.editString()
        at com.codename1.ui.TextArea.pointerReleased(Int32 x, Int32 y)
        at com.codename1.ui.TextField.pointerReleased(Int32 x, Int32 y)
        at com.codename1.ui.Form.pointerReleased(Int32 x, Int32 y)
        at com.codename1.ui.Component.pointerReleased(Int32[] x, Int32[] y)
        at com.codename1.ui.Display.handleEvent(Int32 offset)
        at com.codename1.ui.Display.edtLoopImpl()
        at com.codename1.ui.Display.mainEDTLoop()
        at com.codename1.ui.RunnableWrapper.run()
        at com.codename1.impl.CodenameOneThread.run()
        at java.lang.Thread.threadProc2()
        at java.lang.Thread.threadProc()
        at java.lang.Thread$1.Invoke()
        at com.codename1.impl.NativeThreadImpl.<>c$_DisplayClass6_0.<init>b__0()
        at System.Threading.Tasks.Task.InnerInvoke()
        at System.Threading.Tasks.Task.Execute()
        at System.Threading.Tasks.Task.ExecutionContextCallback(Object obj)
        at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
        at System.Threading.Tasks.Task.ExecuteWithThreadLocal(Task& currentTaskSlot)
        at System.Threading.Tasks.Task.ExecuteEntry(Boolean bPreventDoubleExecution)
        at System.Threading.Tasks.ThreadPoolTaskScheduler.LongRunningThreadWork(Object obj)
        at System.Threading.ThreadHelper.ThreadStart_Context(Object state)
        at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
        at System.Threading.ThreadHelper.ThreadStart(Object obj)

Originating from:
Message=Object reference not set to an instance of an object.
    at com.propertycross.codename1.PropertyCross$1.run()
    at com.codename1.ui.Display.processSerialCalls()
    at com.codename1.ui.Display.edtLoopImpl()
    at com.codename1.ui.Display.invokeAndBlock(Runnable r, Boolean dropEvents)

```

It will show you the call stack with the names of the methods. But it won't show you the line numbers. If the stack trace isn't specific enough, you can add `Log.p()` statements in various positions in my code to help narrow down the source of the exception.

20.3. Customizing the Status Bar

On mobile, the status bar (the bar across the top of the screen with the time, battery level etc...) is updated each time a form is shown. The foreground color, background color, and background opacity are set using the unselected style of the form being shown.

You can override these colors application-wide using the following display properties:

1. `windows.StatusBar.ForegroundColor` - A string representation of an integer RGB color.
2. `windows.StatusBar.BackgroundColor` - A string representation of an integer RGB color.
3. `windows.StatusBar.BackgroundTransparency` - A string representation of 0-255 integer.

e.g.

```
Display d = Display.getInstance();
d.setProperty("windows.StatusBar.ForegroundColor", String.valueOf(0xff0000)); // red
d.setProperty("windows.StatusBar.BackgroundColor", String.valueOf(0xffffffff)); // white
d.setProperty("windows.StatusBar.BackgroundOpacity", String.valueOf(255)); // fully opaque
```

20.4. Associating App with File Types

It is possible to associate your application with file types on UWP using the standard Codename One "AppArg" method in conjunction with the `windows.extensions` build hint. Any content you place in the `windows.extensions` build hint will be embedded inside the `<Extensions/>` section of the Package.appxmanifest file. Then if the app is opened as a result of the user opening a file of the specified type, then the path to that file will be made available to the app via `Display.getProperty("AppArg")`.

Example `windows.extensions` Value:

The following value would associate the app with the file extension ".alsdk". This example is taken from [this MSDN document](#) [<https://msdn.microsoft.com/vi-vn/windows/uwp/launch-resume/handle-file-activation?f=255&MSPPError=-2147217396>].

```
<uap:Extension Category="windows.fileTypeAssociation">
  <uap:FileTypeAssociation Name="alsdk">
    <uap:Logo>images\icon.png</uap:Logo>
    <uap:SupportedFileTypes>
      <uap:FileType>.alsdk</uap:FileType>
    </uap:SupportedFileTypes>
  </uap:FileTypeAssociation>
</uap:Extension>
```

To register your app to be able to handle PDFs, you would add:

