

# Groovy Documentation



**Groovy Introduction**



**Cookbook Examples**



**Getting Started Guide**



**Testing Guide**



**User Guide**



**Developer Guide**



**Advanced Usage  
Guide**



**Modules**



**FAQ**



## Introduction to Groovy

1. Home	8
1.1 Project Information	10
1.1.1 Groovy Release Notes	11
1.1.1.1 Groovy 1.5 release notes	12
1.1.1.2 Groovy 1.6 release notes	28
1.1.1.3 Groovy 1.7 release notes	61
1.1.1.4 Groovy 1.8 release notes	72
1.1.1.4.1 Groovy 1.8.1 release notes	109
1.1.1.5 Groovy 2.0 release notes	111
1.1.1.6 Groovy 2.1 release notes	131
1.1.2 News and Further Information	153
1.1.2.1 Books	153
1.1.2.2 Groovy Series	156
1.1.2.3 PLEAC Examples	157
1.1.3 FAQ	158
1.1.3.1 Class Loading	158
1.1.3.2 FAQ - Classes and Object Orientation	159
1.1.3.3 FAQ - Closures	159
1.1.3.4 FAQ - Collections, Lists, etc.	159
1.1.3.5 FAQ - GSQL	161
1.1.3.6 FAQ - RegExp	164
1.1.3.7 General	167
1.1.3.7.1 How can I edit the documentation	167
1.1.3.8 Language questions	167
1.1.3.8.1 Can I break a Groovy statement into multiple lines anyway I want?	167
1.1.3.8.2 How can I dynamically add a library to the classpath	167
1.1.3.8.3 Why does == differ from Java	168
1.1.3.9 Learning about Groovy FAQ	169
1.1.3.10 License Information	169
1.1.3.11 Runtime vs Compile time, Static vs Dynamic	169
1.1.4 Roadmap, Discussions and Proposals	171
1.1.4.1 Discussions	172
1.1.4.1.1 Feature Wish List	172
1.1.4.1.2 New Groovy Console Wish List	184
1.1.4.2 Proposals	185
1.1.4.2.1 Annotation usage with properties	186
1.1.4.2.2 AST Macros and Annotations	188
1.1.4.2.3 C Sharp Groovinator	195
1.1.4.2.4 GJIT - Groovy Just-In-Time Compiler	207
1.1.4.2.5 Grapes and grab()	207
1.1.4.2.6 Groovy realtime archive internet lookup	213
1.1.4.2.7 JSR-295 Binding Builder	215
1.1.4.2.8 Macro Use Cases	222
1.1.4.2.9 MetaClass Redesign (by blackdrag)	225
1.1.4.2.10 MetaClass Redesign (by blackdrag) part 2	235
1.1.4.2.11 MOP 2.0 ideas	237
1.1.4.2.12 Multiple Assignment Proposal	239
1.1.4.2.13 Multiproject builds	249
1.1.4.2.14 Proposed Website Revamp	251
1.1.4.2.15 Web based GroovyShell	255
1.1.4.2.16 Miscellaneous Feature Requests and Enhancements	256
1.1.4.2.17 Groovy 2.0 modularization	256
1.1.4.3 Roadmap	279
1.1.4.3.1 notes on JDK 7 and JDK 8 future	280
1.1.4.3.2 Not Yet Documented	280
1.1.5 Articles	283
1.1.6 Success Stories	286
1.1.7 Version Scheme	286
1.2 Documentation	287
1.2.1 Getting Started Guide	288
1.2.1.1 Download	289
1.2.1.1.1 InvokeDynamic support	295
1.2.1.2 Feature Overview	296
1.2.1.2.1 Groovlets	297
1.2.1.2.2 Groovy Beans	301
1.2.1.2.3 GroovyMarkup	306
1.2.1.2.4 Groovy Templates	309
1.2.1.3 Quick Start	316
1.2.1.3.1 Installing Groovy	322
1.2.1.3.2 Running	323
1.2.1.4 Beginners Tutorial	326
1.2.1.4.1 Tutorial 1 - Getting started	326

1.2.1.4.2 Tutorial 2 - Code as data, or closures	336
1.2.1.4.3 Tutorial 3 - Classes and Objects	339
1.2.1.4.4 Tutorial 4 - Regular expressions basics	339
1.2.1.4.5 Tutorial 5 - Capturing regex groups	344
1.2.1.4.6 Tutorial 6 - Groovy SQL	350
1.2.1.5 Differences to Other Languages	353
1.2.1.5.1 Differences from Java	354
1.2.1.5.2 Differences from Python	356
1.2.1.5.3 Differences from Ruby	359
1.2.1.6 Groovy style and language feature guidelines for Java developers	359
1.2.1.7 For those new to both Java and Groovy	377
1.2.1.7.1 JN0025-Starting	378
1.2.1.7.2 JN0515-Integers	390
1.2.1.7.3 JN0525-Decimals	428
1.2.1.7.4 JN0535-Floats	457
1.2.1.7.5 JN0545-Dates	500
1.2.1.7.6 JN1015-Collections	539
1.2.1.7.7 JN1025-Arrays	576
1.2.1.7.8 JN1035-Maps	584
1.2.1.7.9 JN1515-Characters	606
1.2.1.7.10 JN1525-Strings	626
1.2.1.7.11 JN1535-Patterns	649
1.2.1.7.12 JN2015-Files	691
1.2.1.7.13 JN2025-Streams	706
1.2.1.7.14 JN2515-Closures	740
1.2.1.7.15 JN2525-Classes	758
1.2.1.7.16 JN2535-Control	783
1.2.1.7.17 JN3015-Types	810
1.2.1.7.18 JN3025-Inheritance	833
1.2.1.7.19 JN3035-Exceptions	850
1.2.1.7.20 JN3515-Interception	857
1.2.1.7.21 JN3525-MetaClasses	904
1.2.1.7.22 JN3535-Reflection	926
1.2.1.8 Groovy for the Office	940
1.2.1.9 Groovy Quick Start Project	940
1.2.2 Reference	942
1.2.3 User Guide	942
1.2.3.1 Control Structures	947
1.2.3.1.1 Logical Branching	947
1.2.3.1.2 Looping	951
1.2.3.1.3 Returning values from if-else and try-catch blocks	956
1.2.3.2 Operators	957
1.2.3.2.1 Operator Overloading	962
1.2.3.3 Bitwise Operations	965
1.2.3.4 Scripts and Classes	966
1.2.3.5 Statements	969
1.2.3.5.1 Extended Guide to Method Signatures	977
1.2.3.6 Reserved Words	982
1.2.3.7 Strings and GString	985
1.2.3.8 Closures	993
1.2.3.8.1 Closures - Formal Definition	997
1.2.3.8.2 Closures - Informal Guide	101
1.2.3.9 Builders	102
1.2.3.9.1 How Builders Work	103
1.2.3.9.2 FactoryBuilderSupport	103
1.2.3.9.3 The @DelegatesTo annotation	103
1.2.3.10 Groovy Truth	105
1.2.3.11 Input Output	105
1.2.3.12 Collections	105
1.2.3.13 Regular Expressions	106
1.2.3.13.1 Documenting Regular Expressions in Groovy	106
1.2.3.14 GDK Extensions to Object	108
1.2.3.15 Grape	108
1.2.3.16 Advanced OO	110
1.2.3.16.1 Groovy way to implement interfaces	110
1.2.3.17 Annotations with Groovy	110
1.2.3.18 Generics	111
1.2.3.19 Using Enums	111
1.2.3.20 Static Import Usage	112
1.2.3.21 Scoping and the Semantics of "def"	112
1.2.3.22 Database features	112
1.2.3.23 Dynamic Groovy	114

1.2.3.23.1 Evaluating the MetaClass runtime	114
1.2.3.23.2 ExpandoMetaClass	114
1.2.3.23.3 Global AST Transformations	116
1.2.3.23.4 Local AST Transformations	117
1.2.3.23.5 Per-Instance MetaClass	117
1.2.3.23.6 Runtime mixins	117
1.2.3.23.7 Using invokeMethod and getProperty	118
1.2.3.23.8 Using methodMissing and propertyMissing	118
1.2.3.24 Ant Integration with Groovy	118
1.2.3.24.1 The groovy Ant Task	118
1.2.3.24.2 The groovyc Ant Task	119
1.2.3.24.3 The groovydoc Ant task	119
1.2.3.24.4 Using Ant from Groovy	120
1.2.3.24.5 Using Ant Libraries with AntBuilder	121
1.2.3.25 Bean Scripting Framework	122
1.2.3.26 Compile-time Metaprogramming - AST Transformations	122
1.2.3.26.1 Bindable and Vetoable transformation	122
1.2.3.26.2 Building AST Guide	123
1.2.3.26.3 Category and Mixin transformations	123
1.2.3.26.4 Compiler Phase Guide	123
1.2.3.26.5 Delegate transformation	124
1.2.3.26.6 Immutable AST Macro	124
1.2.3.26.7 Immutable transformation	124
1.2.3.26.8 Lazy transformation	124
1.2.3.26.9 Newify transformation	125
1.2.3.26.10 PackageScope transformation	125
1.2.3.26.11 Singleton transformation	125
1.2.3.27 GPath	125
1.2.3.28 Groovy and JMX	125
1.2.3.28.1 Groovy JmxBuilder	128
1.2.3.29 Groovy Categories	131
1.2.3.30 Groovy CLI	132
1.2.3.31 Groovy Console	132
1.2.3.32 Groovy Math	133
1.2.3.33 Groovy Shell	134
1.2.3.34 Groovy Utils	135
1.2.3.34.1 ConfigSlurper	135
1.2.3.34.2 ObjectGraphBuilder	135
1.2.3.34.3 ObservableMap	136
1.2.3.35 GUI Programming with Groovy	136
1.2.3.35.1 Swing Builder	136
1.2.3.35.2 SwingXBuilder	156
1.2.3.36 IDE Support	157
1.2.3.36.1 Eclipse Plugin	157
1.2.3.36.2 IntelliJ IDEA Plugin by JetBrains	181
1.2.3.36.3 Old and deprecated pages	182
1.2.3.36.4 Other Plugins	182
1.2.3.37 Integration	183
1.2.3.38 JSR 223 Scripting with Groovy	183
1.2.3.39 Logging	184
1.2.3.40 Migration From Classic to JSR syntax	184
1.2.3.41 Processing XML	185
1.2.3.41.1 Creating XML using Groovy's MarkupBuilder	185
1.2.3.41.2 Creating XML using Groovy's StreamingMarkupBuilder	186
1.2.3.41.3 Creating XML with Groovy and DOM	186
1.2.3.41.4 Processing XML with XSLT	187
1.2.3.41.5 Reading XML using Groovy's DOMCategory	187
1.2.3.41.6 Reading XML using Groovy's XmlParser	188
1.2.3.41.7 Reading XML using Groovy's XmlSlurper	188
1.2.3.41.8 Reading XML with Groovy and DOM	189
1.2.3.41.9 Reading XML with Groovy and SAX	189
1.2.3.41.10 Reading XML with Groovy and StAX	189
1.2.3.41.11 Reading XML with Groovy and XPath	190
1.2.3.41.12 Updating XML with DOMCategory	191
1.2.3.41.13 Updating XML with XmlParser	191
1.2.3.41.14 Updating XML with XmlSlurper	192
1.2.3.41.15 Using Other XML Libraries	192
1.2.3.41.16 Validating XML with a DTD	194
1.2.3.41.17 Validating XML with a W3C XML Schema	195
1.2.3.41.18 Validating XML with RELAX NG	195
1.2.3.41.19 XML Example	195
1.2.3.42 Process Management	196

1.2.3.43 Running Groovy on .NET 2.0 using IKVM	196
1.2.3.44 Things to remember	196
1.2.3.45 Things you can do but better leave undone	196
1.2.3.46 Using Spring Factories with Groovy	197
1.2.3.47 WebStart	197
1.2.3.48 OSGi and Groovy	197
1.2.4 Advanced Usage Guide	199
1.2.4.1 Advanced compiler configuration	199
1.2.4.2 Ant Task Troubleshooting	201
1.2.4.3 Make a builder	201
1.2.4.4 BuilderSupport	203
1.2.4.5 Compiling Groovy	203
1.2.4.5.1 Compiling With Maven2	204
1.2.4.6 Creating an extension module	204
1.2.4.7 Design Patterns with Groovy	204
1.2.4.7.1 Abstract Factory Pattern	205
1.2.4.7.2 Adapter Pattern	205
1.2.4.7.3 Bouncer Pattern	206
1.2.4.7.4 Chain of Responsibility Pattern	206
1.2.4.7.5 Composite Pattern	207
1.2.4.7.6 Decorator Pattern	207
1.2.4.7.7 Delegation Pattern	208
1.2.4.7.8 Flyweight Pattern	209
1.2.4.7.9 Iterator Pattern	209
1.2.4.7.10 Loan my Resource Pattern	209
1.2.4.7.11 Null Object Pattern	209
1.2.4.7.12 Pimp my Library Pattern	210
1.2.4.7.13 Proxy Pattern	210
1.2.4.7.14 Singleton Pattern	210
1.2.4.7.15 State Pattern	211
1.2.4.7.16 Strategy Pattern	213
1.2.4.7.17 Template Method Pattern	213
1.2.4.7.18 Visitor Pattern	214
1.2.4.8 Dynamic language beans in Spring	215
1.2.4.9 Embedding Groovy	217
1.2.4.10 Mixed Java and Groovy Applications	217
1.2.4.11 Influencing class loading at runtime	218
1.2.4.12 Internationalization	218
1.2.4.13 Meta-annotations	219
1.2.4.14 Optimising Groovy bytecodes with Soot	219
1.2.4.15 Polyglot Programming with Groovy	219
1.2.4.15.1 Calling Clojure from Groovy	219
1.2.4.15.2 Calling Scala from Groovy	220
1.2.4.15.3 Constraint Programming	220
1.2.4.15.4 Functional Programming with Groovy	221
1.2.4.15.5 JSR-223 access to other JVM languages	223
1.2.4.16 Refactoring with Groovy	223
1.2.4.16.1 Introduce Assertion	223
1.2.4.16.2 Replace Inheritance with Delegation	223
1.2.4.17 Security	224
1.2.4.18 Writing Domain-Specific Languages	224
1.2.4.19 Type checking extensions	224
1.2.5 Testing Guide	227
1.2.5.1 Unit Testing	227
1.2.5.1.1 Using JUnit 4 with Groovy	228
1.2.5.2 Using Other Testing Frameworks	229
1.2.5.2.1 Using EasyMock with Groovy	229
1.2.5.2.2 Using Instinct with Groovy	229
1.2.5.2.3 Using JBehave with Groovy	230
1.2.5.2.4 Using JDummy with Groovy	231
1.2.5.2.5 Using JMockit with Groovy	231
1.2.5.2.6 Using JMock with Groovy	231
1.2.5.2.7 Using Popper with Groovy	231
1.2.5.2.8 Using RMock with Groovy	233
1.2.5.2.9 Using TestNG with Groovy	233
1.2.5.3 Groovy Mocks	233
1.2.5.3.1 Developer Testing using Closures instead of Mocks	234
1.2.5.3.2 Developer Testing using Maps and Expandos instead of Mocks	234
1.2.5.3.3 Mocking Static Methods using Groovy	235
1.2.5.3.4 Using MockFor and StubFor	235
1.2.5.4 GMock	236
1.2.5.5 Integrating TPTP	236

1.2.5.6 Model-based testing using ModelJUnit	237
1.2.5.7 Test Combinations	237
1.2.5.7.1 Effectiveness of testing combinations with all pairs	239
1.2.5.8 Test Coverage	239
1.2.5.8.1 Code Coverage with Cobertura	239
1.2.5.9 Testing Web Applications	240
1.2.5.10 Testing Web Services	240
1.2.5.11 Using Testing Frameworks with Groovy	241
1.2.5.11.1 Using GSpec with Groovy	242
1.2.6 Developer Guide	243
1.2.6.1 Building Groovy from Source	243
1.2.6.1.1 Setting up IntelliJ IDEA to build Groovy	243
1.2.6.2 Building Groovy from Source using Ant	243
1.2.6.3 Continuous Integration	244
1.2.6.4 From source code to bytecode	244
1.2.6.5 Git	244
1.2.6.6 Groovy Backstage	245
1.2.6.6.1 Groovy Method Invokation	245
1.2.6.7 Groovy Internals	245
1.2.6.8 Release Process	245
1.2.6.9 Setup Groovy Development Environment	246
1.2.7 Cookbook Examples	246
1.2.7.1 Accessing SQLServer using groovy	246
1.2.7.2 Alternate Spring-Groovy-Integration	246
1.2.7.3 Auto setup and download dependencies jars	247
1.2.7.4 Batch Image Manipulation	247
1.2.7.5 Calculating a SHA1 hash for large files	248
1.2.7.6 command line groovy doc or methods lookup	248
1.2.7.7 Compute distance from Google Earth Path (in .kml file)	249
1.2.7.8 Concurrency with Groovy	250
1.2.7.9 Convert SQL Result To XML	251
1.2.7.10 Embedded Derby DB examples	251
1.2.7.11 Embedding a Groovy Console in a Java Server Application	252
1.2.7.12 Executing External Processes From Groovy	253
1.2.7.13 Expect for Groovy	253
1.2.7.14 Formatting simple tabular text data	256
1.2.7.15 Greedy Coin Changer in Groovy	258
1.2.7.16 Groovy Alternatives to Inner Classes	258
1.2.7.17 Groovy as script	258
1.2.7.18 Guy Steele's word split example using Groovy and GPars	259
1.2.7.19 How many Triangles (list, regex, grep) example	260
1.2.7.20 Installing or Update Groovy Automatically	260
1.2.7.21 InstallTextMateGroovyBundle	260
1.2.7.22 Integrating Groovy in an application - a success story	261
1.2.7.23 Iterator Tricks	262
1.2.7.24 Martin Fowler's closure examples in Groovy	263
1.2.7.25 Other Examples	263
1.2.7.26 Parsing Groovy Doc Online	263
1.2.7.27 Plotting graphs with JFreeChart	264
1.2.7.28 PoorMansMixins	264
1.2.7.29 Reading from a Blob	265
1.2.7.30 Recipes For File	265
1.2.7.31 Regex search for a class in the JVM runtime classpath	266
1.2.7.32 Search one or more jar files for a specified file or directory	266
1.2.7.33 Simple file download from URL	267
1.2.7.34 Solving Sudoku	267
1.2.7.35 StreamingMarkupBuilder and IMAP client example	267
1.2.7.36 SwingBuilder with custom widgets and observer pattern	268
1.2.7.37 Tomcat tools	268
1.2.7.38 Unsign Jar Files (Recursively)	268
1.2.7.39 Using Groovy with Boost	269
1.2.7.40 Using Hibernate with Groovy	269
1.2.7.41 Using JGoodies Animation with Groovy	270
1.2.7.42 Using JScience with Groovy	270
1.2.7.43 Using MarkupBuilder for Agile XML creation	271
1.2.7.44 Using Regex Patterns to find word phone numbers	273
1.2.7.45 Using the Delegating Meta Class	274
1.2.7.46 Using the Eclipse Modeling Framework (EMF)	274
1.2.7.47 Using the Eclipse UML2 Framework	274
1.2.7.48 Using the Proxy Meta Class	274
1.2.7.49 Windows Look And Feel for groovyConsole	275
1.2.7.50 WrappingGroovyScript	275

1.2.7.51 Writing to a Blob	275
1.2.7.52 Yaml and Groovy	276
1.2.8 New Documents	276
1.2.8.1 New User Guide	276
1.2.8.1.1 Downloading, Installing and Running	276
1.2.8.1.2 Up and Running	277
1.2.9 TMPGroovy Categories	278
1.3 Community and Support	279
1.3.1 Support	280
1.3.2 Contributing	280
1.3.3 Forums and Newsgroups	281
1.3.4 Mailing Lists	281
1.3.5 User Groups	281
1.3.6 Modules	281
1.3.6.1 COM Scripting	281
1.3.6.1.1 All About Arrays	282
1.3.6.1.2 Change Log	282
1.3.6.1.3 COM Data Types in Scriptom	282
1.3.6.1.4 COM Events	283
1.3.6.1.5 COM Methods and Properties in Scriptom	283
1.3.6.1.6 Passing Values by Reference (in-out)	283
1.3.6.1.7 Scriptom Archive	283
1.3.6.1.8 Scriptom Articles	285
1.3.6.1.9 The Least You Need to Know about COM	286
1.3.6.2 Gant	286
1.3.6.3 GFreeMarker	286
1.3.6.4 Google Data Support	287
1.3.6.5 Gram	287
1.3.6.6 GraphicsBuilder	287
1.3.6.6.1 Alphabetical Index	288
1.3.6.6.2 GraphicsBuilder - Animation	296
1.3.6.6.3 GraphicsBuilder - GDK	296
1.3.6.6.4 GraphicsBuilder - Groodles	296
1.3.6.6.5 GraphicsBuilder - Groups	297
1.3.6.6.6 GraphicsBuilder - Misc	297
1.3.6.6.7 GraphicsBuilder - Outlines	297
1.3.6.6.8 GraphicsBuilder - Paints	298
1.3.6.6.9 GraphicsBuilder - Shapes	299
1.3.6.6.10 GraphicsBuilder - Strokes	302
1.3.6.6.11 Graphicsbuilder - SVG	302
1.3.6.6.12 GraphicsBuilder - Swing	302
1.3.6.6.13 GraphicsBuilder - Transformations	302
1.3.6.6.14 GraphicsBuilder - Tutorials	302
1.3.6.7 Grapplet	302
1.3.6.8 Griffon	303
1.3.6.8.1 Download Griffon	303
1.3.6.8.2 Greet	303
1.3.6.8.3 Griffon JavaOne 2008 Dev Notes	303
1.3.6.8.4 Griffon Quick Start	303
1.3.6.8.5 Griffon RoadMap	304
1.3.6.8.6 Installing Griffon	304
1.3.6.8.7 UberBuilder	304
1.3.6.9 Groosh	304
1.3.6.10 Groovy Jabber-RPC	305
1.3.6.11 GroovyJMS	305
1.3.6.11.1 GroovyJMS Design Docs	305
1.3.6.11.2 GroovyJMS Docs	305
1.3.6.11.3 GroovyJMS Reference Links	306
1.3.6.11.4 GroovyJMS Roadmap and Planning	306
1.3.6.11.5 GroovyJMS - v0.1 Docs and Example	306
1.3.6.11.6 Groovy Messaging Service API	307
1.3.6.12 GroovyLab	307
1.3.6.13 Groovy Monkey	307
1.3.6.14 GroovyRestlet	308
1.3.6.14.1 GroovyRestlet User Guide	308
1.3.6.14.2 RestletTutorialExamples	309
1.3.6.14.3 SpringIntegrationExamples	310
1.3.6.15 Groovy Science	310
1.3.6.16 Groovy SOAP	310
1.3.6.17 GroovySWT	311
1.3.6.17.1 A sample application	311
1.3.6.17.2 Binding	312