```
<uap:Extension Category="windows.fileTypeAssociation">
  <uap:FileTypeAssociation Name="pdf">
    <uap:Logo>images\icon.png</uap:Logo>
    <uap:SupportedFileTypes>
      <uap:FileType ContentType="application/pdf">.pdf</uap:FileType>
    </uap:SupportedFileTypes>
  </uap:FileTypeAssociation>
</uap:Extension>
```

For more information about using the "AppArg" property, see [this blog post](#) [<https://www.codenameone.com/blog/intercepting-urls-on-ios-andriod.html>] which describes its usage on iOS and Android for intercepting URL types.

21. Working with Mac OS X

21.1. Mac OS Desktop Build Options

You can configure Desktop Mac OS build settings, by opening Codename One Settings, and clicking the "Mac Desktop Settings" button:



Figure 543. Mac Desktop settings

This will bring you to the following form:

Figure 544. Mac Desktop settings form

Here you can provide your certificate(s) as a .p12 file, and select a bundle type.

21.1.1. Bundle Types

There are three bundle types which dictate what the build server produces for you when you build your project as a Desktop Mac OS App.

1. **DMG** - Produces a .DMG disk image with your app. This is the preferred format for distributing your app outside of the Mac Appstore. If you provide a Developer ID Application certificate (see "Understanding Certificates" below), this the app will be signed so that users won't receive warnings about "Unidentified developer" when they install your app.
2. **Sandboxed DMG** - Same as **DMG** bundle type except that your app is set up to use the app sandbox. Generally this would be used to test an app that is being distributed via the Appstore, since Appstore apps **must** use the sandbox. If you select this bundle type, you are **required** to provide a Mac App Distribution Certificate, and you should additionally specify entitlements required for your app to function properly. For more information about the app sandbox, see [Apple's documentation on the subject](https://developer.apple.com/app-sandboxing/) [<https://developer.apple.com/app-sandboxing/>].
3. **Mac Appstore Upload (PKG)** - Produces a .PKG file that you can upload to the Mac appstore. This requires that you provide **both** a Mac App Distribution certificate, and a Mac App Installer certificate (see "Understanding Certificates" below). Both of these certificates should be embedded into a single .p12 file (See "Exporting Certificates as p12" below).

21.1.2. Understanding Mac Certificates

For the purposes of Mac application distribution, there are 3 types of certificates that we will be interested in. The type(s) of certificate required will depend on the type of bundle you generate. The certificate types are:

1. **Developer ID Application Certificate (Mac applications)**

This type of certificate is used to sign an app to be distributed **outside** of the Mac Appstore as a DMG image. This corresponds to the "DMG" bundle type in Codename one settings. You can easily identify this kind of certificate because its identity will be of the form "Developer ID Application: YOUR COMPANY NAME (SOME CODE)". E.g. Developer ID Application: Acme Widgets Corp. (XYSD5YF).

2. Mac App Distribution Certificate (Mac Appstore)

This type of certificate is used to sign the .app bundle for an app that is to be distributed in the Mac Appstore. This certificate is required for both the "Sandboxed DMG", and "Mac Appstore Upload (PKG)" bundle types. You can easily identify this kind of certificate because its identity will be of the form "3rd Party Mac Developer Application: YOUR COMPANY NAME (SOME CODE)". E.g. 3rd Party Mac Developer Application: Acme Widgets Corp. (XYSD5YF).

3. Mac App Installer Certificate (Mac Appstore)

This type of certificate is used to sign the .pkg installer for an app that is being submitted to the Mac Appstore. This certificate is required for the "Mac Appstore Upload (PKG)" bundle type only. You can easily identify this kind of certificate because its identity will be of the form "3rd Party Mac Developer Installer: YOUR COMPANY NAME (SOME CODE)". E.g. 3rd Party Mac Developer Installer: Acme Widgets Corp. (XYXD5YF).

21.1.3. Obtaining Certificates

 Apple provides documentation on obtaining certificates [on its website](https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html#/apple_ref/doc/uid/TP40012582-CH31-SW6) [https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html#/apple_ref/doc/uid/TP40012582-CH31-SW6] but the process described there involves Xcode. This section describes an alternate process that doesn't require Xcode.

If you have an Apple developer account, you can manage your certificates [here](https://developer.apple.com/account/mac/certificate) [<https://developer.apple.com/account/mac/certificate>].

Mac Certificates		
Name	Type	Expires
Web Lite Solutions Corp.	Developer ID Application	Jan 07, 2021
Web Lite Solutions Corp.	Developer ID Application	Sep 29, 2021
Web Lite Solutions Corp.	Mac App Distribution	Oct 23, 2018
Web Lite Solutions Corp.	Mac Installer Distribution	Oct 23, 2018

Figure 545. Mac Developer portal certificates

The screenshot above shows an account that already has the three kinds of certificates we will require:

1. **Developer ID Application** - Used for the DMG bundle type.
2. **Mac App Distribution** - Used for the Sandboxed DMG and Mac Appstore Upload (PKG) bundle types.
3. **Mac App Installer** - Used for the Mac Appstore Upload (PKG) bundle type.

If your account doesn't yet have a certificate of the required type, you should begin by pressing the

"+" button in the upper right. This will bring you to a page asking "What type of Certificate do you need?". There are only two options on this page that we'll be interested in:

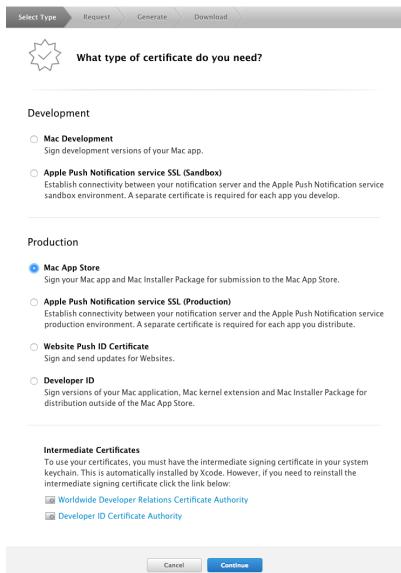


Figure 546. Creating a new certificate

1. **Production > Mac App Store** - For both the Mac App Distribution and Mac App Installer certificates.
2. **Production > Developer ID** - For the Developer ID Application certificate.

Select the option corresponding to the certificate you wish to generate. In either case, you'll be taken to a form to select whether you want an "Installer" certificate or an "Application" certificate. Select the appropriate type.

You will then be prompted to upload a Certificate Signing Request (CSR) file, and it will provide instructions on how to do this via the Keychain app.



You can reuse the same CSR file for generating all 3 certificates.

After generating the certificates, you should download them to your Mac, and import them into your keychain. You should be able to accomplish this by simply double-clicking the downloaded ".cer" file, and following the prompts.

21.1.4. Exporting Certificates as P12



The following section requires access to a Mac, and assumes that you have already generated your 3 certificates

Notice that Mac apps may require three different kinds of certificates, yet the settings page only provides space for a single certificate (P12) file. This is not a mistake. P12 files may contain more than one certificate, and you are expected to include all of the certificates that the build server may require inside a single P12. The build server will automatically extract the certificates it needs according to the bundle type.

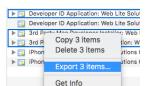
When building the "DMG" bundle type, the build server will look for a "Developer ID Application

"Certificate" inside the P12. If one is found, it will be used to sign the app bundle.

The "Sandboxed DMG" target will look for a "Mac App Distribution Certificate" certificate in the P12.

The "Mac Appstore Upload (PKG)" target will require both a "Mac App Distribution Certificate" and a "Mac App Installer Certificate" to be included in the P12.

The easiest way to produce a P12 that includes all 3 kinds of certificates is to export them from the Keychain Access app (Requires a Mac). Select all 3 certificates at once (using CMD-click), then right click and select "Export 3 Items..."



You will then be prompted to select a location to save the .p12 file, as well as selecting a password for the file.

21.1.5. Entitlements

When distributing apps in the Mac Appstore, or when using the "Sandboxed DMG" bundle type, your app is run inside a sandboxed environment, meaning that it doesn't have access to the outside world. It is provided its own "sandboxed" container for file system access, and it doesn't get any network access. If your app requires access to the "outside world", you need to request entitlements for that access. If you select a bundle type that uses the sandbox, you will be shown a list of all of the available entitlements from which you can "check" the ones that you wish to include.

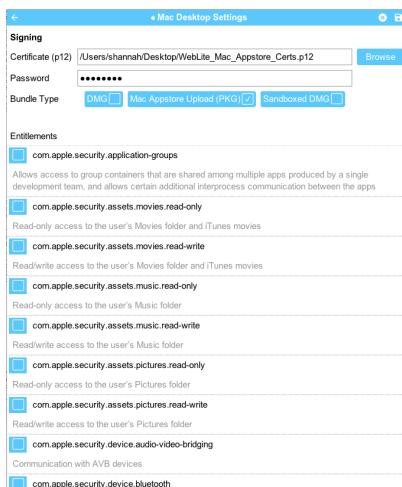


Figure 547. App sandbox entitlements

For more information about the app sandbox, and a full list of entitlements, see [Apple's documentation on the subject](https://developer.apple.com/library/content/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html) [<https://developer.apple.com/library/content/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html>].

22. Security

Security is a "big word". It implies many things and that is also true for the mobile app development so before we get started lets try to define the scope of security.

We will deal only with application security and its communication mechanisms while ignoring everything beyond that scope. Let's start with a simple fact:

Codename One applications are secure on the devices by the nature of the mobile OS security.

For most intents and purposes this will be enough, unless you are specifically concerned about security this section isn't for you. Mobile OS's isolate applications from one another so it's hard for an application to damage the OS or even damage/spy on a different app.

The restrictions laid on apps are here to make them extra secure and on top of that Codename One lays a few big advantages in terms of security:

- Codename One code is compiled (unlike e.g. PhoneGap/Cordova)
- We obfuscate by default which makes the binaries harder to reverse engineer
- We compile the UI to native code too which means typical reverse engineering code will have a harder time following
- We disable debug flags so a hacker won't be able to debug your production app on the device

Despite that you still need to keep in mind that the binary could still be reverse engineered and so it is important to avoid storing keys in the client side code. E.g. if you have an API key to access a service (e.g. Google Cloud key) it needs to be stored in your server and not as a constant in your app!

Each section below discusses some attack vectors against applications and how they can be stopped. Pretty much all of these attacks require a very sophisticated attacker which will only exist for high value targets (e.g. bank apps, government etc.).

22.1. Constant Obfuscation

One of the first things a hacker will do when compromising an app is look at it. E.g. if I want to exploit a bank's login UI I would look at the label next to the login and then search for it in the decompiled code. So if the UI has the String "enter user name and password" I can search for that.

It won't lead directly to a hack or exploit but it will show you the approximate area of the code where we should look and it makes the first step that much easier. Obfuscation helps as it removes descriptive method names but it can't hide the Strings we use in constants. If an app has a secret key within obfuscating it can make a difference (albeit a slight difference).

Notice that this is a temporary roadblock as any savvy hacker would compile the app and connect a debugger eventually (although this is blocked in release builds) and would be able to inspect values of variables/flow. But the road to reverse engineering the app would be harder even with a simple

xor obfuscation.



We're not calling this encoding or encryption since it's neither. It's a simple obfuscation of the data

There are two simple methods in the `Util` class:

```
public static String xorDecode(String s);
public static String xorEncode(String s);
```

They use a simple xor based obfuscation to make a String less readable. E.g. if you have code like this:

```
private static final String SECRET = "Don't let anyone see this....";
```

You might be concerned about the secret, then this would make it slightly harder to find out:

```
// Don't let anyone see this....
private static final String SECRET = Util.xorDecode("RW1tI3Ema219KmpidGFhdTFhdnE1Yn9xajQ1MjM=");
```

Notice that this is **not secure**, if you have a crucial value that must not be found you need to store it in the server. There is no alternative as everything that is sent to the client can be compromised by a determined hacker



Use the comment to help you find the string in the code

Our builtin user specific constants are obfuscated with this method, e.g. normally an app built with Codename One carries some internal data such as the user who built the app etc. This is obfuscated now. We built this small app to encode strings easily so we can copy and paste them into our app easily:

```
Form hi = new Form("Encoder", BoxLayout.y());
TextField bla = new TextField("", "Type Text Here", 20, TextArea.ANY);
TextArea encoded = new TextArea();
SpanLabel decoded = new SpanLabel();
hi.addAll(bla, encoded, decoded);
bla.addDataChangedListener((a, b) -> {
    String s = bla.getText();
    String e = Util.xorEncode(s);
    encoded.setText(e);
    decoded.setText(Util.xorDecode(e));
    hi.getContentPane().animateLayout(100);
});

hi.show();
```

This allows you to type in the first text field and the second text area shows the encoded result. We used a text area so copy/paste would be easy.

For your convenience this app can be accessed here: <https://www.codenameone.com/demos/StringEncoder/index.html>

22.2. Storage Encryption

Codename One had support for bouncy castle encryption for quite a while but it's not as intuitive as we'd like it to be. This makes securing/encrypting your app more painful than it should.

Codename One supports full encryption of the `Storage` (notice the distinction, `Storage` is not `FileSystemStorage`). This is available by installing the bouncy castle cn1lib from the extensions menu then using one line of code