1.3.6.17.3 Changelog	312
1.3.6.17.4 groovy-swt - Comparing the java and the groovy code	313
1.3.6.18 GroovyWS	314
1.3.6.18.1 Consuming a web service	314
1.3.6.18.2 Currency rate calculator	314
1.3.6.18.3 GroovyWS grape config file	314
1.3.6.18.4 GroovyWS installation	314
1.3.6.18.5 Publishing a web-service	314
1.3.6.18.6 TerraServer-USA by Microsoft	315
1.3.6.18.7 Using the Aegis mapping	315
1.3.6.18.8 Using the client with complex objects	315
1.3.6.18.9 Using the Grails XFire plugin and GroovyWS	315
1.3.6.18.10 Using WSClnt in Grails	315
1.3.6.18.11 Using WS-Security	316
1.3.6.18.12 WSClnt configuration	316
1.3.6.19 GSP	316
1.3.6.20 GSQL	317
1.3.6.21 HTTP Builder	317
1.3.6.22 JideBuilder	317
1.3.6.23 MetaBuilder	318
1.3.6.24 Native Launcher	319
1.3.6.25 Proxy-o-Matic	320
1.3.6.26 Windows NSIS-Installer	320
1.3.6.27 Windows Services	321
1.3.6.28 WingSBuilder	321
1.3.6.29 XMLRPC	322
1.3.6.30 GroovyFX	322
1.3.6.30.1 Layout in GroovyFX	323
1.3.6.30.2 SceneGraphBuilder	323
1.3.6.30.3 SceneGraphBuilder - containers	323
1.3.6.30.4 SceneGraphBuilder - controls	326
1.3.6.30.5 SceneGraphBuilder - effects	332
1.3.6.30.6 SceneGraphBuilder - input	336
1.3.6.30.7 SceneGraphBuilder - nodes	337
1.3.6.30.8 SceneGraphBuilder - paint	338
1.3.6.30.9 SceneGraphBuilder - shapes	339
1.3.6.30.10 SceneGraphBuilder - stage	342
1.3.6.30.11 SceneGraphBuilder - transforms	342
1.3.6.30.12 SceneGraphBuilder - web	343
1.3.6.30.13 TimelineBuilder	343
1.3.6.31 Gaelyk	344
1.3.6.32 GMaven	344
1.3.6.33 Groovy Transforms	344
1.3.6.33.1 Scalify	344
1.3.7 Related Projects	344

# Home

## Groovy...

- is an agile and **dynamic language** for the **Java Virtual Machine**
- builds upon the strengths of Java but has **additional power features** inspired by languages like Python, Ruby and Smalltalk
- makes **modern programming features** available to Java developers with **almost-zero learning curve**
- provides the ability to **statically type check** and **statically compile** your code for robustness and performance
- supports **Domain-Specific Languages** and other compact syntax so your code becomes **easy to read and maintain**
- makes writing shell and build scripts easy with its **powerful processing primitives**, OO abilities and an Ant DSL
- increases developer productivity by **reducing scaffolding code** when developing web, GUI, database or console applications
- **simplifies testing** by supporting unit testing and mocking out-of-the-box
- seamlessly **integrates with all existing Java classes and libraries**
- compiles straight to Java bytecode so you can use it anywhere you can use Java

## Experience Groovy 2.1

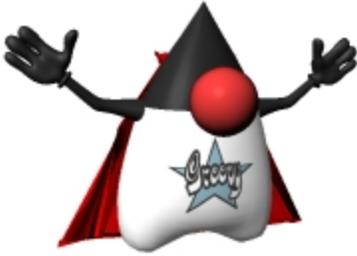


**Groovy 2.1** is the latest **major** and **stable version** of the popular alternative language for the JVM.

To learn more about the novelties, make sure to **read the [detailed release notes](#)**. In a nutshell, Groovy 2.1

- **offers full support for the JDK 7 “invoke dynamic”** bytecode instruction and API for improved performance,
- goes beyond conventional static type checking capabilities with a **special annotation to assist with documentation and type safety of Domain-Specific Languages** and adds **static type checker extensions**,
- provides **additional compilation customization** options,
- features a **meta-annotation facility** for combining annotations elegantly,
- and provides various other enhancements and minor improvements.

"Groovy is like a super version of Java. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing. If you are a developer, tester or script guru, you have to love Groovy."



## Samples

A simple hello world script:

```
def name='World'; println "Hello $name!"
```

A more sophisticated version using Object Orientation:

```
class Greet {
    def name
    Greet(who) { name = who[0].toUpperCase() +
                who[1..-1] }
    def salute() { println "Hello $name!" }
}

g = new Greet('world') // create object
g.salute()             // output "Hello World!"
```

Leveraging existing Java libraries:

```
import static org.apache.commons.lang.WordUtils.*

class Greeter extends Greet {
    Greeter(who) { name = capitalize(who) }
}

new Greeter('world').salute()
```

On the command line:

```
groovy -e "println 'Hello ' + args[0]" World
```

Latest news [\[more\]](#)

## [Guillaume Laforge](#) posted on Jun 14, 2013 [Groovy 2.1.5 bug fix release](#)

Dear all,

I'm pleased to announce the release of Groovy 2.1.5.

Groovy 2.1.5 is a bug fix release of the Groovy 2.1 branch. In particular, it fixes a [problem](#) we've encountered with Groovy 2.1.4 where the extension module descriptors were not found in the "all" JAR, which meant that the usual nice GDK methods extensions for XML and other modules were not found by Groovy.

You can download Groovy 2.1.5 in the [download area](#) and have a look at the [JIRA release notes](#).

Thanks to all who contributed to this release!

Keep on groovy'ing!

[Edit](#)

## Project Information

Some useful project information:

- [Groovy Release Notes](#)
  - [Groovy 1.5 release notes](#)
  - [Groovy 1.6 release notes](#)
  - [Groovy 1.7 release notes](#)
  - [Groovy 1.8 release notes](#)
    - [Groovy 1.8.1 release notes](#)
  - [Groovy 2.0 release notes](#)
  - [Groovy 2.1 release notes](#)
- [News and Further Information](#)
  - [Books](#)
  - [Groovy 1.0 parties](#)
  - [Groovy Series](#)
  - [latest posts from our mailing-lists](#)
  - [News](#)
    - [Release candidate of Groovy 2.0 available](#)
  - [PLEAC Examples](#)
- [Events](#)
- [FAQ](#)
  - [Class Loading](#)
  - [FAQ - Classes and Object Orientation](#)
  - [FAQ - Closures](#)
  - [FAQ - Collections, Lists, etc.](#)

- [FAQ - GSQL](#)
- [FAQ - RegExp](#)
- [General](#)
  - [How can I edit the documentation](#)
- [Language questions](#)
  - [Can I break a Groovy statement into multiple lines anyway I want?](#)
  - [How can I dynamically add a library to the classpath](#)
  - [Why does == differ from Java](#)
- [Learning about Groovy FAQ](#)
- [License Information](#)
- [Runtime vs Compile time, Static vs Dynamic](#)
- [Roadmap, Discussions and Proposals](#)
  - [Discussions](#)
    - [Feature Wish List](#)
    - [New Groovy Console Wish List](#)
  - [Proposals](#)
    - [Annotation usage with properties](#)
    - [AST Macros and Annotations](#)
    - [C Sharp Groovinator](#)
    - [GJIT - Groovy Just-In-Time Compiler](#)
    - [Grapes and grab\(\)](#)
    - [Groovy realtime archive internet lookup](#)
    - [JSR-295 Binding Builder](#)
    - [Macro Use Cases](#)
    - [MetaClass Redesign \(by blackdrag\)](#)
    - [MetaClass Redesign \(by blackdrag\) part 2](#)
    - [MOP 2.0 ideas](#)
    - [Multiple Assignment Proposal](#)
      - [Multiple Assignment](#)
    - [Multiproject builds](#)
    - [Proposed Website Revamp](#)
    - [Web based GroovyShell](#)
    - [Miscellaneous Feature Requests and Enhancements](#)
    - [Groovy 2.0 modularization](#)
  - [Roadmap](#)
    - [notes on JDK 7 and JDK 8 future](#)
    - [Not Yet Documented](#)
      - [Gldapwrap - Usage](#)
- [Articles](#)
  - [James Strachan on Groovy](#)
    - [Nov 2004 - JSR Keynote](#)
- [Success Stories](#)
- [Version Scheme](#)
- [Google Summer of Code](#)
  - [GSoC 2013](#)

## Groovy Release Notes

- [Groovy 1.5 release notes](#)
- [Groovy 1.6 release notes](#)
- [Groovy 1.7 release notes](#)
- [Groovy 1.8 release notes](#)
- [Groovy 2.0 release notes](#)

- [Groovy 2.1 release notes](#)

## Groovy 1.5 release notes

### InfoQ

Article originally [published on InfoQ](#)

[Groovy](#), the Java-like dynamic language for the JVM, matures over time like good wines. After the successful release of Groovy 1.0 in January 2007, the next major milestone with the 1.5 label already hits the shelves. With it, come several interesting novelties that we will examine in this article. The major addition to the language is the support of Java 5 features with annotations, generics and enums, making Groovy **the sole alternative dynamic language for the JVM fully supporting frameworks like Spring, Hibernate, JPA**, Google Guice, or TestNG. Apart from the new Java 5 features, a few syntax enhancements find their way in the language, as well as a more powerful dynamic behavior customization, a Swing UI builder on steroids, and improved tooling support.

## *A groovier Groovy and why it matters*

Groovy's key selling point has always been its **seamless integration with Java**. You can mix and match Groovy and Java classes together in very easy ways: You may have a Groovy class extending a Java class implementing a Groovy interface, and vice versa. Most of the other alternative JVM languages unfortunately won't let you seamlessly interchange classes of two different languages. So if you want to use the best language for the job without compromising your nice class hierarchy, you won't have many options to choose from, and Groovy gives you all the freedom for integrating both languages in the most transparent way.

Groovy shares the same libraries, the same object model, the same threading model, and the same security model as Java. In a way, you can consider Groovy as an implementation detail of your Java project, **without having to bear problematic impedance mismatch issues**.

Groovy is Java, and Groovy makes Java groovier. Compared with other languages, Groovy is certainly the language that provides the **flattest learning curve to Java developers**, thanks to a very similar syntax.

This is even more important to keep in mind that Groovy generates normal Java bytecode and uses the usual JDK libraries, so you won't need to learn whole new APIs or have complex integration mechanisms: out of the box, Groovy and Java are interchangeable. The added benefit is that you can **protect the investment** you made in Java **skills for your developers**, or in **costly application servers**, or third party or **home-grown libraries**, as you can reuse all of them without a problem from Groovy.

Speaking of calling into JDK, third party, or in-house libraries, alternative languages which don't support strong typing, can't always call all Java methods because of the fact they can't choose a certain polymorphic variation of the same method. When choosing an alternative language to improve your productivity or make your code more readable, if you need to call other Java classes, you will have to be very careful to the choice of language you will make, as you may encounter some road blocks along the way.

These days, all major Enterprise frameworks require the use of features like annotations, enums or generics, to be leveraged at their fullest extent. Fortunately, with Groovy 1.5, developers can benefit from the support of all these Java 5 features in their projects. Let's see how annotations, enums and generics can be used from Groovy.

### Java 5 additions

The Groovy compiler always generates Java bytecode which is compatible with older Java VMs, but relies on JDK 1.4 for the core libraries it is using. However, for certain of these Java 5 additions, using Java 5 bytecode was required, so that, for example, the generated classes may contain the bytecode information representing

annotations with runtime retention policy. So although Groovy 1.5 can run on JDK 1.4, certain of these features will only be usable on JDK 5 - when this is the case, it will be mentioned in this article.

### ***Variable arguments***

In Java 5, the ellipsis notation was created to denote methods with variable length arguments. With those little triple dots, Java lets users put as many parameters of the same type at the end of a method - actually, the vararg parameter is just an array of elements of that type. Varargs were already present in Groovy 1.0 - and still work with a JDK 1.4 runtime, but it is good to show how you can use them. Basically, whenever the last argument of a method is an array of objects, or a parameter declaration with a triple dot, you can pass multiple parameters to this method.

A first example will show the usage of varargs in Groovy with the ellipsis:

```
int sum(int... someInts) {  
def total = 0  
for (int i = 0; i < someInts.size(); i++)  
total += someInts[i]  
return total  
}
```

```
assert sum(1) == 1  
assert sum(1, 2) == 3  
assert sum(1, 2, 3) == 6
```

The assertions used in this example show how we can pass as many ints as desired. It is also interesting to see that, for more syntax compatibility with Java, the classical for loop has been added into Groovy - despite the presence of the groovier version with the in keyword that can also transparently iterate over various array or collection types.

Note that it is possible to have varargs support even with an array as last parameter by declaring the method as follows:

```
int sum(int[] someInts) { /* */ }
```

This snippet is really trivial, and there are obviously more expressive ways of calculating a sum. For instance, if you have a list of numbers, you can sum all of them in a single line of code:

```
assert [1, 2, 3].sum() == 6
```

Varargs in Groovy don't require JDK 5 as the underlying Java runtime, unlike annotations that we are now going to look at in the following section.

### ***Annotations***

As shown in the [documentation of JBoss Seam](#) which supports Groovy for writing its entities, controllers and components, annotations like **@Entity**, **@Id**, **@Override** and others can be used to decorate your beans:

```
@Entity
@Name("hotel")
class Hotel implements Serializable
{
@Id @GeneratedValue
Long id

@Length(max=50) @NotNull
String name

@Length(max=100) @NotNull
String address

@Length(max=40) @NotNull
String city

@Length(min=2, max=10) @NotNull
String state

@Length(min=4, max=6) @NotNull
String zip

@Length(min=2, max=40) @NotNull
String country

@Column(precision=6, scale=2)
BigDecimal price

@Override
String toString() {
return "Hotel({name}, {address}, {city},
{zip})"
}
}
```

The Hotel entity is marked with the @Entity annotation, and it's given a name through @Name. Different parameters can be passed to your annotations like in the @Length annotation constraint where different upper and lower bound can be set for validation purpose. You can also notice **Groovy properties** in action: Where are all the getters and setters? Where are the public or private modifiers? You don't have to wait for Java 7 or 8 to get properties! By convention, defining a property is as simple as String country: a private country field will be auto-generated, as well as a public getter and setter. **Your code becomes naturally more concise and readable.**

Annotations can be used on classes, fields, methods and method parameters, like in Java. There are, however, two gotchas to be aware of. Firstly, you can use annotations in Groovy, but you cannot yet define them - however, it will be possible in an upcoming version of Groovy. Secondly, although the syntax is almost 100% the same as in Java, there is a little difference when an array of values is passed in parameter of the annotation: instead of curly braces to surround the elements, Groovy requires the use of square brackets to offer a more homogeneous syntax - since Groovy lists and arrays use square brackets to surround their elements as well.

With annotations in Groovy 1.5, you can easily define your JPA or Hibernate annotated beans (<http://www.curious-creature.org/2007/03/25/persistence-made-easy-with-groovy-and-jpa/>) in Groovy, add an @Transactional annotation on your Spring services, test your Swing UI with TestNG and Fest ([http://www.jroller.com/aalmiray/entry/testing\\_groovy\\_uis\\_with\\_fest](http://www.jroller.com/aalmiray/entry/testing_groovy_uis_with_fest)). All the useful and powerful enterprise frameworks leveraging annotations can be used from your Groovy-powered projects.

### *Enums*

Whenever you need a fixed set of constants of a same type, Enums come in handy. Say you need a clean way to define constants for days without resorting to using integer constants? Then Enums are your friend. The following snippet shows how to define the days of the week:

```
enum Day {  
SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
THURSDAY, FRIDAY, SATURDAY  
}
```

Once you have defined your enum, you can use it as in Java with the usual notation **Day.MONDAY** and you can spice up your switch / case statements with it as well:

```
def today = Day.SATURDAY
switch (today) {
// Saturday or Sunday
case [Day.SATURDAY, Day.SUNDAY]:
println "Weekends are cool"
break
// a day between Monday and Friday
case Day.MONDAY..Day.FRIDAY:
println "Boring work day"
break
default:
println "Are you sure this is a valid day?"
}
```

Notice that Groovy's switch is a bit more powerful than C-like languages switches in that it is possible to use any kind of object in the switch and case. Instead of stacking up seven different case block with each enumerated value, you can regroup them in lists or ranges: whenever the value is in the list or the range, the case will be true and its associated instructions will be executed.

A more complex example inspired by the Java tutorial takes a more astronomical perspective on enums, and shows how your enums can have properties, constructors and methods:

```

enum Planet {
MERCURY (3.303e+23, 2.4397e6),
VENUS    (4.869e+24, 6.0518e6),
EARTH    (5.976e+24, 6.37814e6),
MARS     (6.421e+23, 3.3972e6),
JUPITER  (1.9e+27,    7.1492e7),
SATURN   (5.688e+26, 6.0268e7),
URANUS   (8.686e+25, 2.5559e7),
NEPTUNE  (1.024e+26, 2.4746e7)

```

```

double mass
double radius

```

```

Planet(double mass, double radius) {
this.mass = mass;
this.radius = radius;
}

```

```

void printMe() {
println "${name()} has a mass of ${mass} " +
"and a radius of ${radius}"
}
}
Planet.EARTH.printMe()

```

Like annotations, enums in Groovy require a JDK 5+ to run on, as Java 5 bytecode is generated.

#### *Static imports*

In our previous examples of enums, we always had to prefix the enumerated value with its parent enum class, but thanks to static imports (which work even on a JDK 1.4 runtime) we can save some characters by dropping the Planet prefix:

```

import static Planet.*

SATURN.printMe()

```

No more Planet prefix. But of course, static imports aren't only available for enums, but also work for other classes and static fields. What about doing some math?

```
import static java.lang.Math.*
```

```
assert sin(PI / 6) + cos(PI / 3) == 1
```

Both the static methods of `java.lang.Math` and its static constants were statically imported to make the expression more concise. But if the abbreviations of sine and cosine are not readable for you, you can use aliasing in Groovy with the `as` keyword:

```
import static java.lang.Math.PI  
import static java.lang.Math.sin as sine  
import static java.lang.Math.cos as cosine
```

```
assert sine(PI / 6) + cosine(PI / 3) == 1
```

Aliasing also works with normal imports too, not just static imports, and it can be pretty handy for adding some shortcut notation to very long class names as found in many frameworks, or for renaming methods or constants with non-obvious names, or not following your naming convention standards.

### ***Generics***

A somewhat more controversial feature of Java 5 also finds its way in the latest release of Groovy 1.5 with Generics. Initially, it may feel odd to add even more typing information to a dynamic language, after all. Java developers usually believe that because of type erasure (for backward compatibility reasons with older versions of Java) no information is left in the class bytecode to represent the generic type. However, this is wrong, since through the reflection API, you are able to introspect a class to discover the types of its fields or of its methods arguments with the generics details.

So for instance, when you declare a field of type `List<String>`, somewhere in the bytecode, this information is kept in the form of some meta-information, although this field is really just of type `List`. This kind of reflexive information is used by enterprise frameworks like JPA or Hibernate to be able to relate a collection of elements from an entity to the entity representing the type of these elements.

To put this into practice, let us check if the generics information is kept on class fields:

```

class Talk {
String title
}

class Speaker {
String name
List<Talk> talks = []
}

def me = new Speaker(
name: 'Guillaume Laforge',
talks: [
new Talk(title: 'Groovy'),
new Talk(title: 'Grails')
])

def talksField =
me.class.getDeclaredField('talks')
assert talksField.genericType.toString() ==
'java.util.Listt<Talk>'

```

We defined two classes: a `Speaker` class giving `Talks` at conferences. In the `Speaker` class, the `talks` property is of type `List<Talk>`. Then, we create a `Speaker` instance with two nice shortcuts for initializing the `name` and `talks` properties, and for creating a list of `Talk` instances. Once this setup code is ready, we retrieve the field representing the `talks`, and we check that the generic type information is correct: yes, `talks` is a `List`, but a `List` of `Talks`.

#### *Covariant return types*

In Java 5, if you have a method in a subclass with the same name and parameter types as in a parent class, but with a return type derived from the parent method's return type, then we can override the parent method. In Groovy 1.0, covariant return types were not supported. But in Groovy 1.5, you can use them. Additionally, if you are trying to override a method with a return type not deriving from the parent class method's return type, a compilation error will be thrown. Covariant return types also work with parameterized types.

Beyond the support of Java 5 features which brought a few additions to the language, a few other syntax enhancements have been introduced in Groovy 1.5, and we are going to discover them in the following section.

#### **Syntax additions**

##### *Elvis operator*

Apart from the Java 5 features that brought annotations, generics and enums into Groovy, a new operator finds its way into the language: `?:` the Elvis operator. When you see the operator in question, you will easily guessed why it

was nicknamed that way - if not, think in terms of Smiley. This new operator is, in fact, a shortcut notation for the ternary operator. How many times are you using the ternary operator to change the value of a variable if its content is null to assign it some default value? The typical case in Java is as follows:

```
String name = "Guillaume";  
String displayName = name != null ? name :  
"Unknown";
```

In Groovy, since the language is able to "coerce" types to boolean values as needed (for instance where conditional expressions are required like in if or while constructs), in this statement, we can omit the comparison to null, because when a String is null, it is coerced to false, so in Groovy, the statement would become:

```
String name = "Guillaume"  
String displayName = name ? name : "Unknown"
```

However, you will still notice the repetition of the name variable, which would violate the DRY principle (Don't Repeat Yourself). As this construct is pretty common, the Elvis operator was introduced to simplify such recurring cases, and the statements become:

```
String name = "Guillaume"  
String displayName = name ?: "Unknown"
```

The second occurrence of the name variable is simply omitted and the ternary operator is no more ternary and is shortened to this more concise form.

It is also worth noticing that there are no side effects to this new construct, since the first element (here the name) is not evaluated twice as it would be the case with the ternary operator, so there's no need to introduce some intermediate temporary value to hold the result of the first evaluation of the first element of the ternary operator.

#### *Classical for loop*

Although Groovy is not strictly speaking a 100% superset of Java, the Groovy syntax comes closer to the Java syntax after each release, and more and more Java code is also valid Groovy. The net benefit of this is that when you begin with Groovy, you can copy and paste Java code in your Groovy classes, and this should just work as expected. Then, over time, as you learn the language, you start throwing away the semi-colons which are not mandatory in Groovy, using GStrings (interpolated strings), or closures, etc. Groovy offers a rather flat learning curve to Java developers.

However, there was one omission to this Java-syntax compatibility in the fact the classical for loop inherited from Java's C background wasn't allowed in Groovy. Initially, the Groovy developers thought it was not the nicest syntax of all and preferred using the for / in construct which was more readable. But as the Groovy users regularly asked for this old construct to be also part of Groovy, the team decided to bring it back to Groovy.

With Groovy 1.5, you can either chose the Groovy for / in, or prefer the classical for loop:

```
for (i in 0..9)  
println i
```

```
for (int i = 0; i < 10; i++)  
println i
```

At the end of the day, it is probably more a matter of taste, and long time Groovy users usually prefer the most concise syntax with the `for / in` loop instead.

### *Named parameters without parenthesis*

With its malleable and concise syntax, and its advanced dynamic capabilities, **Groovy is an ideal choice for implementing internal Domain-Specific Languages**. When you want to share a common metaphor between subject matter experts and developers, you can leverage Groovy to create a dedicated business language which models the key concept and business rules of your application. An important aspect of these DSLs is to make the code very readable, and also easier to write by non-technical persons. To achieve this goal even further, the grammar of the language was tweaked to allow us to use named parameters without the surrounding parenthesis.

First of all, in Groovy, named parameters look like this:

```
fund.compare(to: benchmarkFund, in: euros)
compare(fund: someFund, to: benchmark, in:
euros)
```

By adding new properties to numbers - which is possible in Groovy but beyond the scope of this article - we can also write code like this:

```
monster.move(left: 3.meters, at: 5.mph)
```

Now by omitting parenthesis, the code can become a little clearer as shown below:

```
fund.compare to: benchmarkFund, in: euros
compare fund: someFund, to: benchmark, in:
euros
monster.move left: 3.meters, at: 5.mph
```

Obviously, this is not a big difference, but each statement becomes closer to real plain English sentences, and removes the usual boiler-plate technical code of the host language. This little enhancement of the grammar of the Groovy language gives more options to the designers of the business DSL.

### **Improved tooling support**

A common show-stopper when Groovy was a young language, was the lack of good tooling support: both the tool chain and the IDE support weren't up to the task. Fortunately, with the maturity and success of Groovy and the Grails web framework, this situation has changed.

### **Introduction of the "joint" compiler**

Groovy is well-known for its transparent and seamless integration with Java. But this is not just about being able to call Java methods from Groovy scripts, no, the integration between both languages goes well beyond that. For instance, it is totally possible to have a Groovy class extending a Java class which in turns implements a Groovy interface, or vice versa. This is something other alternative languages don't always support, unfortunately. However, when mixing Groovy and Java classes together, so far, you had to be careful when compiling both type of classes by cleverly choosing the order of compilation, and when cyclic dependencies were spanning both languages, you may have hit a "chicken and egg" problem. Fortunately with Groovy 1.5, this is not the case anymore, and thanks to a contribution from JetBrains, the makers of the award winning [Java IDE IntelliJ IDEA](#), a "joint" compiler is available

with which you can compile both Groovy and Java sources together in one go without having to think about dependencies between classes.

If you want to use the joint compiler from the command-line, you can call the `groovyc` command as usual, but specify the `-j` flag which will enable the joint compilation:

```
groovyc *.groovy *.java -j -Jsource=1.4  
-Jtarget=1.4
```

For passing parameters to the underlying `javac` command, you can prefix the flags with the `J` prefix. You can also use the joint compiler through its Ant task from your Ant or Maven build files:

```
<taskdef name="groovyc"  
classname="org.codehaus.groovy.ant.Groovyc"  
classpathref="my.classpath" />
```

```
<groovyc  
srcdir="${mainSourceDirectory}"  
destdir="${mainClassesDirectory}"  
classpathref="my.classpath"  
jointCompilationOptions="-j -Jsource=1.4  
-Jtarget=1.4" />
```

#### *Maven plugin for Groovy*

For the Maven users, there is also a fully featured Maven plugin hosted at Codehaus which allows you to build your Java / Groovy applications: compile your Groovy and Java code, generate documentation from the JavaDoc tags, or it even lets you write your own Maven plugins in Groovy. There is also a Maven archetype to bootstrap your Groovy project more rapidly. For more information, you may have a look at the documentation of the plugin: <http://mojo.codehaus.org/groovy/index.html>

#### *The GroovyDoc documentation tool*

As a Java developer, you are used to documenting your code through JavaDoc tags in the comments of your classes, interfaces, fields, or methods. In Groovy, you can also use such tags in your comments, and have them being used by a tool called GroovyDoc to generate the equivalent JavaDoc documentation for all your Groovy classes.

There's an Ant task you can define then use to generate the documentation as follows:

```
<taskdef name="groovydoc"
classname="org.codehaus.groovy.ant.Groovydoc">
<classpath>
<path path="{mainClassesDirectory}" />
<path refid="compilePath" />
</classpath>
</taskdef>
```

```
<groovydoc
destdir="{docsDirectory}/gapi"
sourcepath="{mainSourceDirectory}"
packagenames="*. *" use="true"
windowtitle="Groovydoc" private="false" />
```

### New interactive shell and the Swing console

Groovy distributions always contained two different shells: a command-line shell as well as a Swing console. Groovysh, the command-line shell, has never been very friendly in terms of interactions with its user: whenever you wanted to execute a statement, you had to type 'go' or 'execute' after each one, so that it got executed. For quick prototyping or playing with some new API, typing 'go' each time was very cumbersome. The situation has changed in Groovy 1.5, since a new interactive shell is born. No need to type 'go' anymore.

This new shell features several enhancements, such as the use of the JLine library which provides ANSI coloring, tab completion for commands, line editing capabilities. You can work with different script buffers, remember already imported classes, load existing scripts, save the current script to a file, browse the history, etc. For detailed explanations of the supported features, please have a look at the [documentation](#).

The command-line shell isn't the only one that received some care, the Swing console has also been improved, with a new toolbar, with advanced undo capabilities, the possibility to increase or decrease the font size, syntax highlighting. A lot of polishing has been applied to the console.

### *IntelliJ IDEA JetGroovy plugin*

I will save the best of tooling support till the end of this section by mentioning the JetGroovy plugin: a free and Open Source IntelliJ IDEA plugin dedicated to the support of both Groovy and Grails. This plugin has been developed by JetBrains themselves, and provides unmatched support for both the language and the web framework.

To list a few of the available features of the Groovy support:

- **Syntax highlighting** for all the syntax, plus different warnings for types not recognized, or when static type information is not known to help spot potential errors.
- Ability to **run Groovy classes, scripts and JUnit test cases written in Groovy**.
- **Debugger**: You can debug step by step across your Java and Groovy code, set breakpoints, show variables, the current stack, etc.
- Joint compiler: the compiler **compiles both Groovy and Java** classes together and is able to resolve dependencies between both languages.
- **Code completion** for packages, classes, properties, fields, variables, methods, keywords, and even specific

- support for the Swing UI builder.
- Advanced class search and find usage.
- **Refactorings**: most of the usual refactorings you've come to love in Java are available as well and work across Java and Groovy, like "surround with", introduce, inline or rename a variable, renaming for packages, classes, methods, and fields.
- **Imports optimizations and code formatting**.
- Structure view: to have a bird's eye view of your classes.

At the end of the day, you won't even notice whether you're developing a class in Groovy or in Java, considering the level of interplay and support offered inside IntelliJ IDEA. This is definitely a plugin to install if you're considering adding some dose of Groovy in your Java project, or if you plan to develop Grails applications.

More information can be found on [JetBrains website](#).

Although I'm only mentioning the plugin for IntelliJ IDEA, for your Groovy developments, you don't have to change your habits. You can also use the Eclipse plugin which is regularly improved by the IBM Project Zero developers, or Sun's brand new support of [Groovy and Grails in NetBeans](#).

## Performance improvements

Along with new features, this new release of Groovy brings noticeable performance improvements as well as lower memory consumption, compared to previous versions. In our informal benchmarks measuring the duration of all our test suites, we noticed speed improvements between 15% to 45% compared with our beta versions of Groovy 1.5 - and certainly higher figures can be expected by comparing with the now old Groovy 1.0. More formal benchmarks have yet to be developed, but those figures have also been confirmed by developers from an insurance company that is using Groovy to write the business rules of their policy risk calculation engine, and from another company who ran several tests on highly parallel machines. Overall, Groovy 1.5 should be faster and leaner in most situations. Your mileage may vary depending on your own context of usage of Groovy.

## Enhanced dynamic capabilities

Through the symbiotic relationships between the Groovy and Grails projects, new dynamic capabilities have been introduced in Groovy, after having matured in the heart of Grails.

Groovy is a dynamic language: put simply, it means that certain things like method dispatch happens at runtime, instead of at compile-time as it is the case of Java and other languages. There is a specific runtime system, called the MOP (stands for Meta-Object Protocol) that is responsible for the dispatching logic. Fortunately, this runtime system is open enough so that people can hook into the system and change its usual behavior. For each Java class and for each Groovy instance, there is an associated meta-class which represents this runtime behavior of your objects. Groovy offered various ways to interact with the MOP by defining custom meta-classes by extending some base class, but thanks to the contribution from the Grails project, a groovier kind of meta-class is available: the `expando` meta-class.

Again, code samples are easier to help us understand the concept. In the following example, the `msg` String instance has got a meta-class that we can access through the `metaClass` property. Then we change the meta-class of the **String** class to add a new method called `up` to the String class, to have a shortcut notation for the **toUpperCase()** method. To do so, we assign a closure to the `up` property of the meta-class which is created as we assign it the closure. This closure takes no argument (hence why it starts with an arrow), and we call the **toUpperCase()** method on the delegate of the closure, which is a special closure variable which represents the real object (here the String instance).

```
def msg = "Hello!"
println msg.metaClass
```

```
String.metaClass.up = { ->
delegate.toUpperCase() }
assert "HELLO!" == msg.up()
```

Through this meta-class, you can query the methods and/or properties which are available:

```
// print all the methods
obj.metaClass.methods.each { println it.name }
// print all the properties
obj.metaClass.properties.each { println it.name
}
```

You can even check that a certain method or property is available, with a finer granularity than through any instanceof check:

```
def msg = 'Hello!'
if (msg.metaClass.respondsTo(msg,
'toUpperCase')) {
println msg.toUpperCase()
}
```

```
if (msg.metaClass.hasProperty(msg, 'bytes')) {
println foo.bytes.encodeBase64()
}
```

These mechanisms are extensively used in the Grails web framework for instance to create dynamic finders: no need for DAOs in most circumstances, as you are able to call a **findByTitle()** dynamic method on a Book domain class. Through meta-classes, Grails automatically decorates the domain classes with such methods. Furthermore, if the method doesn't exist yet, it will be created and cached on first use. This can be accomplished by other advanced hooks as explained below.

Beyond those examples we've seen so far, expando meta-classes also provide some complementary hooks. Four other kind of methods can be added to an expando meta-class:

- **invokeMethod()** lets you intercept all methods calls,
- while **methodMissing()** will be called on last resort only if no other method is found.
- **get/setProperty()** intercepts access to all properties,
- whereas **propertyMissing()** is called when no property can be found.

With expando meta-classes, customizing the behavior of the types of your application becomes easier and can save precious time of development compared with the previous version of Groovy. Obviously, not everybody needs to use

those techniques, but they can be handy in a number of situations where you want to apply some AOP (Aspect Oriented Techniques) to decorate your classes, and when you want to simplify and make more readable the business code of your application, by removing some unnecessary boiler-plate code.

### Swing on steroids

The Groovy project has the chance to have a team of talented Swing developers who worked hard to enhance the capabilities of Groovy to build user interfaces in Swing. The basic brick for building Swing UIs in Groovy is the `SwingBuilder` class: at a syntactical level in your source, you can visually see how Swing components are nested within each other. A simplistic example from the Groovy website shows how to simply create a little GUI:

```
import groovy.swing.SwingBuilder
import java.awt.BorderLayout

import groovy.swing.SwingBuilder
import java.awt.BorderLayout as BL

def swing = new SwingBuilder()
count = 0
def textlabel
def frame = swing.frame(title: 'Frame',
size:[300,300]) {
borderLayout()
textlabel = label(text:"Clicked ${count}
time(s).",
constraints: BL.NORTH)
button(text:'Click Me',
actionPerformed: {count++; textlabel.text =
"Clicked ${count} time(s)."; println
"clicked"},
constraints:BorderLayout.SOUTH)
}
frame.pack()
frame.show()
```

In the novelties, the Swing builder concept has been extend to provide custom component factories. There are additional modules, not bundled with Groovy by default, which integrates the Swing components from JIDE or from the SwingX project into the usual Swing builder code.

Although this topic would deserve a full article, I'm only going to list some of the other improvements in this release, for instance, the `bind()` method. Inspired by the beans binding JSR (JSR-295), you can easily bind components or beans together to have them react upon changes made on each other. In the following example, the size of the

insets of the button will be changed according to the value of the slider component:

```
import groovy.swing.SwingBuilder
import java.awt.Insets

swing = new SwingBuilder()
frame = swing.frame {
  vbox {
    slider(id: 'slider', value:5)
    button('Big Button?!', margin:
bind(source: slider,
sourceProperty: 'value',
converter: { [it, it, it, it] as Insets })))
  }
}
frame.pack()
frame.size = [frame.width + 200, frame.height +
200]
frame.show()
```

Binding components together is such a common task when building user interfaces, that this task has been simplified through this binding mechanism. There are also some other automatic binding options that can be used, but again, a dedicated article would probably be better.

In other new and noteworthy features, a few handy new methods have been added which leverage closures to call the infamous `SwingUtilities` class, and to start new threads: `edt()` will call `invokeAndWait()`, while `doLater()` will call `invokeLater()`, and `doOutside()` will just launch a closure in a new thread. No more ugly anonymous inner classes: just use closures through those shortcut methods!

Last but not least, separating the description of the view and its associated behavior logic has never been easier, thanks to the `build()` method on `SwingBuilder`. You can create a separate script which only contains the view, while the interactions or bindings between components are in the main class, making a clearer separation in the MVC model.

## Summary

In this article, the new and noteworthy features have been outlined, but we have barely scratched the surface of this new version of Groovy. The big highlights are mainly around the new Java 5 features, such as annotations, enums or generics: it makes Groovy perfectly capable of being integrated nicely and seamlessly with Enterprise frameworks such as Spring, Hibernate, or JPA. With the improvements in the syntax and with the enhanced dynamic capabilities, Groovy will let you customize your business logic by creating embedded Domain-Specific Languages, that you can easily integrate at the extension points of your application. The developer experience has progressed significantly through the work poured in the tooling support, this is no more a show stopper to the adoption of Groovy. Overall, with Groovy 1.5, the goal of simplifying the life of developers has never been so well fulfilled, and Groovy should definitely be part of all the Java developers' toolbox.

## Groovy 1.6 release notes

### InfoQ

Article originally [published on InfoQ](#)

[Groovy](#) is a very successful and powerful **dynamic language for the Java Virtual Machine** that provides **seamless integration with Java**, and has its roots firmly planted in Java itself for the syntax and APIs and other languages such as Smalltalk, Python or Ruby for its dynamic capabilities.

*Groovy is used in many Open Source projects such as [Grails](#), [Spring](#), [JBoss Seam](#) and more, as well as **integrated in commercial products** and **Fortune 500 mission-critical applications** for its scripting capabilities offering a nice extension mechanism to these applications, or for its ability to let subject matter experts and developers author embedded Domain-Specific Languages to **express business concepts in a readable and maintainable fashion**.*

In this article, [Guillaume Laforge](#), Groovy Project Manager and Head of Groovy Development at [SpringSource](#), will go through an overview of the novelties offered by the newly released Groovy 1.6.

### Overview of Groovy 1.6

As we shall see in this article, the main highlights of this Groovy 1.6 release are:

- **Greater compile-time and runtime performance improvements**
- Multiple assignments
- Optional return in `if/else` and `try/catch` blocks
- Java 5 annotation definition
- **AST transformations** and all the provided transformation annotations like `@Singleton`, `@Lazy`, `@Immutable`, `@Delegate` and friends
- The Grape module and dependency system and its `@Grab` transformation
- Various Swing builder improvements, thanks to the Swing / [Griffon](#) team, as well as several Swing console improvements
- The integration of **JMX builder**
- Various **metaprogramming improvements**, like the EMC DSL, per-instance metaclasses even for POJOs, and runtime mixins
- **JSR-223** scripting engine built-in
- Out-of-the-box **OSGi readiness**

All those improvements and new features serve one goal: **helping developers be more productive and more agile**, by:

- Focusing more on the task at hand than on boiler-plate technical code
- Leveraging existing Enterprise APIs rather than reinventing the wheel
- Improving the overall performance and quality of the language
- Enabling developers to customize the language at will to derive their own Domain-Specific Languages

But beyond all these important aspects, **Groovy is not just a language, it's a whole ecosystem**.

The improvements in Groovy's generated bytecode information helps capable tools coverage like [Cobertura](#) and its Groovy support, or pave the way for new utilities like [CodeNarc](#) for static code analysis for Groovy.

The malleability of the syntax of the language and its metaprogramming capabilities give birth to advanced testing tools such as the [Easyb](#) Behavior-Driven-Development project, the [GMock](#) mocking library or the [Spock](#) testing and specification framework.

Again, Groovy's flexibility and expressivity and its scripting capabilities open the doors to advanced build scripting or infrastructure systems for your continuous integration practices and project build solutions, such as [Gant](#) and [Graddle](#).

At the tooling level, Groovy also progresses, for instance with its `groovydoc` Ant task to let you generate proper JavaDoc covering, documenting and interlinking both your Groovy and Java source files for your Groovy/Java mixed projects.

And at the same time, IDE makers improve their support for Groovy, by giving users powerful weapons such as cross-language code refactoring, profound understanding of dynamic language idioms, code completion, and more, to make developers productive when using Groovy in their projects.

Now, armed with this knowledge of the Groovy world, it's time to dive into the novelties of Groovy 1.6!

### Performance improvements

A lot of care has been taken to improve both the compile-time and runtime performance of Groovy, compared to previous releases.

The **compiler is 3 to 5 times faster** than in previous releases. This improvement has also been backported in the 1.5.x branch, so that both the old maintenance branch and the current stable branch benefit from this work. Thanks to class lookup caches, the bigger the project, the faster the compilation will be.

However, the most noticeable changes will be in the general runtime performance improvements of Groovy. We used several benchmarks from the [Great Language Shootout](#) to measure our progress. On those we selected, compared to the old Groovy 1.5.x line, the **performance improvements ranged from 150% to 460%**. Micro-benchmarks obviously rarely reflect the kind of code you have in your own projects, but the overall performance of your projects should improve significantly.

### Multiple assignments

In Groovy 1.6, there is only one syntax addition for being able to define and assign several variables at once:

```
def (a, b) = [1, 2]
```

```
assert a == 1
```

```
assert b == 2
```

A more meaningful example may be methods returning longitude and latitude coordinates. If these coordinates are represented as a list of two elements, you can easily get back to each element as follows:

```
def geocode(String location) {
    // implementation returns [48.824068,
    2.531733] for Paris, France
}

def (lat, long) = geocode("Paris, France")

assert lat == 48.824068
assert long == 2.531733
```

And you can also define the types of the variables in one shot as follows:

```
def (int i, String s) = [1, 'Groovy']

assert i == 1
assert s == 'Groovy'
```

For the assignment (with prior definition of the variables), just omit the `def` keyword:

```
def firstname, lastname

(firstname, lastname) = "Guillaume
Laforge".tokenize()

assert firstname == "Guillaume"
assert lastname == "Laforge"
```

If the list on the right-hand side contains more elements than the number of variables on the left-hand side, only the first elements will be assigned in order into the variables. Also, when there are less elements than variables, the extra variables will be assigned null.

So for the case with more variables than list elements, here, `c` will be `null`:

```
def elements = [1, 2]
def (a, b, c) = elements
```

```
assert a == 1
assert b == 2
assert c == null
```

Whereas in the case where there are more list elements than variables, we'll get the following expectations:

```
def elements = [1, 2, 3, 4]
def (a, b, c) = elements
```

```
assert a == 1
assert b == 2
assert c == 3
```

For the curious minds, supporting multiple assignments also means we can do the standard school swap case in one line:

```
// given those two variables
def a = 1, b = 2

// swap variables with a list
(a, b) = [b, a]

assert a == 2
assert b == 1
```

#### Annotation definition

Actually, when I said that multiple assignments were the sole syntax addition, it's not entirely true. Groovy supported the syntax for annotation definition even in Groovy 1.5, but we had not implemented the feature completely. Fortunately, this is now fixed, and it wraps up all the Java 5 features supported by Groovy, such as **static imports**, **generics**, **annotations**, and **enums**, making Groovy the **sole alternative dynamic language for the JVM supporting all those Java 5 features**, which is critical for a seamless Java integration story, and for the usage in Enterprise frameworks relying on annotations, generics and more, like JPA, EJB3, Spring, TestNG, etc.

## Optional return for if/else and try/catch/finally blocks

It is now possible for `if/else` and `try/catch/finally` blocks to return a value when they are the last expression in a method or a closure. No need to explicitly use the `return` keyword inside these constructs, as long as they are the latest expression in the block of code.