```
EncryptedStorage.install("your-pass-encryption-key");
```



Normally you would want that code within your `init(Object)` method

Notice that you can't use storage or preferences to store this data as it would be encrypted (`Preferences` uses `Storage` internally). You can use a password for this key and it would make it way more secure **but** if a user changes his password you might have a problem. In that case you might need the old password to migrate to a new password.

This works thru a new mechanism in storage where you can replace the storage instance with another instance using:

```
Storage.setStorageInstance(new MyCustomStorageSubclass());
```

We can leverage that knowledge to change the encryption password on the encryption storage using pseudo code like this:

```
EncryptedStorage.install(oldKey);
InputStream is = Storage.getInstance().createInputStream(storageFileName);
byte[] data = Util.readInputStream(is);
EncryptedStorage.install(newKey);
OutputStream o = Storage.getInstance().createOutputStream("TestEncryption");
o.write(data);
o.close();
```



It's not a good idea to replace storage objects when an app is running so this is purely for this special case...

If you use preferences it might be a good idea to set their builtin location to a different path using something like `Preferences.setPreferencesLocation("EncryptedPreferences");`.

This is useful as it prevents the encrypted preferences from colliding with the regular preferences.

22.3. Disabling Screenshots

One of the common security features some apps expect is the ability to block a screenshot. In the past apps like snapchat required that you touch the screen to view a photo to block the ability to grab a screenshot (on iOS). This no longer works...

Blocking screenshots is an Android specific feature that can't be implemented on iOS. This is implemented by classifying the app window as secure and you can do that via the build hint `android.disableScreenshots=true`. Once that is added screenshots should no longer work for the app, this might impact other things as well such as the task view which will no longer show the screenshot either.

22.4. Blocking Copy & Paste

Blocking copy & paste is useful for cases where a device might have spyware installed that monitors the clipboard. This also prevents a user from using a password manager (which usually rely on the clipboard), those managers could be compromised and thus if you are building a very secure app this might be necessary.

You can block copy & paste on Android & iOS. Blocking of copy & paste can be implemented globally or on a specific field.

To block copy & paste globally use:

```
Display.getInstance().setProperty("blockCopyPaste", "true");
```

To block copy & paste on a specific field do:

```
textCmp.putClientProperty("blockCopyPaste", Boolean.TRUE);
```



Notice that the inverse of using `false` might not work as expected

22.5. Blocking Jailbreak

iOS & Android are walled gardens which is both a blessing and a curse. Looking at the bright side the walled garden aspect of locked down devices means the devices are more secure by nature. E.g. on a PC that was compromised we can detect the banking details of a user logging into a bank. But on a phone it would be much harder due to the deep process isolation.

This isn't true for jailbroken or rooted devices. In these devices security has been compromised often with good intentions (opening up the ecosystem) but it can also be used as a step in a serious attack on an application!

For obvious reasons it's really hard to accurately detect a jailbroken or rooted device but when

possible if you have a high security app you might want to block the functionality or even raise a "silent alarm" in such a case. To detect this you can use the `isJailbrokenDevice` method as such:

```
if(Display.getInstance().isJailbrokenDevice()) {  
    // probably jailbroken or rooted  
} else {  
    // probably not  
}
```

Notice that this isn't accurate, we can't be 100% sure as there are no official ways to detect jailbreak. That is why it's crucial to encrypt everything and assume the device was compromised to begin with when dealing with very sensitive data. Still it's worthwhile to use these API's to make the life of an attacker just a little bit harder.

22.6. Strong Android Certificates

When Android launched RSA1024 with SHA1 was considered strong enough for the foreseeable future, this hasn't changed completely but the recommendation today is to use stronger cyphers for signing & encrypting as those can be compromised.

APK's are signed as part of the build process when we upload an app to the Google Play Store. This process seems redundant as we generate the signature/certificate ourselves (unlike Apple which generates it for us). However, this is a crucial step as it allows the device to verify upgrades and make sure a new update is from the same original author!

This means that if a hacker takes over your account on Google Play, he still won't be able to ship fake updates to your apps without your certificate. That's important since if a hacker would have access to your certificate he could create an app update that would just send him all the users private information e.g. if you are a bank this could be a disaster.

Android launched with RSA1024/SHA1 as the signing certificates. This was good enough at the time and is still pretty secure. However, these algorithms are slowly eroding and it is conceivable that within the 10-15 year lifetime of an app they might be compromised using powerful hardware. That is why Google introduced support for stronger cryptographic signing into newer versions of Android and you can use that.

22.6.1. The Bad News

There is a downside...

Google only introduced that capability in Android 4.3 so using these new keys will break compatibility with older devices. If you are building a highly secure app this is probably a tradeoff you should accept. If not this might not be worth it for some theoretical benefit.

Furthermore, if your app is already shipping you are out of luck. Due to the obvious security implications once you shipped an app the certificate is final. Google doesn't provide a way to update the certificate of a shipping app. Thus this feature only applies to apps that aren't yet in the play store.

22.6.2. The Good

If you are building a new app this is pretty easy to integrate and requires no changes on your part. Just a new certificate. You can generate the new secure key using instructions in articles like [this one](#) [<https://guardianproject.info/2015/12/29/how-to-migrate-your-android-apps-signing-key/>].

If you are using **Codename One Setting** you can check the box to generate an SHA512 key which will harden the security for the APK.

22.7. Certificate Pinning

When we connect to HTTPS servers our networking code checks the certificate on the server. If the certificate was issued by a trusted certificate authority then the connection goes thru otherwise it fails. Let's imagine a case where I'm sitting in a coffee shop connected to the local wifi, I try to connect to gmail to check my email. Since I use HTTPS to Google I trust my connection is secure.



What if the coffee shop was hacked and the router is listening in on everything?



What if the router grabs the servers certificate and communicates with Google in my name?

This won't work since the data we send to the server is encrypted with the certificate from the server.



So what if the router sends its own "fake certificate"?

That won't work either. All certificates are signed by a "certificate authority" indicating that a google.com certificate is valid.



What if I was able to get my fake certificate authorized by a real certificate authority?

That's a problem!

It's obviously hard to do but if someone was able to do this he could execute a "man in the middle" attack as described above. People were able to fool certificate authorities in the past and gain fake certificates using various methods so this is possible and probably doable for any government level attacker.

22.7.1. Certificate Pinning

This is the attack certificate pinning (or SSL pinning) aims to prevent. We code into our app the "fingerprint" of the certificate that is "good" and thus prevent the app from working when the certificate is changed. This might break the app if we replace the certificate at some point but that might be reasonable in such a case.

To do this we have a [cn1lib](https://github.com/codenameone/SSLCertificateFingerprint/) [https://github.com/codenameone/SSLCertificateFingerprint/]. that fetches the certificate fingerprint from the server, we can just check this fingerprint against a list of "authorized" keys to decide whether it is valid. You can install the [SSLCertificateFingerprint](#) from the extensions section in [Codename One Settings](#) and use something like this to verify your server:

```
if(CheckCert.isCertCheckingSupported()) {  
    String f = CheckCert.getFingerprint(myHttpsURL);  
    if(validKeysList.contains(f)) {  
        // OK it's a good certificate proceed  
    } else {  
        if(Dialog.show("Security Warning", "WARNING: it is possible your communications are being tampered! We suggest quitting the app at once!", "Quit", "Continue")) {  
            Display.getInstance().exitApplication();  
        }  
    }  
} else {  
    // certificate fingerprint checking isn't supported on this platform... It's your decision whether to proceed or not  
}
```

Notice that once connection is established you don't need to verify again for the current application run.

23. Travis CI Integration

This page includes instructions on setting up continuous integration for Codename One applications using GitHub and Travis-CI.

Travis CI is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub.

Open source projects may be tested at no charge via travis-ci.org. Private projects may be tested at travis-ci.com on a fee basis. TravisPro provides custom deployments of a proprietary version on the customer's own hardware.

Although the source is technically free software and available piecemeal on GitHub under permissive licenses, the company notes that it is unlikely that casual users could successfully integrate it on their own platforms.

— Wikipedia

Codename One provides single-click integration with Travis-CI via the Codename One Settings tool. Once integration is set up, Travis will automatically build your project and run unit tests following each commit of your project to GitHub.

23.1. Quick Start

Assuming you already have a Codename One project, you only need to do three things to enable Travis:

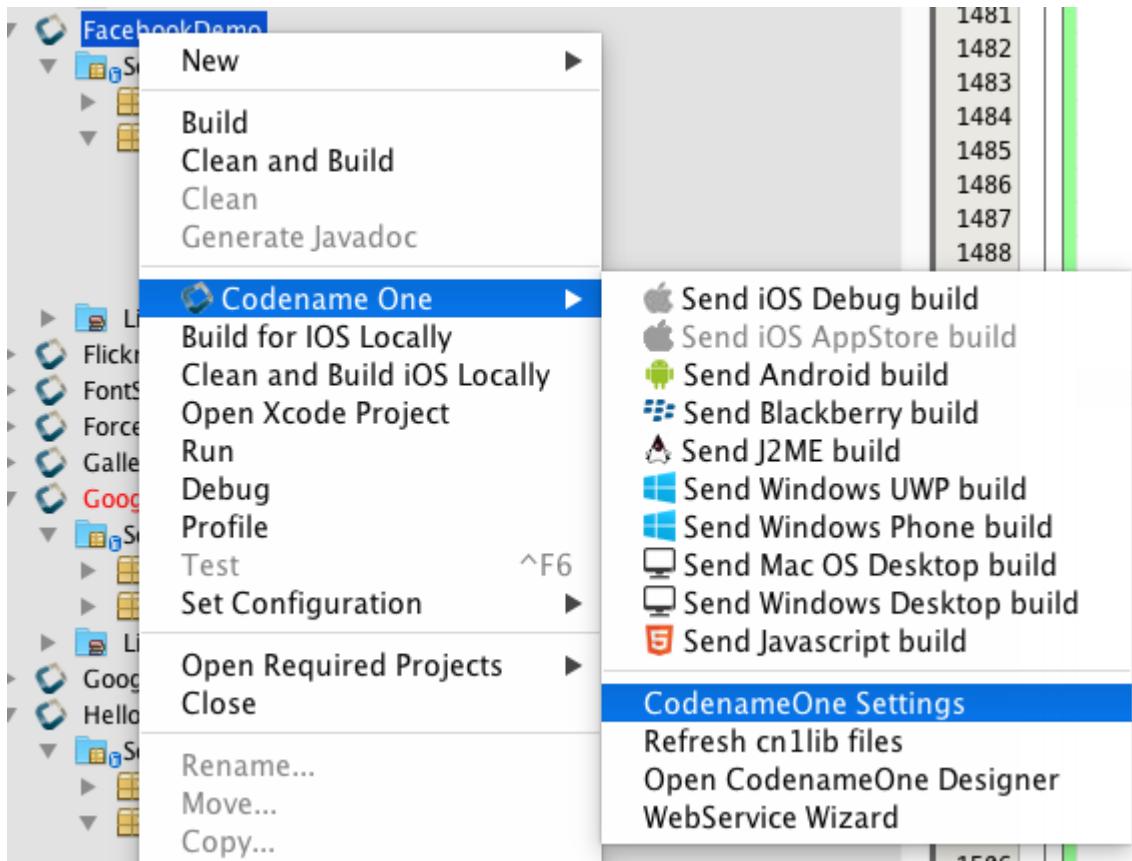
1. Enable Travis in Codename One Settings
2. Push the project to GitHub.
3. Turn on your repository on travis-ci.org [<https://travis-ci.org>] if your repository is public or travis-ci.com [<https://travis-ci.com>] if your repository is private.



Travis CI is free for public GitHub repositories. If your repository is private, then you'll need a travis-ci.com [<https://travis-ci.com/>] account. See their [plans and pricing here](https://travis-ci.com/plans) [<https://travis-ci.com/plans>].

23.1.1. Enabling Travis

Open Codename One settings by right clicking the project, and selecting "Codename One" > Codename One Settings"



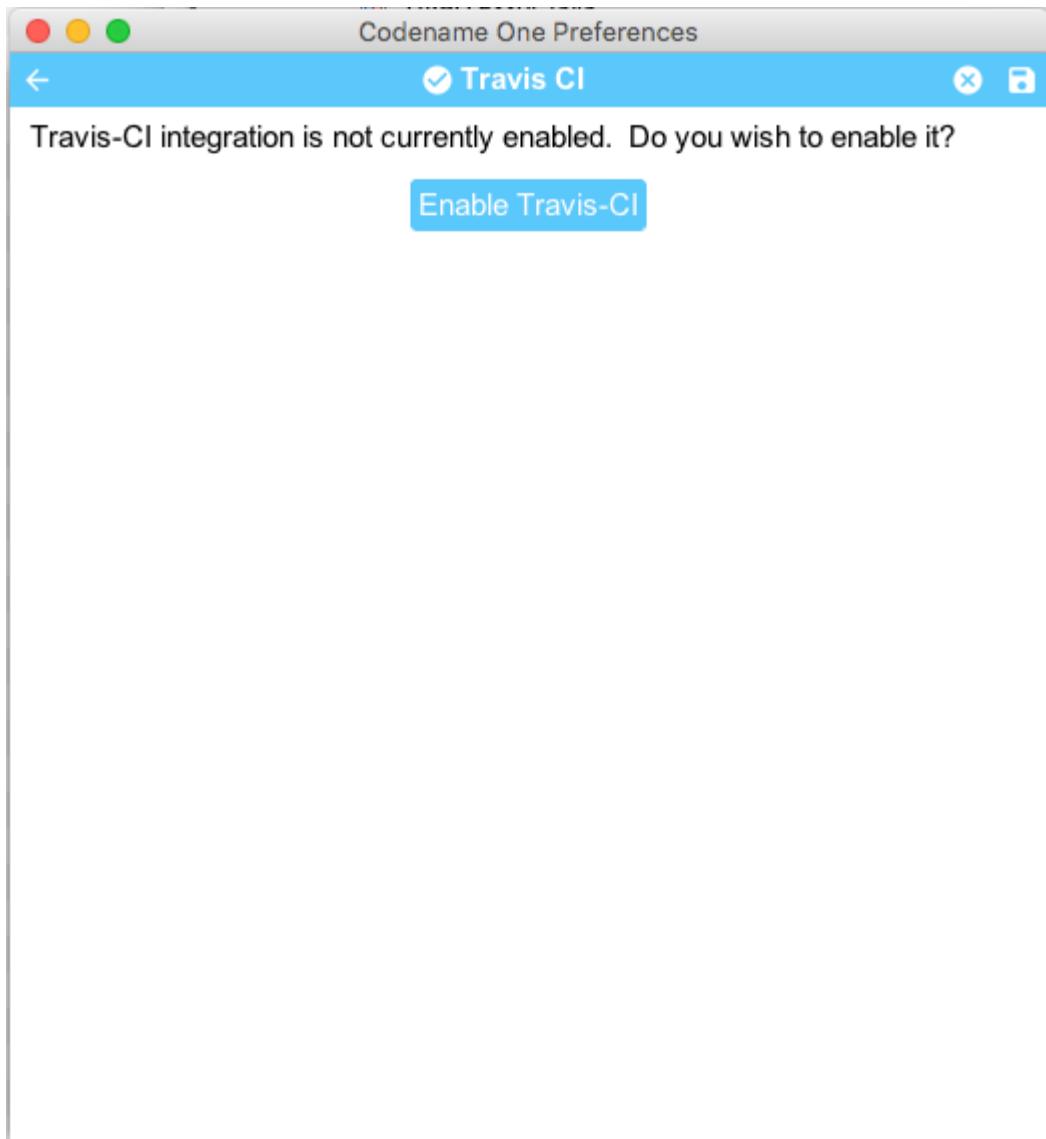
Then click on the "Travis CI Integration" button



Travis CI

[Configure project for continuous integration with Travis-CI](#)

If your project isn't already configured for travis, you'll see a form as follows:



Click the "Enable Travis CI" button.

This will install some travis scripts from the [cn1-travis-template](https://github.com/shannah/cn1-travis-template) [https://github.com/shannah/cn1-travis-template] project. In particular, it will add the following files to your project:

1. **.travis.yml** - A travis script that is set up to build your project and run its unit tests.
2. **.travis/** - A directory containing some utility shell scripts that are used the **.travis.yml** script.

Activating/Deactivating Jobs

The **.travis.yml** script that was installed in the previous step includes jobs for testing your app on many different platforms. Currently JavaSE (the Codename One simulator), and a selection of iOS and Android versions are included, but we will be adding more platforms as time goes on. The goal is to provide jobs for every platform that Codename One supports.

You can select the platforms you want Travis to test against by selecting or deselecting the platform, as shown below.

Codename One Preferences

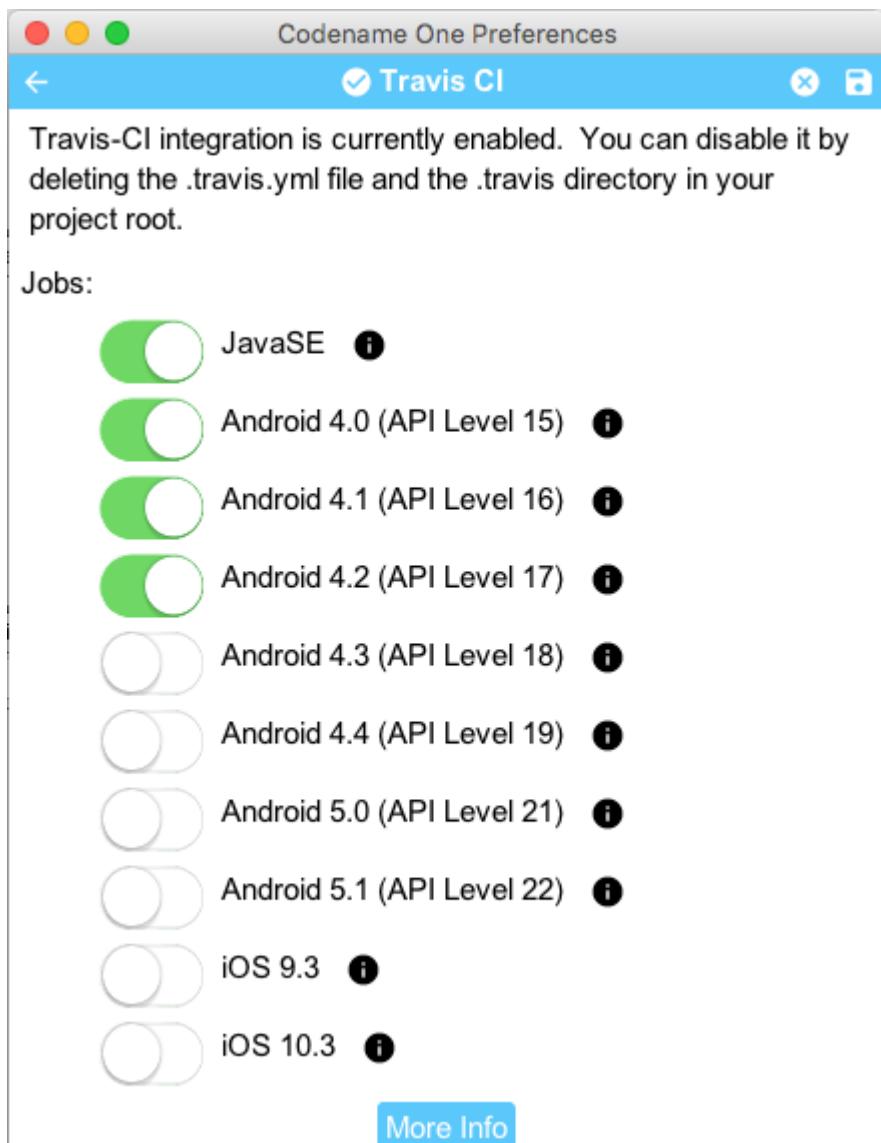