As an example, the following method will return 1, although the `return` keyword was omitted.

```
def method() {  
    if (true) 1 else 0  
}
```

```
assert method() == 1
```

For `try/catch/finally` blocks, the last expression evaluated is the one being returned. If an exception is thrown in the `try` block, the last expression in the `catch` block is returned instead. Note that `finally` blocks don't return any value.

```
def method(bool) {  
    try {  
        if (bool) throw new Exception("foo")  
        1  
    } catch(e) {  
        2  
    } finally {  
        3  
    }  
}
```

```
assert method(false) == 1  
assert method(true) == 2
```

## AST Transformations

Although at times, it may sound like a good idea to extend the syntax of Groovy to implement new features (like this is the case for instance for multiple assignments), most of the time, we can't just add a new keyword to the grammar, or create some new syntax construct to represent a new concept. However, with the idea of AST (Abstract Syntax Tree) Transformations, we are able to tackle new and innovative ideas without necessary grammar changes.

When the Groovy compiler compiles Groovy scripts and classes, at some point in the process, the source code will end up being represented in memory in the form of a Concrete Syntax Tree, then transformed into an Abstract Syntax Tree. The purpose of AST Transformations is to let developers hook into the compilation process to be able

to modify the AST before it is turned into bytecode that will be run by the JVM.

**AST Transformations provides Groovy with improved compile-time metaprogramming capabilities** allowing powerful flexibility at the language level, without a runtime performance penalty.

There are two kinds of transformations: global and local transformations.

- Global transformations are applied to by the compiler on the code being compiled, wherever the transformation apply. A JAR added to the classpath of the compiler should contain a service locator file at `META-INF/services/org.codehaus.groovy.transform.ASTTransformation` with a line with the name of the transformation class. The transformation class must have a no-args constructor and implement the `org.codehaus.groovy.transform.ASTTransformation` interface. It will be run against every source in the compilation, so be sure to not create transformations which scan all the AST in an expansive and time-consuming manner, to keep the compiler fast.
- Local transformations are transformations applied locally by annotating code elements you want to transform. For this, we reuse the annotation notation, and those annotations should implement `org.codehaus.groovy.transform.ASTTransformation`. The compiler will discover them and apply the transformation on these code elements.

Groovy 1.6 provides several local transformation annotations, in the Groovy Swing Builder for data binding (`@Bindable` and `@Vetoable`), in the Grape module system for adding script library dependencies (`@Grab`), or as general language features without requiring any syntax change to support them (`@Singleton`, `@Immutable`, `@Delegate`, `@Lazy`, `@Newify`, `@Category`, `@Mixin` and `@PackageScope`). Let's have a look at some of these transformations (`@Bindable` and `@Vetoable` will be covered in the section related to the Swing enhancements, and `@Grab` in the section about Grape).

#### *@Singleton*

Whether the singleton is pattern or an anti-pattern, there are still some cases where we need to create singletons. We're used to create a private constructor, a `getInstance()` method for a static field or even an initialized `public static final` field. So instead of writing code like this in Java:

```
public class T {  
    public static final T instance = new T();  
    private T() {}  
}
```

You just need to annotate your type with the `@Singleton` annotation:

```
@Singleton class T {}
```

The singleton instance can then simply be accessed with `T.instance` (direct public field access).

You can also have the lazy loading approach with an additional annotation parameter:

```
@Singleton(lazy = true) class T {}
```

Would become more or less equivalent to this Groovy class:

```

class T {
    private static volatile T instance
    private T() {}
    static T getInstance () {
        if (instance) {
            instance
        } else {
            synchronized(T) {
                if (instance) {
                    instance
                } else {
                    instance = new T ()
                }
            }
        }
    }
}

```

Lazy or not, once again, to access the instance, simply do `T.instance` (property access, shortcut for `T.getInstance()`).

### *@Immutable*

Immutable objects are ones which don't change after initial creation. Such objects are frequently desirable because they are simple and can be safely shared even in multi-threading contexts. This makes them great for functional and concurrent scenarios. The rules for creating such objects are well-known:

- No mutators (methods that modify internal state)
- Class must be final
- Fields must be private and final
- Defensive copying of mutable components
- `equals()`, `hashCode()` and `toString()` must be implemented in terms of the fields if you want to compare your objects or use them as keys in e.g. maps

Instead of writing a very long Java or Groovy class mimicking this immutability behavior, Groovy lets you just write an immutable class as follow:

```
@Immutable final class Coordinates {  
    Double latitude, longitude  
}
```

```
def c1 = new Coordinates(latitude: 48.824068,  
longitude: 2.531733)  
def c2 = new Coordinates(48.824068, 2.531733)
```

```
assert c1 == c2
```

All the boiler-plate code is generated at compile-time for you! The example shows that to instantiate such immutable coordinates, you can use one of the two constructors created by the transformation, one taking a map whose keys are the properties to set to the values associated with those keys, and the other taking the values of the properties as parameters. The `assert` also shows that `equals()` was implemented and allows us to properly compare such immutable objects.

You can have a look at the [details of the implementation](#) of this transformation. For the record, the Groovy example above using the `@Immutable` transformation is over 50 lines of equivalent Java code.

*@Lazy*

Another transformation is `@Lazy`. Sometimes, you want to handle the initialization of a field of your class lazily, so that its value is computed only on first use, often because it may be time-consuming or memory-expensive to create. The usual approach is to customize the getter of said field, so that it takes care of the initialization when the getter is called the first time. But in Groovy 1.6, you can now use the `@Lazy` annotation for that purpose:

```
class Person {  
    @Lazy pets = ['Cat', 'Dog', 'Bird']  
}
```

```
def p = new Person()  
assert !(p.dump().contains('Cat'))  
  
assert p.pets.size() == 3  
assert p.dump().contains('Cat')
```

In the case of complex computation for initializing the field, you may need to call some method for doing the work, instead of a value like our `pets` list. This is then possible to have the lazy evaluation being done by a closure call, as the following example shows:

```
class Person {  
    @Lazy List pets = { /* complex computation  
here */ }()  
}
```

There is also an option for leveraging Soft references for garbage collection friendliness for expensive data structures that may be contained by such lazy fields:

```
class Person {  
    @Lazy(soft = true) List pets = ['Cat',  
'Dog', 'Bird']  
}
```

```
def p = new Person()  
assert p.pets.contains('Cat')
```

The internal field created by the compiler for `pets` will actually be a Soft reference, but accessing `p.pets` directly will return the value (ie. the list of pets) held by that reference, making the use of the soft reference transparent to the user of that class.

#### *@Delegate*

Java doesn't provide any built-in delegation mechanism, and so far Groovy didn't either. But with the `@Delegate` transformation, a class field or property can be annotated and become an object to which method calls are delegated. In the following example, an `Event` class has a date delegate, and the compiler will delegate all of `Date`'s methods invoked on the `Event` class to the `Date` delegate. As shown in the latest assert, the `Event` class has got a `before(Date)` method, and all of `Date`'s methods.

```

import java.text.SimpleDateFormat

class Event {
    @Delegate Date when
    String title, url
}

def df = new SimpleDateFormat("yyyy/MM/dd")

def gr8conf = new Event(title: "GR8
Conference",
                        url: "http://www.gr8conf.org",
                        when:
df.parse("2009/05/18"))
def javaOne = new Event(title: "JavaOne",
                        url: "http://java.sun.com/javaone/",
                        when:
df.parse("2009/06/02"))

assert gr8conf.before(javaOne.when)

```

The Groovy compiler adds all of `Date`'s methods to the `Event` class, and those methods simply delegate the call to the `Date` field. If the delegate is not a final class, it is even possible to make the `Event` class a subclass of `Date` simply by extending `Date`, as shown below. No need to implement the delegation ourselves by adding each and every `Date` methods to our `Event` class, since the compiler is friendly-enough with us to do the job itself.

```

class Event extends Date {
    @Delegate Date when
    String title, url
}

```

In the case you are delegating to an interface, however, you don't even need to explicitly say you implement the interface of the delegate. The `@Delegate` transformation will take care of this and implement that interface. So the

instances of your class will automatically be instanceof the delegate's interface.

```
import java.util.concurrent.locks.*

class LockableList {
    @Delegate private List list = []
    @Delegate private Lock lock = new Reentrant
Lock()
}

def list = new LockableList()

list.lock()
try {
    list << 'Groovy'
    list << 'Grails'
    list << 'Griffon'
} finally {
    list.unlock()
}

assert list.size() == 3
assert list instanceof Lock
assert list instanceof List
```

In this example, our `LockableList` is now a composite of a list and a lock and is instanceof of `List` and `Lock`. However, if you didn't intend your class to be implementing these interfaces, you would still be able to do so by specifying a parameter on the annotation:

```
@Delegate(interfaces = false) private List list
= []
```

*@Newify*

The `@Newify` transformation proposes two new ways of instantiating classes. The first one is providing Ruby like

approach to creating instances with a `new()` class method:

```
@Newify rubyLikeNew() {  
    assert Integer.new(42) == 42  
}
```

## **rubyLikeNew()**

But it is also possible to follow the Python approach with omitting the `new` keyword. Imagine the following tree creation:

```
class Tree {  
    def elements  
        Tree(Object... elements) { this.elements =  
elements as List }  
}
```

```
class Leaf {  
    def value  
        Leaf(value) { this.value = value }  
}
```

```
def buildTree() {  
    new Tree(new Tree(new Leaf(1), new Leaf(2))  
    , new Leaf(3))  
}
```

## **buildTree()**

The creation of the tree is not very readable because of all those `new` keywords spread across the line. The Ruby approach wouldn't be more readable, since a `new()` method call for creating each element is needed. But by using `@Newify`, we can improve our tree building slightly to make it easier on the eye:

```
@Newify([Tree, Leaf]) buildTree() {  
    Tree(Tree(Leaf(1), Leaf(2)), Leaf(3))  
}
```

You'll also notice that we just allowed `Tree` and `Leaf` to be *newified*. By default, under the scope which is annotated, all instantiations are *newified*, but you can limit the reach by specifying the classes you're interested in. Also, note that for our example, perhaps a Groovy builder may have been more appropriate, since its purpose is to indeed create any kind of hierarchical / tree structure.

If we take another look at our coordinates example from a few sections earlier, using both `@Immutable` and `@Newify` can be interesting for creating a path with a concise but type-safe manner:

```
@Immutable final class Coordinates {  
    Double latitude, longitude  
}
```

```
@Immutable final class Path {  
    Coordinates[] coordinates  
}
```

```
@Newify([Coordinates, Path])  
def build() {  
    Path(  
        Coordinates(48.824068, 2.531733),  
        Coordinates(48.857840, 2.347212),  
        Coordinates(48.858429, 2.342622)  
    )  
}
```

```
assert build().coordinates.size() == 3
```

A closing remark here: since a `Path(Coordinates[] coordinates)` was generated, we can use that constructor in a *varargs* way in Groovy, just as if it had been defined as `Path(Coordinates... coordinates)`.

#### ***@Category and @Mixin***

If you've been using Groovy for a while, you're certainly familiar with the concept of Categories. It's a mechanism to extend existing types (even final classes from the JDK or third-party libraries), to add new methods to them. This is

also a technique which can be used when writing Domain-Specific Languages. Let's consider the example below:

```
final class Distance {  
    def number  
    String toString() { "${number}m" }  
}  
  
class NumberCategory {  
    static Distance getMeters(Number self) {  
        new Distance(number: self)  
    }  
}  
  
use(NumberCategory) {  
    def dist = 300.meters  
  
    assert dist instanceof Distance  
    assert dist.toString() == "300m"  
}
```

We have a simplistic and fictive `Distance` class which may have been provided by a third-party, who had the bad idea of making the class `final` so that nobody could ever extend it in any way. But thanks to a Groovy Category, we are able to decorate the `Distance` type with additional methods. Here, we're going to add a `getMeters()` method to numbers, by actually decorating the `Number` type. By adding a getter to a number, you're able to reference it using the nice property syntax of Groovy. So instead of writing `300.getMeters()`, you're able to write `300.meters`.

The downside of this category system and notation is that to add instance methods to other types, you have to create `static` methods, and furthermore, there's a first argument which represents the instance of the type we're working on. The other arguments are the normal arguments the method will take as parameters. So it may be a bit less intuitive than a normal method definition we would have added to `Distance`, should we have had access to its source code for enhancing it. Here comes the `@Category` annotation, which transforms a class with instance methods into a Groovy category:

```

@Category(Number)
class NumberCategory {
    Distance getMeters() {
        new Distance(number: this)
    }
}

```

No need for declaring the methods `static`, and the `this` you use here is actually the number on which the category will apply, it's not the real `this` of the category instance should we create one. Then to use the category, you can continue to use the `use(Category) {}construct`. What you'll notice however is that these kind of categories only apply to one single type at a time, unlike classical categories which can be applied to any number of types.

Now, pair `@Category` extensions to the `@Mixin` transformation, and you can mix in various behavior in a class, with an approach similar to multiple inheritance:

```

@Category(Vehicle) class FlyingAbility {
    def fly() { "I'm the ${name} and I fly!" }
}

```

```

@Category(Vehicle) class DivingAbility {
    def dive() { "I'm the ${name} and I dive!" }
}
}

```

```

interface Vehicle {
    String getName()
}

```

```

@Mixin(DivingAbility)
class Submarine implements Vehicle {
    String getName() { "Yellow Submarine" }
}

```

```
@Mixin(FlyingAbility)
class Plane implements Vehicle {
    String getName() { "Concorde" }
}
```

```
@Mixin([DivingAbility, FlyingAbility])
class JamesBondVehicle implements Vehicle {
    String getName() { "James Bond's vehicle" }
}
```

```
assert new Plane().fly() ==
    "I'm the Concorde and I fly!"
assert new Submarine().dive() ==
    "I'm the Yellow Submarine and I dive!"
```

```
assert new JamesBondVehicle().fly() ==
    "I'm the James Bond's vehicle and I
fly!"
assert new JamesBondVehicle().dive() ==
```

# "I'm the James Bond's vehicle and I dive!"

You don't inherit from various interfaces and inject the same behavior in each subclass, instead you mixin the categories into your class. Here, our marvelous James Bond vehicle gets the flying and diving capabilities through mixins.

An important point to make here is that unlike `@Delegate` which can *inject* interfaces into the class in which the delegate is declared, `@Mixin` just does runtime mixing — as we shall see in the metaprogramming enhancements further down in this article.

## `@PackageScope`

Groovy's convention for properties is that any *field* without any visibility modifier is exposed as a property, with a getter and a setter transparently generated for you. For instance, this `Person` class exposes a getter `getName()` and a setter `setName()` for a private `name` field:

```
class Person {
    String name
}
```

Which is equivalent to this Java class:

```
public class Person {
    private String name;
    public String getName() { return name; }
    public void setName(name) { this.name =
name; }
}
```

That said, this approach has one drawback in that you don't have the possibility to define a field with package-scope visibility. To be able to expose a field with package-scope visibility, you can now annotate your field with the `@PackageScope` annotation.

## Grape, the Groovy Adaptable / Advanced Packaging Engine

To continue our overview of the AST transformations, we'll now learn more about Grape, a mechanism to add and leverage dependencies in your Groovy scripts. Groovy scripts can require certain libraries: by explicitly saying so in your script with the `@Grab` transformation or with the `Grape.grab()` method call, the runtime will find the needed JARs for you. With Grape, you can easily distribute scripts without their dependencies, and have them downloaded on first use of your script and cached. Under the hood, Grape uses Ivy and Maven repositories containing the libraries you may need in your scripts.

Imagine you want to get the links of all the PDF documents referenced by the Java 5 documentation. You want to parse the HTML page as if it were an XML-compliant document (which it is not) with the Groovy `XmlParser`, so you can use the TagSoup SAX-compliant parser which transforms HTML into well-formed valid XML. You don't even have to mess up with your classpath when running your script, just *grab* the TagSoup library through Grape:

```

import org.ccil.cowan.tagsoup.Parser

// find the PDF links in the Java 1.5.0
documentation
@Grab(group='org.ccil.cowan.tagsoup',
module='tagsoup', version='0.9.7')
def getHtml() {
    def tagsoupParser = new Parser()
    def parser = new XmlParser(tagsoupParser)

parser.parse("http://java.sun.com/j2se/1.5.0/do
wnload-pdf.html")
}

html.body.'**'.a.@href.grep(~/.*\pdf/).each{
println it }

```

For the pleasure of giving another example: let's use the [Jetty servlet container](#) to expose [Groovy templates](#) in a few lines of code:

```

import org.mortbay.jetty.Server
import org.mortbay.jetty.servlet.*
import groovy.servlet.*

@Grab(group = 'org.mortbay.jetty', module =
'jetty-embedded', version = '6.1.0')
def runServer(duration) {
    def server = new Server(8080)
    def context = new Context(server, "/",
Context.SESSIONS);
    context.resourceBase = "."
    context.addServlet(TemplateServlet,
"* .gsp")
    server.start()
    sleep duration
    server.stop()
}

```

## **runServer(10000)**

Grape will download Jetty and its dependencies on first launch of this script, and cache them. We're creating a new Jetty Server on port 8080, then expose Groovy's `TemplateServlet` at the root of the context — Groovy comes with its own powerful template engine mechanism. We start the server and let it run for a certain duration. Each time someone will hit <http://localhost:8080/somepage.gsp>, it will display the `somepage.gsp` template to the user — those template pages should be situated in the same directory as this server script.

Grape can also be used as a method call instead of as an annotation. You can also install, list, resolve dependencies from the command-line using the `grape` command. For [more information on Grape](#), please refer to the documentation.

### **Swing builder improvements**

To wrap up our overview of AST transformations, let's finish by speaking about two transformations very useful to Swing developers: `@Bindable` and `@Vetoable`. When creating Swing UIs, you're often interested in monitoring the changes of value of certain UI elements. For this purpose, the usual approach is to use JavaBeans `PropertyChangeListener`s to be notified when the value of a class field changes. You then end up writing this very common boiler-plate code in your Java beans:

```

import java.beans.PropertyChangeSupport;

```

```
import java.beans.PropertyChangeListener;

public class MyBean {
    private String prop;

    PropertyChangeSupport pcs = new PropertyChangeSupport(this);

    public void addPropertyChangeListener(PropertyChangeListener l) {
        pcs.add(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        pcs.remove(l);
    }

    public String getProp() {
        return prop;
    }

    public void setProp(String prop) {
        pcs.firePropertyChanged("prop", this.prop, this.prop = prop);
    }
}
```

Fortunately, with Groovy and the `@Bindable` annotation, this code can be greatly simplified:

```
class MyBean {  
    @Bindable String prop  
}
```

Now pair that with Groovy's Swing builder `new bind()` method, define a text field and bind its value to a property of your data model:

```
textField text: bind(source: myBeanInstance,  
sourceProperty: 'prop')
```

Or even:

```
textField text: bind { myBeanInstance.prop }
```

The binding also works with simple expressions in the closure, for instance something like this is possible too:

```
bean location: bind { pos.x + ', ' + pos.y }
```

You may also be interested in having a look at [ObservableMap](#) and [ObservableList](#), for a similar mechanism on maps and lists.

Along with `@Bindable`, there's also a `@Vetoable` transformation for when you need to be able to veto some property change. Let's consider a `Trompetist` class, where the performer's name is not allowed to contain the letter 'z':

```

import java.beans.*
import groovy.beans.Vetoable

class Trumpetist {
    @Vetoable String name
}

def me = new Trumpetist()
me.vetoableChange = { PropertyChangeEvent pce
->
    if (pce.newValue.contains('z'))
        throw new PropertyVetoException("The
letter 'z' is not allowed in a name", pce)
}

me.name = "Louis Armstrong"

try {
    me.name = "Dizzy Gillespie"
    assert false: "You should not be able to
set a name with letter 'z' in it."
} catch (PropertyVetoException pve) {
    assert true
}

```

Looking at a more thorough Swing builder example with binding:

```

import groovy.swing.SwingBuilder
import groovy.beans.Bindable
import static javax.swing.JFrame.EXIT_ON_CLOSE

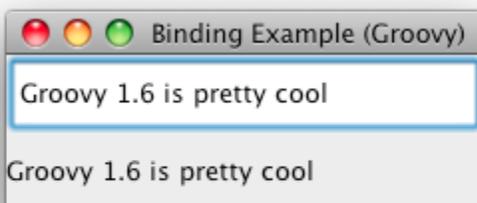
class TextModel {
    @Bindable String text
}

def textModel = new TextModel()

SwingBuilder.build {
    frame( title: 'Binding Example (Groovy)',
size: [240,100], show: true,
        locationRelativeTo: null,
defaultCloseOperation: EXIT_ON_CLOSE ) {
        GridLayout cols: 1, rows: 2
        TextField id: 'textField'
        bean textModel, text: bind{
textField.text }
        label text: bind{ textModel.text }
    }
}

```

Running this script shows up the frame below with a text field and a label below, and the label's text is bound on the text field's content.



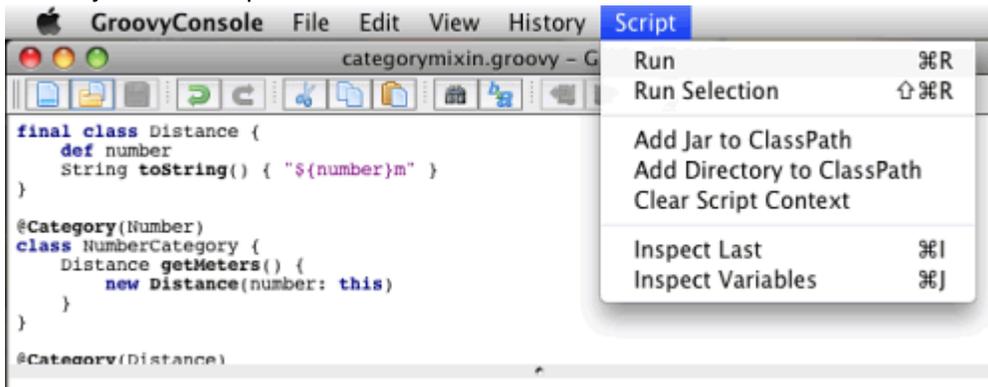
SwingBuilder has evolved so nicely in the past year that the Groovy Swing team decided to launch a new project based on it, and on the Grails foundations: project [Griffon](#) was born. Griffon proposes to bring the *Convention over Configuration* paradigm of Grails, as well as all its project structure, plugin system, gant scripting capabilities, etc.

If you are developing Swing rich clients, make sure to have a look at [Griffon](#).

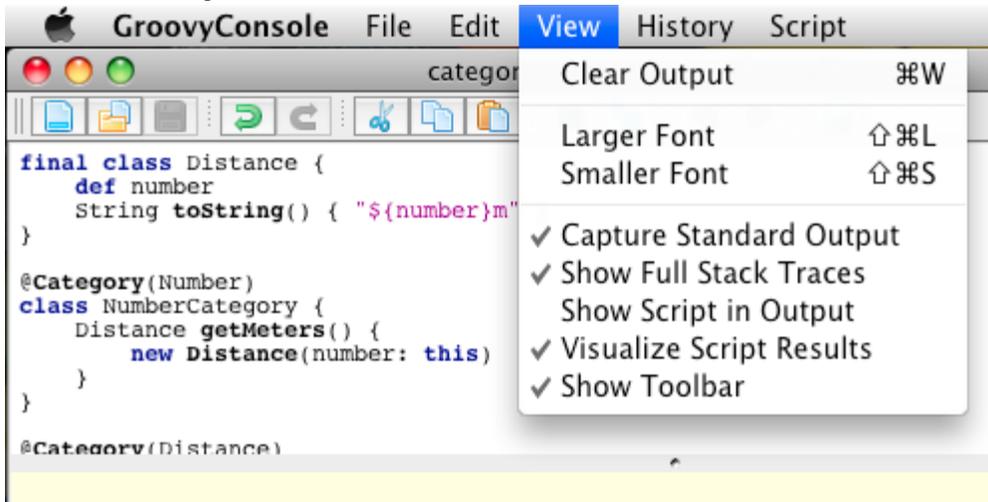
## Swing console improvements

Swinging along the topic of UIs, the Swing console has also evolved:

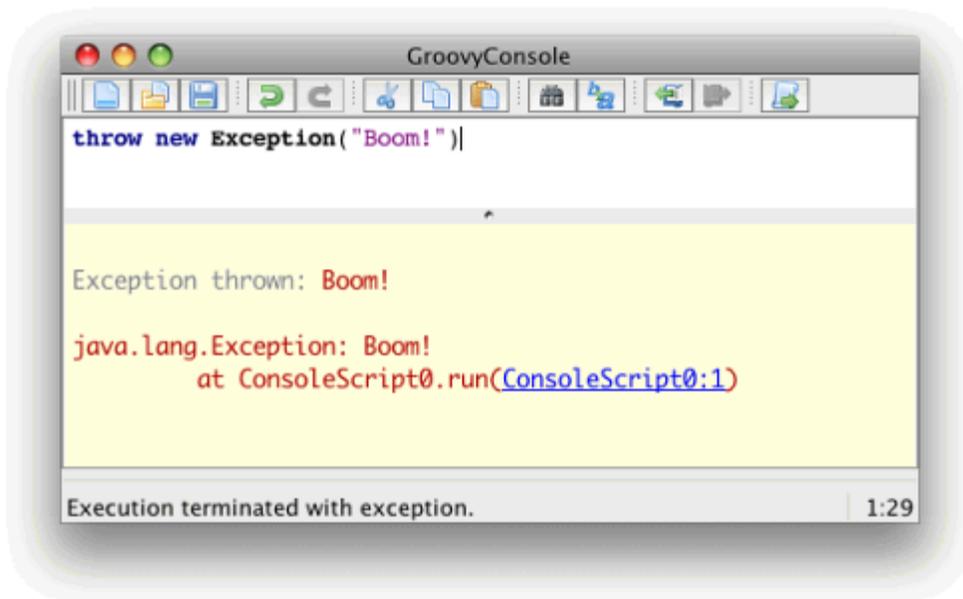
- The console can be run as an Applet (`groovy.ui.ConsoleApplet`).
- Beyond syntax highlighting, the editor also supports code indentation.
- Drag'n dropping a Groovy script over the text area will open the file.
- You can modify the classpath with which the script in the console is being run, by adding a new JAR or a directory to the classpath as shown in the screenshot below.



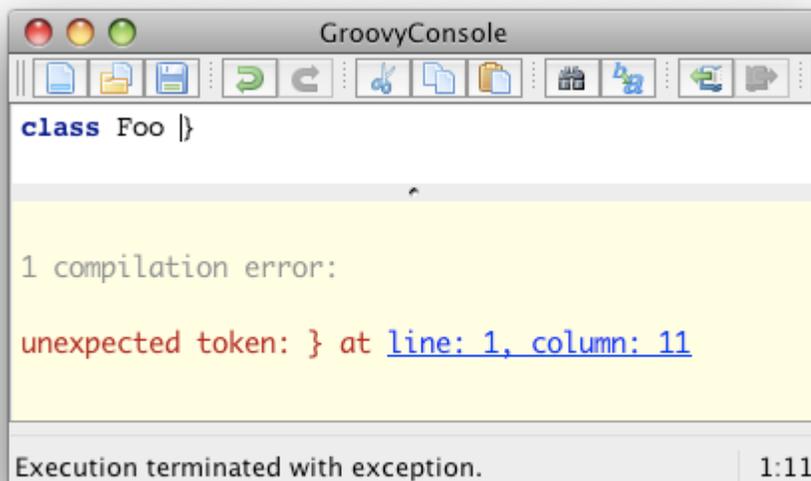
- A couple options have been added to the view menu item: for showing the script being run in the output area, and for visualizing the execution results.



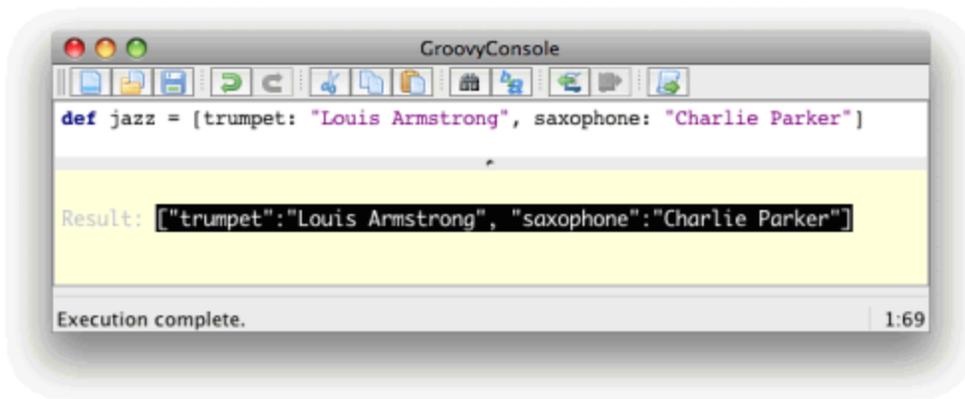
- When an exception is thrown in your script, the lines of the stacktrace relative to your script are clickable, for easy navigation to the point where the error occurred.



- Also, when your script contains compilation errors, the error messages are clickable too.



Back on the visualization of the results in the script output area, a fun system was added to let you customize how certain results are rendered. When you execute a script returning a map of Jazz musicians, you may see something like this in your console:

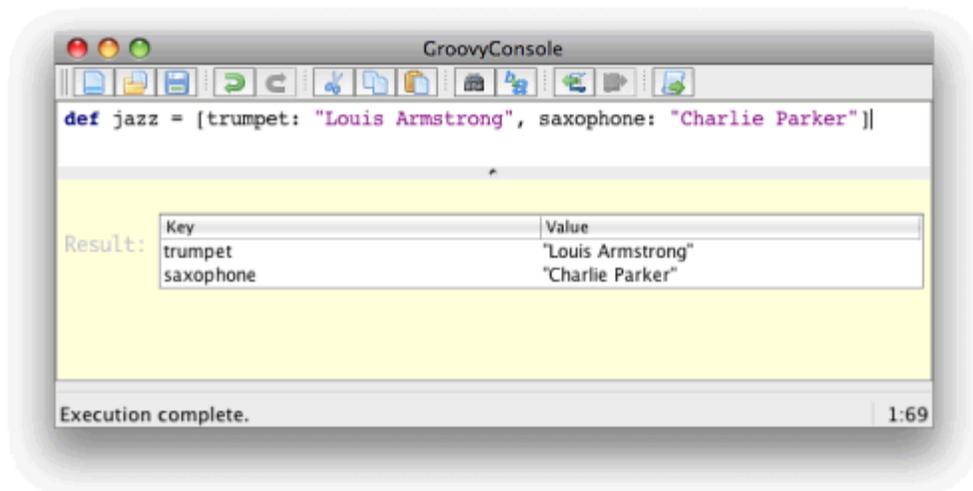


What you see here is the usual textual representation of a Map. But, what if we enabled custom visualization of certain results? The Swing console allows you to do just that. First of all, you have to ensure that the visualization option is ticked: `View -> Visualize Script Results` – for the record, all settings of the Groovy Console are stored and remembered thanks to the Preference API. There are a few result visualizations built-in: if the script returns a `java.awt.Image`, a `javax.swing.Icon`, or a `java.awt.Component` with no parent, the object is displayed instead of its `toString()` representation. Otherwise, everything else is still just represented as text. Now, create the following Groovy script in `~/.groovy/OutputTransforms.groovy`:

```
import javax.swing.*

transforms << { result ->
    if (result instanceof Map) {
        def table = new.JTable(
            result.collect{ k, v ->
                [k, v?.inspect()] as Object[]
            } as Object[][],
            ['Key', 'Value'] as Object[])
        table.preferredViewportSize =
table.preferredSize
        return new JScrollPane(table)
    }
}
```

The Groovy Swing console will execute that script on startup, injecting a `transforms` list in the binding of the script, so that you can add your own script results representations. In our case, we transform the `Map` into a nice-looking Swing `JTable`. And we're now able to visualize maps in a friendly and attractive fashion, as the



screenshot below shows:

The Swing console is obviously not to be confused with a real full-blown IDE, but for daily scripting tasks, the console is a handy tool in your toolbox.

### Metaprogramming enhancements

What makes Groovy a dynamic language is its Meta-Object Protocol and its concept of metaclasses which represent the runtime behavior of your classes and instances. In Groovy 1.6, we continue improving this dynamic runtime system, bringing several new capabilities into the mix.

#### *Per instance metaclass even for POJOs*

So far, Groovy POGOs (Plain Old Groovy Objects) could have a per-instance metaclass, but POJOs could only have one metaclass for all instances (ie. a per-class metaclass). This is now not the case anymore, as POJOs can have a per-instance metaclass too. Also, setting the metaclass property to null will restore the default metaclass.

#### *ExpandoMetaClass DSL*

Initially developed under the [Grails](#) umbrella and integrated back into Groovy 1.5, ExpandoMetaClass is a very handy way for changing the runtime behavior of your objects and classes, instead of writing full-blown `MetaClass` classes. Each time, we want to add / change several properties or methods of an existing type, there is too much of a repetition of `Type.metaClass.xxx`. Take for example this extract of a [Unit manipulation DSL](#) dealing with operator overloading:

```
Number.metaClass.multiply = { Amount amount ->
amount.times(delegate) }
Number.metaClass.div =      { Amount amount ->
amount.inverse().times(delegate) }
```

```
Amount.metaClass.div =      { Number factor ->
delegate.divide(factor) }
Amount.metaClass.div =      { Amount factor ->
delegate.divide(factor) }
Amount.metaClass.multiply = { Number factor ->
delegate.times(factor) }
Amount.metaClass.power =    { Number factor ->
delegate.pow(factor) }
Amount.metaClass.negative = { ->
delegate.opposite() }
```

The repetition, here, looks obvious. But with the `ExpandoMetaClass` DSL, we can streamline the code by regrouping the operators per type:

```

Number.metaClass {
    multiply { Amount amount ->
amount.times(delegate) }
    div      { Amount amount ->
amount.inverse().times(delegate) }
}

```

```

Amount.metaClass {
    div <<   { Number factor ->
delegate.divide(factor) }
    div <<   { Amount factor ->
delegate.divide(factor) }
    multiply { Number factor ->
delegate.times(factor) }
    power    { Number factor ->
delegate.pow(factor) }
    negative { -> delegate.opposite() }
}

```

A `metaClass()` method takes a closure as single argument, containing the various definitions of the methods and properties, instead of repeating the `Type.metaClass` on each line. When there is just one method of a given name, use the pattern `methodName { /* closure */ }`, but when there are several, you should use the append operator and follow the pattern `methodName << { /* closure */ }`. Static methods can also be added through this mechanism, so instead of the classical approach:

```

// add a fqn() method to Class to get the fully
// qualified name of the class (ie. simply
Class#getName)
Class.metaClass.static.fqn = { delegate.name }

```

```

assert String.fqn() == "java.lang.String"

```

You can now do:

```

Class.metaClass {
    'static' {
        fqn { delegate.name }
    }
}

```

Note here that you have to quote the `static` keyword, to avoid this construct to look like a static initializer. For one off method addition, the classical approach is obviously more concise, but when you have several methods to add, the EMC DSL makes sense.

The usual approach for adding properties to existing classes through `ExpandoMetaClass` is to add a getter and a setter as methods. For instance, say you want to add a method that counts the number of words in a text file, you could try this:

```

File.metaClass.getWordCount = {
    delegate.text.split(/\w/).size()
}

```

```

new File('myFile.txt').wordCount

```

When there is some logic inside the getter, this is certainly the best approach, but when you just want to have new properties holding simple values, through the `ExpandoMetaClass` DSL, it is possible to define them. In the following example, a `lastAccessed` property is added to a `Car` class — each instance will have its property. Whenever a method is called on that car, this property is updated with a newer timestamp.

```

class Car {
    void turnOn() {}
    void drive() {}
    void turnOff() {}
}

```

```

Car.metaClass {
    lastAccessed = null
    invokeMethod = { String name, args ->
        def metaMethod =
    delegate.metaClass.getMetaMethod(name, args)
        if (metaMethod) {
            delegate.lastAccessed = new Date()
        }
    }
}

```

```
        metaMethod.doMethodInvoke(delegate,
args)
    } else {
        throw new MissingMethodException(name,
delegate.class, args)
    }
}
}
```

```
def car = new Car()
println "Last accessed: ${car.lastAccessed ? :
'Never'}"
```

```
car.turnOn()
println "Last accessed: ${car.lastAccessed ? :
'Never'}"
```

```
car.drive()
sleep 1000
println "Last accessed: ${car.lastAccessed ? :
'Never'}"
```

```
sleep 1000
car.turnOff()
```

```
println "Last accessed: ${car.lastAccessed ?:  
'Never'}"
```

In our example, in the DSL, we access that property through the `delegate` of the closure, with `delegate.lastAccessed = new Date()`. And we intercept any method call thanks to `invokeMethod()`, delegating to the original method for the call, and throwing an exception in case the method doesn't exist. Later on, you can see by executing this script that `lastAccessed` is updated as soon as we call a method on our instance.

### *Runtime mixins*

Last metaprogramming feature we'll cover today: runtime mixins. `@Mixin` allowed you to mixin new behavior to classes you owned and were designing. But you could not mixin anything to types you didn't own. Runtime mixins propose to fill that gap by letting you add a mixin on any type at runtime. If we think again about our example of vehicles with some mixed-in capabilities, if we didn't *own* James Bond's vehicle and give it some diving ability, we could use this mechanism:

```
// provided by a third-party  
interface Vehicle {  
    String getName()  
}
```

```
// provided by a third-party  
class JamesBondVehicle implements Vehicle {  
    String getName() { "James Bond's vehicle" }  
}
```

```
JamesBondVehicle.mixin DivingAbility,  
FlyingAbility
```

```
assert new JamesBondVehicle().fly() ==  
    "I'm the James Bond's vehicle and I  
fly!"  
assert new JamesBondVehicle().dive() ==  
    "I'm the James Bond's vehicle and I  
dive!"
```

One or more mixins can be passed as argument to the static `mixin()` method added by Groovy on `Class`.

## JSR-223 Groovy Scripting Engine

Before Groovy 1.6, if you wanted to integrate Groovy in your Java projects through JSR-223 / `javax.script.*`, you had to download a Groovy script engine implementation from [java.net](http://java.net), and put the JAR in your classpath. This additional step wasn't very developer friendly, requiring some additional work — the JAR wasn't even provided in the Groovy distribution. Thankfully, 1.6 comes with an implementation of the `javax.script.*` APIs.

Below, you'll find an example evaluating Groovy expressions (the code is in Groovy, but it's straightforward to convert it back to Java):

```
import javax.script.*  
  
def manager = new ScriptEngineManager()  
def engine = manager.getEngineByName("groovy")  
  
assert engine.evaluate("2 + 3") == 5
```

Please note that the `javax.script.*` APIs are available only on Java 6.

## JMX Builder

Originating as an [external Open-Source project](#) hosted on Google Code, JMX Builder has been integrated in Groovy 1.6, to simplify the life of developers needing to interact or expose JMX services. JMX Builder features:

- Domain Specific Language (DSL) for JMX API using Builder pattern
- Simplified JMX API's programmability
- Declaratively expose Java/Groovy objects as JMX managed MBeans
- Support class-embedded or explicit descriptors
- Inherent support for JMX's event model
- Seamlessly create JMX event broadcasters
- Attach event listeners as inline closures
- Use Groovy's dynamic nature to easily react to JMX events notifications
- Provides a flexible registration policy for MBean
- No special interfaces or class path restrictions
- Shields developer from complexity of JMX API
- Exposes attribute, constructors, operations, parameters, and notifications
- Simplifies the creation of connector servers and connector clients
- Support for exporting JMX timers

You can find [more information on JMX Builder](#) and its very extensive coverage of the JMX system. Lots of examples will show you how to create a JMX connector server or client, how to easily export POGOs as JMX managed beans, how to listen to JMX events, and much more.

## Improved OSGi support

The Groovy jar files are released with correct OSGi metadata, so they can be loaded as a bundle into any OSGi compliant container, such as Eclipse Equinox or Apache Felix. You can find [more information on how to use Groovy and OSGi](#) on the Groovy project website. This tutorial will explain how to:

- Load Groovy as an OSGi service
- Write a Groovy OSGi service
- Include the Groovy JAR within a bundle
- Publish a service written in Groovy
- Consume a service from Groovy
- Troubleshoot in case you're encountering any problem along the way

You may also be interested in, for instance, how you can [use different versions of Groovy in your application](#), thanks to OSGi.

## Summary

Groovy continues its march towards the goal of **simplifying the life of developers**, providing various new features and improvements in this new release: AST transformations reducing dramatically the number of lines of code to express certain concerns and patterns and opening the language to developers for further extension, several **metaprogramming enhancements to streamline your code** and let you write **more expressive business rules**, and **support for common enterprise APIs** such as Java 6's scripting APIs, JMX management system, or OSGi's programming model. All of this is done obviously **without compromising on the seamless integration with Java**, and furthermore, with a **level of performance way higher than previous releases**.

## Groovy 1.7 release notes

### Coverage of these notes

The below release notes cover the new features of the Groovy 1.7 final release.

## IDE Support

Before diving directly into the new features in Groovy 1.7, please let me mention the great progress made in terms of IDE support for Groovy (and also for [Griffon](#), [Gradle](#), [Gant](#) or [Grails](#)). All the major IDEs (Eclipse, [IntelliJ IDEA](#), [NetBeans](#)) provide an excellent level of support for the language. Initially, the Eclipse support was the one lacking the most, but thanks to the hard work of the SpringSource Eclipse team, we now have a great environment for developing mixed Java / Groovy applications with features like cross-language refactoring, Groovy-specific code completion, and more.

### New and Improved Eclipse plugin

For more information on the Eclipse support, please have a look at the [Groovy Eclipse plugin home page](#), as well as the notes for the [M1 release](#) and [M2 release](#).

## New features

### Anonymous Inner Classes and Nested Classes

Although oftentimes closures and maps coercion suffice, there are still areas where the lack of Anonymous Inner Classes (AIC) and Nested Classes (NC) can be problematic. That's why we decided to eventually implement AIC and NC in Groovy 1.7.

### Be Careful

The implementation of AIC and NC follows the Java lead, but you should not take out the Java Language Spec and keep shaking the head about things that are different. The implementation done looks much like what we do for `groovy.lang.Closure`, with some benefits and some differences. Accessing private fields and methods for example can become a problem, but on the other hand local variables don't have to be final.

### *Nested Static Classes*

Here's an example of Nested Static Classes:

```
class A {  
    static class B {}  
}  
  
new A.B()
```

The usage of static nested classes is the best supported one. If you absolutely need an inner class, you should make it a static one.

### *Anonymous Inner Classes*

Some other examples, this time for Anonymous Inner Classes:

```
boolean called = false  
  
Timer timer = new Timer()  
timer.schedule(new TimerTask() {  
    void run() {  
        called = true  
    }  
}, 0)  
sleep 100  
  
assert called
```

### **i More information**

If you want to learn a bit more about the cases which are currently supported in 1.7, you can have a look at one of our [unit tests](#) covering this new feature.

#### ***Accessing the Outer Context from a Nested Class***

If you are in a nested class Y and the surrounding class is X, then you can access the variable v of X in Java by X.this.v. Groovy does not support this syntax.

#### ***Creating Instances of Non-Static Inner Classes***

In Java you can do this:

```
public class Y {
    public class X {}
    public X foo() {
        return new X();
    }
    public static X createX(Y y) {
        return y.new X();
    }
}
```

It should be noted that the nested class X needs a reference to the outer class instance of Y. For this Java will create a constructor that takes Y as first parameter in X. This constructor is synthetic, so it won't appear in any code completion.

In case of new X(), like you have it in method foo(), then compiler will then create new X(this) instead. In case of createX the compiler will create new X(y). Groovy does not support this.

Instead Groovy supports giving the instance in like the compiler would do it. That means the code above has to be rewritten as

```
public class Y {
    public class X {}
    public X foo() {
        return new X(this);
    }
    public static X createX(Y y) {
        return new X(y);
    }
}
```

#### Caution

Caution though, Groovy supports calling methods with one parameter without giving an argument. The parameter will then have the value null. Basically the same rules apply to calling a constructor. There is a danger that you will write `new X()` instead of `new X(this)` for example. Since this might also be the regular way we have not yet found a good way to prevent this problem.