Travis CI

Travis-CI integration is currently enabled. You can disable it by deleting the .travis.yml file and the .travis directory in your project root.

Jobs:

- JavaSE i
- Android 4.0 (API Level 15) i
- Android 4.1 (API Level 16) i
- Android 4.2 (API Level 17) i
- Android 4.3 (API Level 18) i
- Android 4.4 (API Level 19) i
- Android 5.0 (API Level 21) i
- Android 5.1 (API Level 22) i
- iOS 9.3 i
- iOS 10.3 i

More Info



i Only the JavaSE job is available for non-enterprise subscribers. If you don't have an enterprise subscription, the Android and iOS options will be disabled in your settings. If you have an enterprise subscription, and your Android and iOS options are disabled, then you may not be logged in correctly. Check on the main menu of settings to ensure that you're logged into the correct account.

! On-device jobs such as Android and iOS require that you have the **CN1USER** and **CN1PASS** environment variables set in your Travis settings. These are used to build your project on the build server.

23.1.2. Pushing to GitHub

The process of setting up a Github repository is beyond the scope of this document. Please refer to [github's documentation](https://guides.github.com/activities/hello-world/) [<https://guides.github.com/activities/hello-world/>] for details on this process.

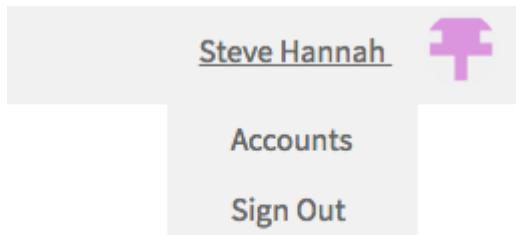
! Add a .gitignore file to your project to prevent you from committing all of your .jar files along with your project. Take a look at the [.gitignore file from the Kitchen Sink demo app](https://github.com/codenameone/KitchenSink/blob/master/.gitignore) [<https://github.com/codenameone/KitchenSink/blob/master/.gitignore>] for a sample .gitignore file that is suitable for a typical Codename One project.

23.1.3. Activate Repository On Travis

Now that your project is in a Github repository, you just need to activate the repository within Travis. For this part, you'll need an account on travis-ci.org (for public repositories) or travis-ci.com (for private repositories).

Steps to activate your repository in Travis.

1. Go to [travis-ci](https://travis-ci.org) [<https://travis-ci.org>], and login.
2. Click on your name in the upper right corner, to go to your profile page.



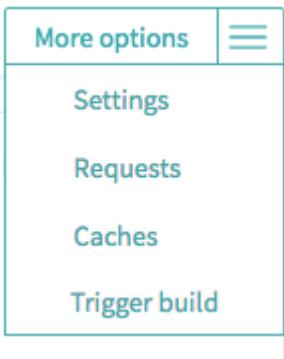
3. Click the "Sync account" button in the upper right to load your Github repositories.
4. Flip the switch next to the repository you want to activate in the list.

A screenshot of the Travis CI profile page for 'Steve Hannah'. The top navigation bar shows the user's name 'Steve Hannah' and a pink Travis logo. The main content area shows the user has 167 repositories. A 'Sync account' button is visible. Below it, instructions for activating a repository are shown: 'Flick the repository switch on', 'Add .travis.yml file to your repository', and 'Trigger your first build with a git push'. A list of repositories owned by 'shannah' is displayed, including 'admobfullscreen-codenameone', 'auth0-codenameone', 'background-exec-demo', 'binaries', and 'bluetoothle-codenameone'. A 'Filter repositories' input field is also present.

23.1.4. Setting Environment Variables

If you have any Android or iOS jobs activated in your travis script, you'll need to set the `CN1PASS` and `CN1USER` environment variables in Travis. Do this by first clicking on the repository, then select

"More Options" in the upper right, and "Settings".



The settings form will allow you to enter environment variables.

Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

CN1PASS 			
CN1USER 			
Name	Value	OFF	Display value in build log	Add

23.1.5. Testing Travis Script

To test the travis script, all you need to do is commit a change to your project in it, then push it to github.