## Annotations

Groovy's support of annotations is identical to Java 5 annotations, but we felt that in some cases it would be interesting to be able to add annotations in other places than the usual places (types, fields, methods, parameters, etc.). For instance, in Java, it is impossible to add annotations on imports or packages. Groovy does go beyond and adds support for annotation on imports, packages and variable declarations. We'll take a look at the usage of those extended annotations on Grape.

### *Grape*

The Grape dependency system lets you request dependencies in your scripts, without having to deal with downloading, packaging, or specifying the classpath yourself. To use Grape, we had to use the `@Grab` annotation to "grab" a dependency from a repository (Maven's central repository, for example). The problem was that annotation had to be attached to some allowed elements, ie. the places where annotations can be put in Java. Now, we can put annotations on imports:

```

@Grab(group='net.sf.json-lib',
module='json-lib', version='2.3',
classifier='jdk15')
import net.sf.json.groovy.*

assert new JsonSlurper().parseText(
    new JsonGroovyBuilder().json {
        book(title: "Groovy in Action",
author:"Dierk König et al")
    }.toString()
).book.title == 'Groovy in Action'

```

Another example with @Grab on variable declarations:

```

@Grab('net.sf.json-lib:json-lib:2.3:jdk15')
def builder = new
net.sf.json.groovy.JsonGroovyBuilder()

def books = builder.books {
    book(title: "Groovy in Action", author:
"Dierk Koenig")
}

assert books.toString() ==
'''{"books":{"book":{"title":"Groovy in
Action","author":"Dierk Koenig"}}}'''

```

### **i Remark**

Please note on this one an improvement in the `@Grab` annotation: we provide a shorter version taking just a String as value parameter representing the dependency, in addition to the more verbose example in the previous example. You simply append the group, module, version and classifier together, joined by colons.

A Grape **resolver** was added, so you can specify a remote location where grapes are downloaded from:

```
@GrabResolver(name='restlet.org',  
root='http://maven.restlet.org')  
@Grab(group='org.restlet',  
module='org.restlet', version='1.1.6')  
import org.restlet.Restlet  
// ...
```

### **Power Asserts**

Groovy's "assert" keyword has sometimes been criticized as it's, in a way, limited, as it just checks that the expression it's being passed is true or false. Unlike with testing frameworks such as JUnit/TestNG and the various additional assertion utilities, where you get nicer and more descriptive messages, Groovy's assert would just tell you the expression was false, and would give the value of variables used in the expression, but nothing more. With Power Asserts, initially developed in the [Spock Framework](#), the output of the assert is now much nicer and provides a visual representation of the value of each sub-expressions of the expression being asserted. For example:

```
assert new File('foo.bar') == new  
File('example.txt')
```

Will yield:

## Caught: Assertion failed:

```
assert new File('foo.bar') == new  
File('example.txt')
```

|  
**foo.bar**

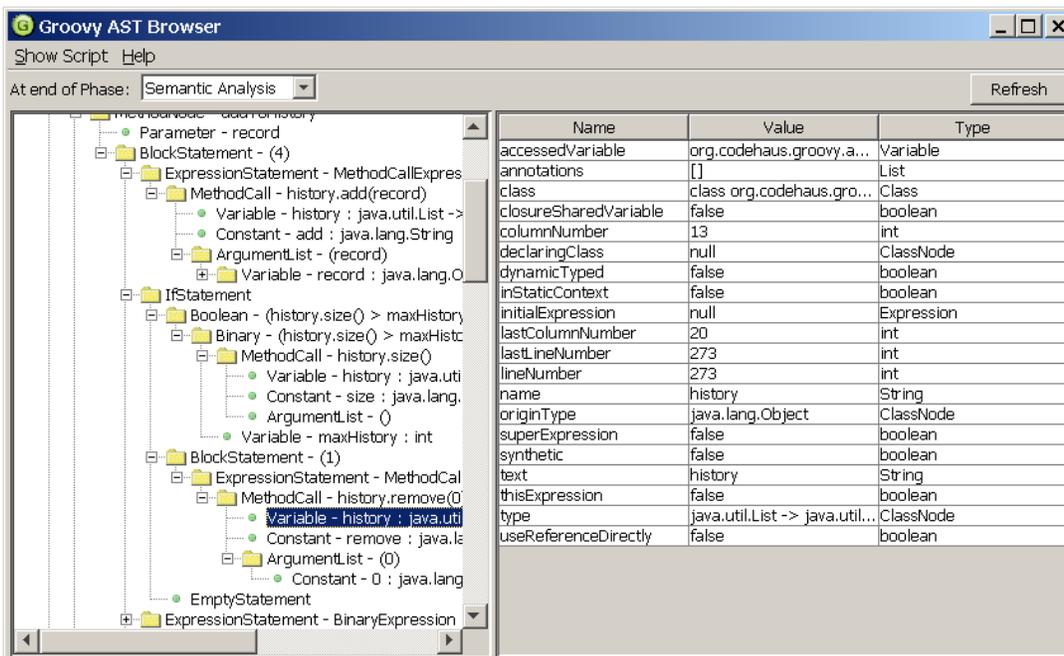
| |  
| **example.txt**  
**false**

## AST

With Groovy 1.6, we introduced AST Transformations, for letting developers do compile-time metaprogramming, by modifying the Abstract Syntax Tree before it is transformed into bytecode. In Groovy 1.6, several such transformations were added, especially "local" transformations triggered by annotations (such as @Delegate, @Singleton, @Bindable and friends). However powerful this feature is, writing AST transformation has always been a bit tedious. Groovy 1.7 features two new features which should help simplify the work of AST transformation writers: an AST viewer and an AST builder.

### AST Viewer

The following screenshot shows a new window that can be launched from the Groovy Swing Console. You can visualize the AST of a script you're working on in the console: for instance, writing the code you'd like to create in your AST transformation. The AST viewer greatly help with figuring out how Groovy builds its AST when compiling your Groovy code.



The screenshot shows the Groovy AST Browser window. The left pane displays a tree view of the AST for the code snippet above. The right pane shows a table of properties for the selected node.

Name	Value	Type
accessedVariable	org.codehaus.groovy.a...	Variable
annotations	[]	List
class	class org.codehaus.gro...	Class
closureSharedVariable	false	boolean
columnNumber	13	int
declaringClass	null	ClassNode
dynamicTyped	false	boolean
inStaticContext	false	boolean
initialExpression	null	Expression
lastColumnNumber	20	int
lastLineNumber	273	int
lineNumber	273	int
name	history	String
originType	java.lang.Object	ClassNode
superExpression	false	boolean
synthetic	false	boolean
text	history	String
thisExpression	false	boolean
type	java.util.List -> java.util...	ClassNode
useReferenceDirectly	false	boolean

## ***AST Builder***

Visualizing the AST is one thing, but we also need a mechanism to create and modify ASTs more easily. The introduction of the AST builder simplifies the authoring of AST transformations, by giving you three different approaches for working on the AST:

- building from string
- building from code
- building from specification

Before the AST builder, one had to create and instantiate manually all the various AST nodes. Let's see how those three forms help with this, for instance for creating a node representing a constant string.

### **Building from string**

```
List<ASTNode> nodes = new  
AstBuilder().buildFromString('' "Hello"  
'')
```

### **Building from code**

```
List<ASTNode> nodes = new  
AstBuilder().buildFromCode { "Hello" }
```

### **Building from specification**

```
List<ASTNode> nodes = new  
AstBuilder().buildFromSpec {  
    block {  
        returnStatement {  
            constant "Hello"  
        }  
    }  
}
```

### **i For more information**

Please have a look at the documentation on the [AST Builder](#). You'll discover the advantages and inconveniences of the various forms, and why all three are needed depending on what you want to achieve with the AST.

## **Other minor enhancements**

### **Ability to customize the Groovy Truth**

In Groovy, booleans aren't the sole things which can be evaluated to true or false, but for instance, null, empty strings or collections are evaluated to false or true if of length > 0 or non-empty. This notion of "truth" was coined "Groovy Truth" in the [Groovy in Action](#) book. With Groovy Truth, instead of doing frequent null checks, you could simply write:

```
def string = "more than one character"  
if (string) { println "the String is neither  
null nor empty" }
```

Up until Groovy 1.7, only a small set of classes had a certain meaning with regards to how they were coerced to a boolean value, but now it is possible to provide a method for coercion to boolean in your own classes. For example, the following Predicate class offers the ability to coerce Predicate instances to true or false, thanks to the implementation of the boolean `asBoolean()` method:

```
class Predicate {  
    boolean value  
    boolean asBoolean() { value }  
}  
  
assert new Predicate(value: true)  
assert !new Predicate(value: false)
```

It is also possible to use categories or `ExpandoMetaClass` to inject an `asBoolean()` method, or to override an existing one (even one on the small set of classes with special Groovy truth behavior).

### **Dependency upgrades**

Some of the dependencies of Groovy have been upgraded to newer versions.

For instance, Groovy now uses the latest ASM version, which is "invokedynamic"-ready. So as we progress towards

the inclusion of JSR-292 / invokedynamic, we'll be ready and be using the latest version of ASM. We also use the latest version of Ivy which is used by the Grape dependency module.

### **Rewrite of the GroovyScriptEngine**

The GroovyScriptEngine (which is also used by Groovlets) has been rewritten to solve various dependency issues it was suffering from, and the outcome of this is that it should also now be much faster overall.

The new logic uses additional phase operations to track dependencies. As a result the error-prone class loader technique to track them is gone now. These operations ensure that every script file will be tracked, its dependencies recorded during compilation and all transitive dependencies will be calculated. And only scripts will be recorded as dependency, no classes. The new GroovyScriptEngine also uses only one compilation "process" for script compilation which solves the problem of circular or mutual dependencies, that caused stack overflows in the past. As a result the new engine can reliably handle dependencies and should be much faster.

### **Groovy console preferences**

A small annoyance, especially for developers using big LCD screens: the Groovy Console didn't remember preferences of position of the separator between the coding area and output view, or the font size being used. This is now fixed, as the console remembers such settings. You won't need anymore to adjust the console to your liking each time you run it, it should now have some more brain cells to remember your preferences.

### **New output window for the Groovy console**

There is a new visualization option for the results of the execution of your scripts in your Groovy Console. Instead of displaying the results in the bottom output pane, it's now possible to use an external window for viewing those results. Run your script with CTRL-R or CMD-R, you will see something like the following screenshot. You can then dismiss the window by hitting Escape, CTRL-W (CMD-W on Macs) or Enter.

```
malariaDsl.groovy * - GroovyConsole
1 class Drug {
2     String name
3     String toString() { name }
4 }
5
6 class DrugQuantity {
7     int number
8     String toString() {
9         number == 1 ? "1 pill" : "$number pills"
10    }
11 }
12
13
14
15
16 Integer.metaClass.getPills = { -> new DrugQuantity
17
18 Number.metaClass.getHours = { -> new TimeDuration
19
20
21 def take(Map m, DrugQuantity dq) {
22     println "Take $dq of $m.of in $m.in.number $
23 }
24
25 def chloroquinine = new Drug(name: "Chloroquinine")
26
27 take 2.pills, of: chloroquinine, in: 6.hours

Execution complete. Result was null. 1:1
```

You will also notice the addition of line numbers in the gutter of the editor area.

## SQL batch updates and transactions

### *Batch updates*

The Groovy Sql class now features batch updates, thanks to its new `withBatch()` method, taking a closure and a statement instance:

```
sql.withBatch { stmt ->
    ["Paul", "Jochen", "Guillaume"].each {
name ->
        stmt.addBatch "insert into PERSON
(name) values ($name)"
    }
}
```

### *Transactions*

Similarly, there's a withTransaction() method added to Sql, which works also with datasets:

```
def persons = sql.dataSet("person")
sql.withTransaction {
    persons.add name: "Paul"
    persons.add name: "Jochen"
    persons.add name: "Guillaume"
}
```

## Groovy 1.8 release notes

### Table of Contents

- [Groovy 1.8 release notes](#)
  - [Command chains for nicer Domain-Specific Languages](#)
  - [Performance improvements](#)
  - [GPars bundled within the Groovy distribution](#)
  - [Closure enhancements](#)
    - [Closure annotation parameters](#)
    - [Closure functional flavors](#)
      - [Closure composition](#)
      - [Closure trampoline](#)
      - [Closure memoization](#)
  - [Currying improvements](#)
- [Native JSON support](#)
  - [Reading JSON](#)
  - [JSON builder](#)
  - [Pretty printing JSON content](#)

- [New AST Transformations](#)
  - [@Log](#)
  - [@Field](#)
  - [@PackageScope enhancements](#)
  - [@AutoClone](#)
  - [@AutoExternalizable](#)
  - [Controlling the execution of your code](#)
    - [@ThreadInterrupt](#)
    - [@TimedInterrupt](#)
    - [@ConditionalInterrupt](#)
  - [@ToString](#)
  - [@EqualsAndHashCode](#)
  - [@TupleConstructor](#)
  - [@Canonical](#)
  - [@InheritConstructors](#)
  - [@WithReadLock and @WithWriteLock](#)
  - [@ListenerList](#)
- [Alignments with JDK 7](#)
  - [Diamond operator](#)
- [New DGM methods](#)
- [Miscellaneous](#)
  - [Slashy strings](#)
  - [Dollar slashy strings](#)
  - [Compilation customizers](#)
  - [\(G\)String to Enum coercion](#)
  - [Maps support isCase\(\)](#)
  - [Grape/Grab Improvements](#)
    - [Shorter notation for @GrabResolver](#)
    - [Compact form for optional Grab attributes](#)
  - [Sql improvements](#)
  - [Storing AST node metadata](#)
  - [Ability to customize the GroovyDoc templates](#)

## Groovy 1.8 release notes

The 1.8 release of Groovy comes with many new features that greatly enhance

- the dynamic expressiveness of Groovy, specifically for defining DSLs
- runtime performance
- concurrent and parallel execution
- design by contract
- functional programming style
- first-class JSON support
- compile-time meta programming
- and more helpers and library additions

These features have undergone the Groovy developer process with formal descriptions, discussion, and voting (GEP - Groovy Enhancement Proposal) for core parts and less formal developer discussions and JIRA voting for additional parts.

Our goal has stayed the same, though: to give the Java developer a tool that makes him more productive, allows

him to achieve his goals faster and with a smaller margin of error, and extend the scalability of the Java platform from full-blown enterprise projects to everyday "getting things done" tasks.

### Command chains for nicer Domain-Specific Languages

Thanks to its flexible syntax and its compile-time and runtime metaprogramming capabilities, Groovy is well known for its Domain-Specific Language capabilities. However, we felt that we could improve upon the syntax further by removing additional punctuation symbols when users chain method calls. This allows DSL implementors to develop command descriptions that read almost like natural sentences.

Before Groovy 1.8, we could omit parentheses around the arguments of a method call for top-level statements. But we couldn't chain method calls. The new "command chain" feature allows us to chain such parentheses-free method calls, requiring neither parentheses around arguments, nor dots between the chained calls. The general idea is that a call like `a b c d` will actually be equivalent to `a(b).c(d)`. This also works with multiple arguments, closure arguments, and even named arguments. Furthermore, such command chains can also appear on the right-hand side of assignments. Let's have a look at some examples supported by this new syntax:

```
turn left then right  
// equivalent to: turn(left).then(right)  
take 2.pills of chloroquinine after 6.hours  
// equivalent to:  
take(2.pills).of(chloroquinine).after(6.hours)  
paint wall with red, green and yellow  
// equivalent to: paint(wall).with(red,  
green).and(yellow)  
  
// with named parameters too  
check that: margarita tastes good  
// equivalent to: check(that:  
margarita).tastes(good)  
  
// with closures as parameters  
given { } when { } then { }  
// equivalent to:  
given({}).when({}).then({})
```

It is also possible to use methods in the chain which take no arguments, but in that case, the parentheses are needed:

```
select all unique() from names  
// equivalent to:  
select(all).unique().from(names)
```

If your command chain contains an odd number of elements, the chain will be composed of method / arguments, and will finish by a final property access:

```
take 3 cookies  
// equivalent to: take(3).cookies  
  
// and also this: take(3).getCookies()
```

This new command chain approach opens up interesting possibilities in terms of the much wider range of DSLs which can now be written in Groovy. This new feature has been developed thanks to the Google Summer of Code program, where our student, Lidia, helped us modify the Groovy Antlr grammar to extend top-level statements to accept that command chain syntax.

The above examples illustrate using a command chain based DSL but not how to create one. You will be able to find some [further examples](#) of "command chains" on the Groovy Web Console but to illustrate creating such a DSL, we will show just a couple of examples - first using maps and Closures:

```

show = { println it }
square_root = { Math.sqrt(it) }

def please(action) {
  [the: { what ->
    [of: { n -> action(what(n)) }]}]
}

please show the square_root of 100
// equivalent to:
please(show).the(square_root).of(100)
// ==> 10.0

```

Or if you prefer Japanese and a metaprogramming style (see [here](#) for more details):

```

// Japanese DSL using GEP3 rules
Object.metaClass. =
Object.metaClass. =
{ clos -> clos(delegate) }

= { it }
= { println it }
= { Math.sqrt(it) }

100 // First, show the square root of
100
// => 10.0

```

As a second example, consider how you might write a DSL for simplifying one of your existing APIs. Maybe you

need to put this code in front of customers, business analysts or testers who might be not hard-core Java developers. We'll use the `splitter` from the Google [Guava libraries](#) project as it already has a nice Fluent API. Here is how we might use it out of the box:

```
@Grab('com.google.guava:guava:r09')
import com.google.common.base.*
def result =
Splitter.on(',').trimResults(CharMatcher.is(
'_ ' as char)).split("_a ,_b_
,c__").iterator().toList()
assert result == ['a ', 'b_ ', 'c']
```

It reads fairly well for a Java developer but if that is not your target audience or you have many such statements to write, it could be considered a little verbose. Again, there are many options for writing a DSL. We'll keep it simple with Maps and Closures. We'll first write a helper method:

```
def split(string) {
  [on: { sep ->
    [trimming: { trimChar ->

Splitter.on(sep).trimResults(CharMatcher.is(
trimChar as
char)).split(string).iterator().toList()
    }]}
  ]}
}
```

now instead of this line from our original example:

```
def result =
  Splitter.on(',').trimResults(CharMatcher.is(
    '_' as char)).split("_a ,_b_
,c__").iterator().toList()
```

we can write this:

```
def result = split "_a ,_b_ ,c__" on ', '
trimming '_'
```

### Performance improvements

Groovy's flexible metaprogramming model involves numerous decision points when making method calls or accessing properties to determine whether any metaprogramming hooks are being utilized. During complex expression calculations, such decision points involved identical checks being executed numerous times. Recent performance improvements allow some of these checks to be bypassed during an expression calculation once certain initial assumptions have been checked. Basically if certain preconditions hold, some streamlining can take place.

Groovy 1.8.0 contains two main streams of optimization work:

- There are several optimizations for basic operations on integers like plus, minus, multiply, increment, decrement and comparisons. This version doesn't support the mixed usage of different types. If an expression contains different types, then it falls back to the classical way of performing the operation, i.e. no streamlining occurs.
- There is also an optimization for direct method calls. Such a method call is done directly if it is done on "this" and if the argument types are a direct match with the parameter types of the method we may call. Since this is an operation that does not behave too well with a method call logic based on runtime types we select only methods where the primitive types match, the parameter types are final or for methods that take no arguments. Currently methods with a variable parameter list are not matched in general, unless a fitting array is used for the method call.

Those two areas of optimization are only the beginning of further similar improvements. Upcoming versions of the Groovy 1.8.x branch will see more optimizations coming. In particular, primitive types other than integers should be expected to be supported shortly.

### GPars bundled within the Groovy distribution

The [GPars](#) project offers developers new intuitive and safe ways to handle Java or Groovy tasks concurrently, asynchronously, and distributed by utilizing the power of the Java platform and the flexibility of the Groovy language. Groovy 1.8 now bundles GPars 0.11 in the libraries of the Groovy installation, so that you can leverage all the features of the library for Fork/Join, Map/Filter/Reduce, DataFlow, Actors, Agents, and more with all the Groovy goodness.

To learn more about GPars, head over to the [GPars website](#), read the [detailed online user guide](#), or check out chapter 17 of [Groovy in Action, 2nd Edition \(MEAP\)](#).

## Closure enhancements

Closures are a central and essential piece of the Groovy programming language and are used in various ways throughout the Groovy APIs. In Groovy 1.8, we introduce the ability to use closures as annotation parameters. Closures are also a key part of what gives Groovy its functional flavor.

### *Closure annotation parameters*

In Java, there's a limited set of types you can use as annotation parameters (String, primitives, annotations, classes, and arrays of these). But in Groovy 1.8, we're going further and let you use closures as annotation parameters – which are actually transformed into a class parameter for compatibility reasons.

```
import java.lang.annotation.*

@Retention(RetentionPolicy.RUNTIME)
@interface Invariant {
    Class value() // will hold a closure
}

class
}

@Invariant({ number >= 0 })
class Distance {
    float number
    String unit
}

def d = new Distance(number: 10, unit:
"meters")

def anno = Distance.getAnnotation(Invariant)
def check = anno.value().newInstance(d, d)

assert check(d)
```

Closure annotation parameters open up some interesting possibilities for framework authors! As an example, the [G Contracts](#) project, which brings the "Design by Contract" paradigm to Groovy makes heavy use of annotation parameters to allow preconditions, postconditions and invariants to be declared.

### **Closure functional flavors**

#### **Closure composition**

If you recall your math lessons, function composition may be a concept you're familiar with. And in turn, **Closure composition** is about that: the ability to compose Closures together to form a new Closure which chains the call of those Closures. Here's an example of composition in action:

```
def plus2 = { it + 2 }
def times3 = { it * 3 }

def times3plus2 = plus2 << times3
assert times3plus2(3) == 11
assert times3plus2(4) == plus2(times3(4))

def plus2times3 = times3 << plus2
assert plus2times3(3) == 15
assert plus2times3(5) == times3(plus2(5))

// reverse composition
assert times3plus2(3) == (times3 >>
plus2)(3)
```

To see more examples of Closure composition and reverse composition, please have a look at our [test case](#).

#### **Closure trampoline**

When writing recursive algorithms, you may be getting the infamous stack overflow exceptions, as the stack starts to have a too high depth of recursive calls. An approach that helps in those situations is by using Closures and their new [trampoline](#) capability.

Closures are wrapped in a `TrampolineClosure`. Upon calling, a trampolined Closure will call the original Closure waiting for its result. If the outcome of the call is another instance of a `TrampolineClosure`, created perhaps as a result to a call to the `trampoline()` method, the Closure will again be invoked. This repetitive invocation of returned trampolined Closures instances will continue until a value other than a trampolined Closure is returned. That value will become the final result of the trampoline. That way, calls are made serially, rather than filling the stack.

Here's an example of the use of `trampoline()` to implement the factorial function:

```
def factorial
  factorial = { int n, def accu = 1G ->
    if (n < 2) return accu
    factorial.trampoline(n - 1, n * accu)
  }
  factorial = factorial.trampoline()

assert factorial(1)      == 1
assert factorial(3)      == 1 * 2 * 3
assert factorial(1000) == 402387260... //
plus another 2560 digits
```

#### Closure memoization

Another improvement to Closures is the ability to [memoize](#) the outcome of previous (ideally side-effect free) invocations of your Closures. The return values for a given set of Closure parameter values are kept in a cache, for those memoized Closures. That way, if you have an expensive computation to make that takes seconds, you can put the return value in cache, so that the next execution with the same parameter will return the same result – again, we assume results of an invocation are the same given the same set of parameter values.

There are three forms of memoize functions:

- the standard `memoize()` which caches all the invocations
- `memoizeAtMost(max)` call which caches a maximum number of invocations
- `memoizeAtLeast(min)` call which keeps at least a certain number of invocation results
- and `memoizeBetween(min, max)` which keeps a range results (between a minimum and a maximum)

Let's illustrate that:

```
def plus = { a, b -> sleep 1000; a + b
}.memoize()
assert plus(1, 2) == 3 // after 1000ms
assert plus(1, 2) == 3 // return immediately
assert plus(2, 2) == 4 // after 1000ms
assert plus(2, 2) == 4 // return immediately

// other forms:

// at least 10 invocations cached
def plusAtLeast = { ... }.memoizeAtLeast(10)

// at most 10 invocations cached
def plusAtMost = { ... }.memoizeAtMost(10)

// between 10 and 20 invocations cached
def plusAtLeast = { ... }.memoizeBetween(10,
20)
```

### *Currying improvements*

Currying improvements have also been backported to recent releases of Groovy 1.7, but it's worth outlining here for reference. Currying used to be done only from left to right, but it's also possible to do it from right to left, or from a given index, as the following examples demonstrate:

```
// right currying
def divide = { a, b -> a / b }
def halver = divide.rcurry(2)
assert halver(8) == 4

// currying n-th parameter
def joinWithSeparator = { one, sep, two ->
    one + sep + two
}
def joinWithComma =
    joinWithSeparator.ncurry(1, ', ')
assert joinWithComma('a', 'b') == 'a, b'
```

### Native JSON support

With the ubiquity of JSON as an interchange format for our applications, it is natural that Groovy added support for JSON, in a similar fashion as the support Groovy's always had with XML. So Groovy 1.8 introduces a JSON builder and parser.

#### *Reading JSON*

A `JsonSlurper` class allows you to parse JSON payloads, and access the nested Map and List data structures representing that content. JSON objects and arrays are indeed simply represented as Maps and Lists, giving you access to all the GPath expression benefits (subscript/property notation, find/findAll/each/inject/groupBy/etc.). Here's an example showing how to find all the recent commit messages on the Grails project:

```
import groovy.json.*

def payload = new
URL("http://github.com/api/v2/json/commits/1
ist/grails/grails-core/master").text

def slurper = new JsonSlurper()
def doc = slurper.parseText(payload)

doc.commits.message.each { println it }
```

If you want to see some more examples of the usage of the JSON parser, you can have a look at the [JsonSlurper tests](#) in our code base.

#### *JSON builder*

Parsing JSON data structures is one thing, but we should also be able to produce JSON content just like we create markup with the MarkupBuilder. The following example:

```
import groovy.json.*

def json = new JsonBuilder()

json.person {
    name "Guillaume"
    age 33
    pets "Hector", "Felix"
}

println json.toString()
```

Will create the JSON output:

```
{"person": {"name": "Guillaume", "age": 33, "pets": ["Hector", "Felix"]}}
```

You can find some more usages of the JSON builder in our [JsonBuilder tests](#).

#### *Pretty printing JSON content*

When given a JSON data structure, you may wish to pretty-print it, so that you can more easily inspect it, with a more friendly layout. So for instance, if you want to pretty print the result of the previous example, you could do:

```
import groovy.json.*  
  
println  
JsonOutput.prettyPrint('' '{ "person": { "name":  
"Guillaume", "age": 33, "pets": [ "Hector", "Felix"  
" ] } } '' )
```

Which would result in the following pretty-printed output:

```
{  
  "person": {  
    "name": "Guillaume",  
    "age": 33,  
    "pets": [  
      "Hector",  
      "Felix"  
    ]  
  }  
}
```

The Groovy compiler reads the source code, builds an Abstract Syntax Tree (AST) from it, and then puts the AST into bytecode. With AST transformations, the programmer can hook into this process. A general description of this process, an exhaustive description of all available transformations, and a guide of how to write your own ones can be found for example in [Groovy in Action, 2nd Edition \(MEAP\)](#), chapter 9.

Below is a list of all new transformations that come with Groovy 1.8. They save you from writing repetitive code and help avoiding common errors.

### **@Log**

You can annotate your classes with the @Log transformation to automatically inject a logger in your Groovy classes, under the `log` property. Four kind of loggers are actually available:

- @Log for java.util.logging
- @Commons for Commons-Logging
- @Log4j for Log4J
- @Slf4j for SLF4J

Here's a sample usage of the @Log transformation:

```
import groovy.util.logging.*  
  
@Log  
class Car {  
    Car() {  
        log.info 'Car constructed'  
    }  
}  
  
def c = new Car()
```

You can change the name of the logger by specifying a different name, for instance with `@Log('myLoggerName')`.

Another particularity of these logger AST transformations is that they take care of wrapping and safe-guarding logger calls with the usual `isSomeLevelEnabled()` calls. So when you write `log.info 'Car constructed'`, the generated code is actually equivalent to:

```
if (log.isLoggable(Level.INFO)) {  
    log.info 'Car constructed'  
}
```

### **@Field**

When defining variables in a script, those variables are actually local to the script's run method, so they are not accessible from other methods of the script. A usual approach to that problem has been to store variables in the binding, by not defining those variables and by just assigning them a value. Fortunately, the `@Field` transformation provides a better alternative: by annotating your variables in your script with this annotation, the annotated variable will become a private field of the script class.

More concretely, you'll be able to do as follows:

```
@Field List awe = [1, 2, 3]
def awesum() { awe.sum() }
assert awesum() == 6
```

### **@PackageScope enhancements**

The `@PackageScope` annotation can be placed on classes, methods or fields and is used for turning off Groovy's visibility conventions and reverting back to Java conventions. This ability is usually only needed when using 3rd party libraries which rely on the package scope visibility. When adding the `@PackageScope` annotation to a field, Groovy will assign package scope access to the field rather than automatically treating it as a property (and adding setters/getters). Annotating a class or method with `@PackageScope` will cause Groovy to revert to Java's convention of leaving the class/method as package scoped rather than automatically promoting it to public scope. The class variant can also take one or more parameters to allow nested setting of visibility of attributes within the class - see the Javadoc for more details. Recent releases of Groovy 1.7 have had a more limited version of this annotation.

### **@AutoClone**

The `@AutoClone` annotation is placed on classes which you want to be `Cloneable`. The annotation instructs the compiler to execute an AST transformation which adds a public `clone()` method and adds `Cloneable` to the classes implements list of interfaces. Because the JVM doesn't have a one-size-fits-all cloning strategy, several customizations exist for the cloning implementation:

- By default, the `clone()` method will call `super.clone()` before calling `clone()` on each `Cloneable` property of the class. Example usage:

```
import groovy.transform.AutoClone

@AutoClone
class Person {
    String first, last
    List favItems
    Date since
}
```

Which will create a class of the following form:

```

class Person implements Cloneable {
    ...
    public Object clone() throws CloneNotSupportedException {
        Object result = super.clone()
        result.favItems = favItems.clone()
        result.since = since.clone()
        return result
    }
    ...
}

```

- Another popular cloning strategy is known as the copy constructor pattern. If any of your fields are `final` and `Cloneable` you should set `style=COPY_CONSTRUCTOR` which will then use the copy constructor pattern.
- As a final alternative, if your class already implements the `Serializable` or `Externalizable` interface, you might like to set `style=SERIALIZATION` which will then use serialization to do the cloning.

See the Javadoc for `AutoClone` for further details.

#### **@AutoExternalizable**

The `@AutoExternalizable` class annotation is used to assist in the creation of `Externalizable` classes. The annotation instructs the compiler to execute an AST transformation which adds `writeExternal()` and `readExternal()` methods to a class and adds `Externalizable` to the interfaces which the class implements. The `writeExternal()` method writes each property (or field) for the class while the `readExternal()` method will read each one back in the same order. Properties or fields marked as `transient` are ignored. Example usage:

```

import groovy.transform.*

@AutoExternalize
class Person {
    String first, last
    List favItems
    Date since
}

```

Which will create a class of the following form:

```

class Person implements Externalizable {
    ...
    void writeExternal(ObjectOutput out)
throws IOException {
        out.writeObject(first)
        out.writeObject(last)
        out.writeObject(favItems)
        out.writeObject(since)
    }

    void readExternal(ObjectInput oin) {
        first = oin.readObject()
        last = oin.readObject()
        favItems = oin.readObject()
        since = oin.readObject()
    }
    ...
}

```

### *Controlling the execution of your code*

When integrating user-provided Groovy scripts and classes in your Java application, you may be worried about code that would eat all your CPU with infinite loops, or that call methods like `System.exit(0)` (for the latter, check the section on compiler customizers, and particularly the `SecureASTCustomizer`). It would be interesting to have a way to control the execution of that Groovy code, to be able to interrupt its execution when the thread is interrupted, when a certain duration has elapsed, or when a certain condition is met (lack of resources, etc).

Groovy 1.8 introduces three transformations for those purposes, as we shall see in the following sections. By default, the three transformations add some checks in at the beginning of each method body, and each closure body, to check whether a condition of interruption is met or not.

Note that those transformations are local (triggered by an annotation). If you want to apply them transparently, so that the annotation doesn't show up, I encourage you to have a look at the `ASTTransformationCustomizer` explained at the end of this article.

Cédric Champeau, our most recent Groovy committer, who implemented those features, has a [very nice blog post](#) covering those code interruption transformations.

### @ThreadInterrupt

You don't need to write checks in your scripts for whether the current thread of execution has been interrupted or not, by default, the transformation will add those checks for you for scripts and classes, at the beginning of each method body and closure body:

```
@ThreadInterrupt  
import groovy.transform.ThreadInterrupt  
  
while (true) {  
    // eat lots of CPU  
}
```

You can specify a `checkOnMethodStart` annotation parameter (defaults to true) to customize where checks are added by the transformation (adds an interrupt check by default as the first statement of a method body). And you can also specify the `applyToAllClasses` annotation parameter (default to true) if you want to specify whether only the current class or script should have this interruption logic applied or not.

### @TimedInterrupt

With `@TimedInterrupt`, you can interrupt the script after a certain amount of time:

```
@TimedInterrupt(10)  
import groovy.transform.TimedInterrupt  
  
while (true) {  
    // eat lots of CPU  
}
```

In addition to the previous annotation parameters we mentioned for `@ThreadInterrupt`, you should specify `value`, the amount of time to wait, and `unit` (defaulting to `TimeUnit.SECONDS`) to specify the unit of time to be used.

### @ConditionalInterrupt

An example of `@ConditionalInterrupt` which leverages the closure annotation parameter feature, and the `@Field` transformation as well:

```
@ConditionalInterrupt({ counter++ > 2 })  
import groovy.transform.ConditionalInterrupt  
import groovy.transform.Field  
  
@Field int counter = 0  
  
100.times {  
    println 'executing script method...'  
}
```

You can imagine defining any kind of condition: on counters, on resource availability, on resource usage, and more.

#### ***@ToString***

Provides your classes with a default `toString()` method which prints out the values of the class' properties (and optionally the property names and optionally fields). A basic example is here:

```
import groovy.transform.ToString  
  
@ToString  
class Person {  
    String name  
    int age  
}  
  
println new Person(name: 'Pete', age: 15)  
// => Person(Pete, 15)
```

And here's another example using a few more options:

```
@ToString(includeNames = true, includeFields  
= true)  
class Coord {  
    int x, y  
    private z = 0  
}  
println new Coord(x:20, y:5)  
// => Coord(x:20, y:5, z:0)
```

#### ***@EqualsAndHashCode***

Provides your classes with `equals()` and `hashCode()` methods based on the values of the class' properties (and optionally fields and optionally super class values for `equals()` and `hashCode()`).

```
import groovy.transform.EqualsAndHashCode  
  
@EqualsAndHashCode  
class Coord {  
    int x, y  
}  
  
def c1 = new Coord(x:20, y:5)  
def c2 = new Coord(x:20, y:5)  
  
assert c1 == c2  
assert c1.hashCode() == c2.hashCode()
```

#### ***@TupleConstructor***

Provides a tuple (ordered) constructor. For POGOs (plain old Groovy objects), this will be in addition to Groovy's default "named-arg" constructor.

```
import groovy.transform.TupleConstructor

@TupleConstructor
class Person {
    String name
    int age
}

def p1 = new Person(name: 'Pete', age: 15)
// map-based
def p2 = new Person('Pete', 15)
// tuple-based

assert p1.name == p2.name
assert p1.age == p2.age
```

#### *@Canonical*

Allows you to combine @ToString, @EqualsAndHashCode and @TupleConstructor. For those familiar with Groovy's @Immutable transform, this provides similar features but for mutable objects.

```
import groovy.transform.Canonical

@Canonical
class Person {
    String name
    int age
}

def p1 = new Person(name: 'Pete', age: 15)
def p2 = new Person('Paul', 15)

p2.name = 'Pete'
println "${p1.equals(p2)} $p1 $p2"
// => true Person(Pete, 15) Person(Pete, 15)
```

By default, `@Canonical` gives you vanilla versions for each of the combined annotations. If you want to use any of the special features that the individual annotations give you, simply include the individual annotation as well.

```

import groovy.transform.*

@Canonical
@ToString(includeNames = true)
class Person {
    String name
    int age
}

def p = new Person(name: 'Pete', age: 15)
println p
// => Person(name:Pete, age:15)

```

You will find a great [write-up on @Canonical, @ToString, @EqualsAndHashCode and @TupleConstructor](#) on John Prystash's weblog.

#### *@InheritConstructors*

Sometimes, when you want to subclass certain classes, you also need to override all the constructors of the parent, even if only to call the super constructor. Such a case happens for instance when you define your own exceptions, you want your exceptions to also have the constructors taking messages and throwable as parameters. But instead of writing this kind of boilerplate code each time for your exceptions:

```

class CustomException extends Exception {
    CustomException() { super() }
    CustomException(String msg) { super(msg) }
}

CustomException(String msg, Throwable t)
{ super(msg, t) }
CustomException(Throwable t) { super(t) }
}
}

```

Simply use the `@InheritConstructors` transformation which takes care of overriding the base constructors for you:

```
import groovy.transform.*  
  
@InheritConstructors  
class CustomException extends Exception {}
```

#### *@WithReadLock and @WithWriteLock*

Those two transformations, combined together, simplify the usage of `java.util.concurrent.locks.ReentrantReadWriteLock`, are safer to use than the `synchronized` keyword, and improve upon the `@Synchronized` transformation with a more granular locking.

More concretely, with an example, the following:

```
import groovy.transform.*  
  
class ResourceProvider {  
    private final Map<String, String> data =  
new HashMap<>()  
  
    @WithReadLock  
    String getResource(String key) {  
        return data.get(key)  
    }  
  
    @WithWriteLock  
    void refresh() {  
        //reload the resources into memory  
    }  
}
```

Will generate code as follows:

```
import
java.util.concurrent.locks.ReentrantReadWrit
eLock
import
java.util.concurrent.locks.ReadWriteLock

class ResourceProvider {
    private final ReadWriteLock
    $reentrantlock = new
    ReentrantReadWriteLock()
    private final Map<String, String> data =
    new HashMap<String, String>()

    String getResource(String key) {
        $reentrantlock.readLock().lock()
        try {
            return data.get(key)
        } finally {

    $reentrantlock.readLock().unlock()
        }
    }

    void refresh() throws Exception {
        $reentrantlock.writeLock().lock()
        try {
            //reload the resources into
memory
        } finally {

    $reentrantlock.writeLock().unlock()
        }
```

```

    }
}
}

```

### **@ListenerList**

If you annotate a Collection type field with @ListenerList, it generates everything that is needed to follow the bean event pattern. This is kind of an EventType independent version of what @Bindable is for PropertyChangeEvents.

This example shows the most basic usage of the @ListenerList annotation. The easiest way to use this annotation is to annotate a field of type List and give the List a generic type. In this example we use a List of type MyListener. MyListener is a one method interface that takes a MyEvent as a parameter. The following code is some sample source code showing the simplest scenario.

```

interface MyListener {
    void eventOccurred(MyEvent event)
}

class MyEvent {
    def source
    String message

    MyEvent(def source, String message) {
        this.source = source
        this.message = message
    }
}

class MyBeanClass {
    @ListenerList
    List<MyListener> listeners
}

```

- + addMyListener(MyListener) : void - This method is created based on the generic type of your annotated List field. The name and parameter type is are based on the List field's generic parameter.
- + removeMyListener(MyListener) : void- This method is created based on the generic type of your annotated List field. The name and parameter type is are based on the List field's generic parameter.

- + `getMyListeners() : MyListener[]` - This method is created based on the generic type of your annotated List field. The name is the plural form of the List field's generic parameter, and the return type is an array of the generic parameter.
- + `fireEventOccurred(MyEvent) : void` - This method is created based on the type that the List's generic type points to. In this case, MyListener is a one method interface with an `eventOccurred(MyEvent)` method. The method name is `fire[MethodName of the interface]` and the parameter is the parameter list from the interface. A `fireX` method will be generated for each public method in the target class, including overloaded methods.

## Alignments with JDK 7

Groovy 1.9 will be the version which will align as much as possible with the upcoming JDK 7, so beyond those aspects already covered in Groovy (like strings in switch and others), most of those "Project Coin" proposals will be in 1.9, except the "diamond operator" which was added in 1.8, as explained in the following paragraph.

### *Diamond operator*

Java 7 will introduce the "diamond" operator in generics type information, so that you can avoid the usual repetition of the parameterized types. Groovy decided to adopt the notation before JDK 7 is actually released. So instead of writing:

```
List<List<String>> list1 = new  
ArrayList<List<String>>()
```

You can "omit" the parameterized types and just use the pointy brackets, which now look like a diamond:

```
List<List<String>> list1 = new ArrayList<>()
```

## New DGM methods

- `count` Closure variants

```
def isEven = { it % 2 == 0 }  
assert [2,4,2,1,3,5,2,4,3].count(isEven) == 5
```

- `countBy`

```
assert [0:2, 1:3] == [1,2,3,4,5].countBy{ it % 2 }  
assert [(true):2, (false):4] == 'Groovy'.toList().countBy{ it == 'o' }
```

- plus variants specifying a starting index

```
assert [10, 20].plus(1, 'a', 'b') == [10, 'a', 'b', 20]
```

- equals for Sets and Maps now do flexible numeric comparisons (on values for Maps)

```
assert [1L, 2.0] as Set == [1, 2] as Set
assert [a:2, b:3] == [a:2L, b:3.0]
```

- toSet for primitive arrays, Strings and Collections

```
assert [1, 2, 2, 2, 3].toSet() == [1, 2, 3] as Set
assert 'groovy'.toSet() == ['v', 'g', 'r', 'o', 'y'] as Set
```

- min / max methods for maps taking closures  
(also available in Groovy 1.7)

```
def map = [a: 1, bbb: 4, cc: 5, dddd: 2]

assert map.max { it.key.size() }.key == 'dddd'
assert map.min { it.value }.value == 1
```

- map withDefault{}

Oftentimes, when using a map, for example for counting the frequency of words in a document, you need to check that a certain key exists, before doing something with the associating value (like incrementing it). Nothing really complex, but we could improve upon that with a new method, called `withDefault`. So instead of writing code like below:

```
def words = "one two two three three three".split()
def freq = [:]
words.each {
    if (it in freq)
        freq[it] += 1
    else
        freq[it] = 1
}
```

Thanks to the new method (also backported to 1.7), you can write the example as follows:

```
def words = "one two two three three three".split()
def freq = [:].withDefault { k -> 0 }
words.each {
    freq[it] += 1
}
```

## Miscellaneous

### *Slashy strings*

Slashy strings are now multi-line:

```
def poem = /
to be
or
not to be
/

assert poem.readlines().size() == 4
```

This is particularly useful for multi-line regexs when using the regex free-spacing comment style (though you would still need to escape slashes):

```
// match yyyy-mm-dd from this or previous
century
def dateRegex = /(?x)      # enable
whitespace and comments
((?:19|20)\d\d)          # year (group 1)
(non-capture alternation for century)
-                          # separator
(0[1-9]|1[012])          # month (group 2)
-                          # separator
(0[1-9]|12[0-9]|3[01])   # day (group 3)
/

assert '04/04/1988' ==
'1988-04-04'.find(dateRegex) { all, y, m, d
-> [d, m, y].join('/') }
```

### *Dollar slashy strings*

A new string notation has been introduced: the "dollar slashy" string. This is a multi-line GString similar to the slashy string, but with slightly different escaping rules. You are no longer required to escape slash (with a preceding

backslash) but you can use '\$\$' to escape a '\$' or '\$/' to escape a slash if needed. Here's an example of its usage:

```
def name = "Guillaume"
def date = "April, 21st"

def dollarSlashy = $/
  Hello $name,
  today we're ${date}
  $ dollar-sign
  $$ dollar-sign
  \ backslash
  / slash
  $/ slash
/$

println dollarSlashy
```

This form of string is typically used when you wish to embed content that may naturally contains slashes or backslashes and you don't want to have to rework the content to include all of the necessary escaping. Some examples are shown below:

- Embedded XML fragments with backslashes:

```
def tic = 'tic'

def xml = $/
  <xml>
  $tic\tac
  </xml>
/$

assert "\n<xml>\ntic\tac\n</xml>\n" == xml
```

Better than a normal (now multi-line) slashy string where you would have to escape the slashes or a triple quote (""") GString where you would have to escape the backslashes.