```
$ git add .  
$ git commit -m "Some changes"  
$ git push origin master
```

Then go to <https://travis-ci.org>, and you should see your repository listed in the left menu. Click on it to follow your build status

Travis CI About Us Blog Status Help Steve Hannah

shannah / cn1-travis-template

build error

Current Branches Build History Pull Requests

More options



master Fixed node version in xcode 7.3 job

#41 started

Cancel build

Commit 58414e8

Compare 2c3647f..58414e8

Branch master

Steve Hannah authored and committed

Running for 23 sec

Build Jobs

	# 41.1		</> JDK: oraclejdk8		CN1_PLATFORM=javase CN1_RUN...	31 sec	
	# 41.2		</> JDK: oraclejdk8		CN1_PLATFORM=android API=15 C...	-	
	# 41.3		</> JDK: oraclejdk8		CN1_PLATFORM=android API=16 C...	4 min 59 sec	
	# 41.4		</> JDK: oraclejdk8		CN1_PLATFORM=android API=17 C...	11 min 25 sec	
	# 41.5		</> JDK: oraclejdk8		CN1_PLATFORM=android API=18 C...	12 min 1 sec	
	# 41.6		</> JDK: oraclejdk8		CN1_PLATFORM=android API=19 CN1_RUNTESTS_ANDROID_EMULATOR=1	7 min 37 sec	
	# 41.7		</> JDK: oraclejdk8			6 min 3 sec	
	# 41.8		</> JDK: oraclejdk8		CN1_PLATFORM=android API=22 C...	4 min 47 sec	
	# 41.9		</> Node.js: 6 Xcode: xcod...		CN1_PLATFORM=ios DEVICE=9.3 C...	3 min 21 sec	
	# 41.10		</> Node.js: 6 Xcode: xcod...		CN1_PLATFORM=ios DEVICE=10.3 ...	23 sec	

23.1.6. Writing Unit Tests

In order to make the most out of continuous integration, you'll want to write unit tests for your app. See [this video/post](https://www.codenameone.com/blog/test-it.html) [https://www.codenameone.com/blog/test-it.html] for a light introduction to Codename One unit testing, and also check out the [javadocs for the com.codename1.testing package](https://www.codenameone.com/javadoc/com/codename1/testing/package-summary.html) [https://www.codenameone.com/javadoc/com/codename1/testing/package-summary.html].

Disabling Travis

If you want to disable travis, or start fresh with a new travis script, you can simply delete the `.travis` directory and `.travis.yml` file from your project. If you want to reactivate travis later, you can do so through the Codename One settings using the same procedure as described above. That will re-download the latest travis script from the [online template](https://github.com/shannah/cn1-travis) [https://github.com/shannah/cn1-travis]

template].

24. Working with Codename One Sources

24.1. Checking out the Sources

```
$ mkdir workspace  
$ cd workspace  
$ git clone https://github.com/codenameone/CodenameOne
```



Creating a clean "workspace" directory is optional, and there is nothing special about the name "workspace". It is just recommended to create a clean directory into which you check out Codename One, because building Codename One will check out a few dependent projects and place them at the same level as the CodenameOne folder, so having a clean workspace will make it easier to manage.

24.2. Building Sources

```
$ cd CodenameOne  
$ ant
```



The `cd CodenameOne` command should take you to the root project directory which contains subfolders "CodenameOne", "Ports", etc.. - not the subfolder named "CodenameOne".

24.3. Running Unit Tests

You can run the unit test suite locally in the Codename One simulator using the "test-javase" ant target.

```
$ ant test-javase
```

24.4. Running iOS Unit Tests



Running iOS Unit tests is currently limited to internal use only by Codename One.

TLDR: You can run Codename One's unit tests on a local iOS simulator by running the "test-ios" ant target in the main CodenameOne directory. E.g.

```
ant test-ios
```

This assumes the following requirements

1. You are running Mac OS X with Xcode 7.3 or higher installed.
2. You have npm installed and in your environment PATH.
3. You have appium installed and running.

24.4.1. Installing npm

Either download the installer from <https://nodejs.org/en/> OR run `brew install node` from terminal.

24.4.2. Installing and Running Appium

Open a new terminal window, then

```
$ mkdir appium  
$ cd appium  
$ npm install appium  
$ ./node_modules/.bin/appium
```



You may want to run the last command `./node_modules/.bin/appium` as a daemon so that you can continue to use the same terminal window. E.g. `./node_modules/.bin/appium &`

24.4.3. Running the Unit tests

```
$ cd CodenameOne  
$ ant test-ios
```

24.4.4. Overriding the Codename One Build Server Target

```
$ ant test-ios -Dcn1.iphone.target=iphone_new
```

This will send to the "iphone_new" build server instead of the default "iphone" server.