- Or windows pathnames containing a slash at the end:

```
def dir = $/C:\temp\/$
```

Previously, triple quote (""") GString required extra escaping, and the above sequence was illegal for a normal slashy string. Now, ugly workarounds are not needed.

- Embedding multi-line regexs when using the regex free-spacing comment style (particularly ones which contain slashes):

```
// match yyyy-mm-dd from current or previous century
def dateRegex = $/(?x)    # enable whitespace and comments
((?:19|20)\d\d)         # year (group 1) (non-capture alternation for
century)
[- /.]                  # seperator
(0[1-9]|1[012])         # month (group 2)
[- /.]                  # seperator
(0[1-9]|[12][0-9]|3[01]) # day (group 3)
/$

assert '04/04/1988' == '1988-04-04'.find(dateRegex) { all, y, m, d -> [d,
m, y].join('/') }
assert '10-08-1989' == '1989/08/10'.find(dateRegex) { all, y, m, d -> [d,
m, y].join('-') }
```

So, you can cut and paste most PERL regex examples without further escaping.

- Or Strings which are themselves Groovy code fragments containing slashes:

```
def alphabet = ('a'..'z').join(' ')
def code = $/
  def normal = '\b\t\n\r'
  def slashy = /\b\t\n\r/
  assert '$alphabet'.size() == 26
  assert normal.size() == 4
  assert slashy.size() == 8
/$
println code
Eval.me(code)
```

Again allowing you to cut and paste many slashy string Groovy examples and have them embedded within dollar slashy strings without further escaping.

### **Compilation customizers**

The compilation of Groovy code can be configured through the `CompilerConfiguration` class, for example for setting the encoding of your sources, the base script class, the recompilation parameters, etc). `CompilerConfiguration` now has a new option for setting *compilation customizers* (belonging to the `org.codehaus.groovy.control.customizers` package). Those customizers allow to customize the compilation process in three ways:

- adding default imports with the `ImportCustomizer`: so you don't have to always add the same imports all over again

- securing your scripts and classes with the `SecureASTCustomizer`: by allowing/disallowing certain classes, or special AST nodes (Abstract Syntax Tree), filtering imports, you can secure your scripts to avoid malicious code or code that would go beyond the limits of what the code should be allowed to do.
- applying AST transformations with the `ASTTransformationCustomizer`: lets you apply transformations to all the class nodes of your compilation unit.

For example, if you want to apply the `@Log` transformation to all the classes and scripts, you could do:

```
import
org.codehaus.groovy.control.CompilerConfigur
ation
import
org.codehaus.groovy.control.customizers.*
import groovy.util.logging.Log

def configuration = new
CompilerConfiguration()
configuration.addCompilationCustomizers(new
ASTTransformationCustomizer(Log))

def shell = new GroovyShell(configuration)
shell.evaluate("""
    class Car {
        Car() {
            log.info 'Car constructed'
        }
    }

    log.info 'Constructing a car'
    def c = new Car()
""")
```

This will log the two messages, the one from the script, and the one from the `Car` class constructor, through `java.util.logging`. No need to apply the `@Log` transformation manually to both the script and the class: the

transformation is applied to all class nodes transparently. This mechanism can also be used for adding global transformations, just for the classes and scripts that you compile, instead of those global transformations being applied to all scripts and classes globally.

If you want to add some default imports (single import, static import, star import, star static imports, and also aliased imports and static imports), you can use the import customizer as follows:

```
import
org.codehaus.groovy.control.CompilerConfigur
ation
import
org.codehaus.groovy.control.customizers.*

def configuration = new
CompilerConfiguration()
def custo = new ImportCustomizer()
custo.addStaticStar(Math.name)
configuration.addCompilationCustomizers(cust
o)

def shell = new GroovyShell(configuration)
shell.evaluate("""
    cos PI/3
""")
```

When you want to evaluate Math expressions, you don't need anymore to use the `import static java.lang.Math.*` star static import to import all the Math constants and static functions.

#### ***(G)String to Enum coercion***

Given a String or a GString, you can coerce it to Enum values bearing the same name, as the sample below presents:

```
enum Color {
    red, green, blue
}

// coercion with as
def r = "red" as Color

// implicit coercion
Color b = "blue"

// with GStrings too
def g = "${'green'}" as Color
```

#### *Maps support isCase()*

Maps now support `isCase()`, so you can use maps in your switch/case statements, for instance:

```
def m = [a: 1, b: 2]
def val = 'a'

switch (val) {
    case m: "key in map"; break
    // equivalent to // case { val in m }:
    ...
    default: "not in map"
}
```

#### *Grape/Grab Improvements*

Shorter notation for `@GrabResolver`

When you need to specify a special grab resolver, for when the artifacts you need are not stored in Maven central, you could use:

```
@GrabResolver(name = 'restlet.org', root =  
'http://maven.restlet.org')  
@Grab('org.restlet:org.restlet:2.0.6')  
import org.restlet.Restlet
```

Groovy 1.8 adds a shorter syntax as well:

```
@GrabResolver('http://maven.restlet.org')  
@Grab('org.restlet:org.restlet:2.0.6')  
import org.restlet.Restlet
```

Compact form for optional Grab attributes

The `@Grab` annotation has numerous options. For example, to download the Apache commons-io library (where you wanted to set the `transitive` and `force` attributes - not strictly needed for this example but see the `Grab` or `Ivy` documentation for details on what those attributes do) you could use a grab statement similar to below:

```
@Grab(group='commons-io',  
module='commons-io', version='2.0.1',  
transitive=false, force=true)
```

The compact form for grab which allows the artifact information to be represented as a string now supports specifying additional attributes. As an example, the following script will download the commons-io jar and the corresponding javadoc jar before using one of the commons-io methods.

```
@Grab('commons-io:commons-io:2.0.1;transitive=false;force=true')
@Grab('commons-io:commons-io:2.0.1;classifier=javadoc')
import static
org.apache.commons.io.FileSystemUtils.*
assert freeSpaceKb() > 0
```

### *Sql improvements*

The `eachRow` and `rows` methods in the `groovy.sql.Sql` class now support paging. Here's an example:

```
sql.eachRow('select * from PROJECT', 2, 2) {
row ->
    println "${row.name.padRight(10)}
($row.url)"
}
```

Which will start at the second row and return a maximum of 2 rows. Here's an example result from a database containing numerous projects with their URLs:

```
Grails      (http://grails.org)
Griffon     (http://griffon.codehaus.org)
```

### *Storing AST node metadata*

When developing AST transformations, and particularly when using a visitor to navigate the AST nodes, it is sometimes tricky to keep track of information as you visit the tree, or if a combination of transforms need to be sharing some context. The `ASTNode` base class features 4 methods to store node metadata:

- `public Object getNodeMetaData(Object key)`
- `public void copyNodeMetaData(ASTNode other)`
- `public void setNodeMetaData(Object key, Object value)`
- `public void removeNodeMetaData(Object key)`

### *Ability to customize the GroovyDoc templates*

GroovyDoc uses hard-coded templates to create the JavaDoc for your Groovy classes. Three templates are used: top-level templates, a package-level template, a class template. If you want to customize these templates, you can subclass the Groovydoc Ant task and override the `getDocTemplates()`, `getPackageTemplates()`, and `getClassTemplates()` methods pointing at your own templates. Then you can use your custom GroovyDoc Ant task in lieu of Groovy's original one.

## Groovy 1.8.1 release notes

### Table of Contents

- [This is a DRAFT PAGE for an unreleased version of Groovy and is incomplete and subject to change](#)
- [Groovy 1.8.1 release notes](#)
  - [Support for begin\(\) / end\(\) methods when processing files line by line with the groovy command](#)
  - [Sql batch support that allows PreparedStatements to be used as shown in these examples:](#)

 **This is a DRAFT PAGE for an unreleased version of Groovy and is incomplete and subject to change**

### Groovy 1.8.1 release notes

The 1.8.1 release of Groovy is primarily a bug fix release but also comes with the new features outlined below.

#### Support for begin() / end() methods when processing files line by line with the groovy command

A feature found in other scripting languages like Perl or Awk is to be able to have a begin / end method when processing a file line by line. The groovy command supports this mode of operation, but didn't support the begin / end methods. (this feature is actually going to be available in 1.7.11 and 1.8.1, but time ran short for inclusion in the 1.8 final release)

More concretely, if you have a text file named `dummy.txt`, and you want to count the number of lines it contains, you could do this on the command-line:

```
groovy -a -ne 'def begin() { nb = 0 }; def
end() { System.err.println nb }; nb++'
dummy.txt
```

Sql batch support that allows PreparedStatements to be used as shown in these examples:

Shown for a batch size of 20:

```

def updateCounts = sql.withBatch(20, 'insert
into TABLENAME(a, b, c) values (?, ?, ?)') {
ps ->
    ps.addBatch(10, 12, 5)        // varargs
style
    ps.addBatch([7, 3, 98])      // list
    ps.addBatch([22, 67, 11])
    ...
}

```

Named parameters (into maps or domain objects) are also supported:

```

def updateCounts = sql.withBatch(20, 'insert
into TABLENAME(a, b, c) values (?foo, :bar,
:baz)') { ps ->
    ps.addBatch([foo:10, bar:12, baz:5]) //
map
    ps.addBatch(foo:7, bar:3, baz:98)    //
Groovy named args allow outer brackets to be
dropped
    ...
}

```

Named ordinal parameters (into maps or domain objects) are also supported:

```

def updateCounts = sql.withBatch(20, 'insert
into TABLENAME(a, b, c) values (?1.foo,
?2.bar, ?2.baz)') { ps ->
    ps.addBatch([[foo:22], [bar:67,
baz:11]]) // list of maps or domain objects
    ps.addBatch([foo:10], [bar:12, baz:5])
// varargs allows outer brackets to be
dropped
    ps.addBatch([foo:7], [bar:3, baz:98])
    ...
}
def updateCounts2 = sql.withBatch(5, 'insert
into TABLENAME(a, b, c) values (?1, ?2.bar,
?2.baz)') { ps ->
    ps.addBatch(10, [bar:12, baz:5])
    ps.addBatch(7, [bar:3, baz:98])
    ...
}

```

## Groovy 2.0 release notes

## Groovy 2.0 release notes

**i** The content of this page was originally featured in the [What's new in Groovy 2.0 article from InfoQ](#).

The newly released Groovy 2.0 brings key static features to the language with static type checking and static compilation, adopts JDK 7 related improvements with Project Coin syntax enhancements and the support of the new “invoke dynamic” JVM instruction, and becomes more modular than before. In this article, we’re going to look into those new features in more detail.

- [Groovy 2.0 release notes](#)
- [A “static theme” for a dynamic language](#)
  - [Static type checking](#)
    - [Spotting obvious typos](#)

- [Check your assignments and return values](#)
- [More on type inference](#)
- [Common type conversions still allowed](#)
- [Mixing dynamic features and statically typed methods](#)
- [Type inference and instanceof checks](#)
- [Lowest Upper Bound](#)
- [Flow typing](#)
- [Static compilation](#)
- [The Java 7 and JDK 7 theme](#)
  - [Project Coin syntax enhancements](#)
    - [Binary literals](#)
    - [Underscore in number literals](#)
    - [Multicatch block](#)
  - [Invoke Dynamic support](#)
    - [Enabling invoke dynamic support](#)
    - [Promising performance improvements](#)
- [A more modular Groovy](#)
  - [Groovy modules](#)
  - [Extension modules](#)
    - [Contributing an instance method](#)
    - [Contributing a static method](#)
    - [Extension module descriptor](#)
    - [Grabbing an extension](#)
- [Summary](#)

## A “static theme” for a dynamic language

### Static type checking

Groovy, by nature, is and will always be a dynamic language. However, Groovy is often used as a "Java scripting language", or as a "better Java" (ie. a Java with less boilerplate and more power features). A lot of Java developers actually use and embed Groovy in their Java applications as an extension language, to author more expressive business rules, to further customize the application for different customers, etc. For such Java-oriented use cases, developers don't need all the dynamic capabilities offered by the language, and they usually expect the same kind of feedback from the Groovy compiler as the one given by javac. In particular, they want to get compilation errors (rather than runtime errors) for things like typos on variable or method names, incorrect type assignments and the like. That's why Groovy 2 features static type checking support.

#### *Spotting obvious typos*

The static type checker is built using Groovy's existing powerful AST (Abstract Syntax Tree) transformation mechanisms but for those not familiar with these mechanisms you can think of it as an optional compiler plugin triggered through an annotation. Being an optional feature, you are not forced to use it if you don't need it. To trigger static type checking, just use the `@TypeChecked` annotation on a method or on a class to turn on checking at your desired level of granularity. Let's see that in action with a first example:

```
import groovy.transform.TypeChecked

void someMethod() {}

@TypeChecked
void test() {
    // compilation error:
    // cannot find matching method
    someMethod()
    someMethod()

    def name = "Marion"
    // compilation error:
    // the variable naaammme is undeclared
    println naaammme
}
```

We annotated the test() method with the @TypeChecked annotation, which instructs the Groovy compiler to run the static type checking for that particular method at compilation time. We're trying to call someMethod() with some obvious typos, and to print the name variable again with another typo, and the compiler will throw two compilation errors because respectively, the method and variable are not found or declared.

*Check your assignments and return values*

The static type checker also verifies that the return types and values of your assignments are coherent:

```

import groovy.transform.TypeChecked

@TypeChecked
Date test() {
    // compilation error:
    // cannot assign value of Date
    // to variable of type int
    int object = new Date()

    String[] letters = ['a', 'b', 'c']

    // compilation error:
    // cannot assign value of type String
    // to variable of type Date
    Date aDateVariable = letters[0]

    // compilation error:
    // cannot return value of type String
    // on method returning type Date
    return "today"
}

```

In this example, the compiler will complain about the fact you cannot assign a Date in an int variable, nor can you return a String instead of a Date value specified in the method signature. The compilation error from the middle of the script is also interesting, as not only does it complain of the wrong assignment, but also because it shows type inference at play, because the type checker, of course, knows that letters[0] is of type String, because we're dealing with an array of Strings.

#### *More on type inference*

Since we're mentioning type inference, let's have a look at some other occurrences of it. We mentioned the type checker tracks the return types and values:

```
import groovy.transform.TypeChecked

@TypeChecked
int method() {
    if (true) {
        // compilation error:
        // cannot return value of type String
        // on method returning type int
        'String'
    } else {
        42
    }
}
```

Given a method returning a value of primitive type `int`, the type checker is able to also check the values returned from different constructs like `if / else` branches, `try / catch` blocks or `switch / case` blocks. Here, in our example, one branch of the `if / else` blocks tries to return a `String` value instead of a primitive `int`, and the compiler complains about it.

#### *Common type conversions still allowed*

The static type checker, however, won't complain for certain automatic type conversions that Groovy supports. For instance, for method signatures returning `String`, `boolean` or `Class`, Groovy converts return values to these types automatically:

```

import groovy.transform.TypeChecked

@TypeChecked
boolean booleanMethod() {
    "non empty strings are evaluated to true"
}

assert booleanMethod() == true

@TypeChecked
String stringMethod() {
    // StringBuilder converted to String
    calling toString()
    new StringBuilder() << "non empty string"
}

assert stringMethod() instanceof String

@TypeChecked
Class classMethod() {
    // the java.util.List class will be
    returned
    "java.util.List"
}

assert classMethod() == List

```

The static type checker is also clever enough to do type inference:

```

import groovy.transform.TypeChecked

@TypeChecked
void method() {
    def name = " Guillaume "

    // String type inferred (even inside
    GString)
    println "NAME = ${name.toUpperCase()}"

    // Groovy GDK method support
    // (GDK operator overloading too)
    println name.trim()

    int[] numbers = [1, 2, 3]
    // Element n is an int
    for (int n in numbers) {
        println n
    }
}

```

Although the name variable was defined with `def`, the type checker understands it is of type `String`. Then, when this variable is used in the interpolated string, it knows it can call `String`'s `toUpperCase()` method, or the `trim()` method later one, which is a method added by the Groovy Development Kit decorating the `String` class. Last, when iterating over the elements of an array of primitive ints, it also understands that an element of that array is obviously an `int`.

#### *Mixing dynamic features and statically typed methods*

An important aspect to have in mind is that using the static type checking facility restricts what you are allowed to use in Groovy. Most runtime dynamic features are not allowed, as they can't be statically type checked at compilation time. So adding a new method at runtime through the type's metaclasses is not allowed. But when you need to use some particular dynamic feature, like Groovy's builders, you can opt out of static type checking should you wish to.

The `@TypeChecked` annotation can be put at the class level or at the method level. So if you want to have a whole class type checked, put the annotation on the class, and if you want only a few methods type checked, put the

annotation on just those methods. Also, if you want to have everything type checked, except a specific method, you can annotate the latter with `@TypeChecked(TypeCheckingMode.SKIP)` — or `@TypeChecked(SKIP)` for short, if you statically import the associated enum. Let's illustrate the situation with the following script, where the `greeting()` method is type checked, whereas the `generateMarkup()` method is not:

```
import groovy.transform.TypeChecked
import groovy.xml.MarkupBuilder

// this method and its code are type checked
@TypeChecked
String greeting(String name) {
    generateMarkup(name.toUpperCase())
}

// this method isn't type checked
// and you can use dynamic features like the
markup builder
String generateMarkup(String name) {
    def sw = new StringWriter()
    new MarkupBuilder(sw).html {
        body {
            div name
        }
    }
    sw.toString()
}

assert greeting("Cédric").contains("CÉDRIC")
```

#### *Type inference and instanceof checks*

Current production releases of Java don't support general type inference; hence we find today many places where code is often quite verbose and cluttered with boilerplate constructs. This obscures the intent of the code and without the support of powerful IDEs is also harder to write. This is the case with `instanceof` checks: You often check

the class of a value with instanceof inside an if condition, and afterwards in the if block, you must still use casts to be able to use methods of the value at hand. In plain Groovy, as well as in the new static type checking mode, you can completely get rid of those casts.

```
import groovy.transform.TypeChecked
import groovy.xml.MarkupBuilder

@TypeChecked
String test(Object val) {
    if (val instanceof String) {
        // unlike Java:
        // return ((String)val).toUpperCase()
        val.toUpperCase()
    } else if (val instanceof Number) {
        // unlike Java:
        // return
((Number)val).intValue().multiply(2)
        val.intValue() * 2
    }
}

assert test('abc') == 'ABC'
assert test(123) == '246'
```

In the above example, the static type checker knows that the val parameter is of type String inside the if block, and of type Number in the else if block, without requiring any cast.

#### *Lowest Upper Bound*

The static type checker goes a bit further in terms of type inference in the sense that it has a more granular understanding of the type of your objects. Consider the following code:

```
import groovy.transform.TypeChecked

// inferred return type:
// a list of numbers which are comparable
and serializable
@TypeChecked test() {
    // an integer and a BigDecimal
    return [1234, 3.14]
}
```

In this example, we return, intuitively, a list of numbers: an Integer and a BigDecimal. But the static type checker computes what we call a “lowest upper bound”, which is actually a list of numbers which are also serializable and comparable. It’s not possible to denote that type with the standard Java type notation, but if we had some kind of intersection operator like an ampersand, it could look like `List<Number & Serializable & Comparable>`.

#### *Flow typing*

Although this is not really recommended as a good practice, sometimes developers use the same untyped variable to store values of different types. Look at this method body:

```

import groovy.transform.TypeChecked

@TypeChecked test() {
    def var = 123           // inferred
    type is int
    var = "123"           // assign var
    with a String

        println var.toInteger() // no problem,
    no need to cast

        var = 123
        println var.toUpperCase() // error, var
    is int!
}

```

The var variable is initialized with an int. Then, a String is assigned. The “flow typing” algorithm follows the flow of assignment and understands that the variable now holds a String, so the static type checker will be happy with the toInteger() method added by Groovy on top of String. Next, a number is put back in the var variable, but then, when calling toUpperCase(), the type checker will throw a compilation error, as there’s no toUpperCase() method on Integer.

There are some special cases for the flow typing algorithm when a variable is shared with a closure which are interesting. What happens when a local variable is referenced in a closure inside a method where that variable is defined? Let’s have a look at this example:

```
import groovy.transform.TypeChecked

@TypeChecked test() {
    def var = "abc"
    def cl = {
        if (new Random().nextBoolean()) var =
new Date()
    }
    cl()
    var.toUpperCase() // compilation error!
}
```

The var local variable is assigned a String, but then, var might be assigned a Date if some random value is true. Typically, it's only at runtime that we really know if the condition in the if statement of the closure is made or not. Hence, at compile-time, there's no chance the compiler can know if var now contains a String or a Date. That's why the compiler will actually complain about the toUpperCase() call, as it is not able to infer that the variable contains a String or not. This example is certainly a bit contrived, but there are some more interesting cases:

```

import groovy.transform.TypeChecked

class A                { void foo() {} }
class B extends A { void bar() {} }

@TypeChecked test() {
    def var = new A()
    def cl = { var = new B() }
    cl()
    // var is at least an instance of A
    // so we are allowed to call method foo()
    var.foo()
}

```

In the test() method above, var is assigned an instance of A, and then an instance of B in the closure which is call afterwards, so we can at least infer that var is of type A.

All those checks added to the Groovy compiler are done at compile-time, but the generated bytecode is still the same dynamic code as usual — no changes in behavior at all.

Since the compiler now knows a lot more about your program in terms of types, it opens up some interesting possibilities: what about compiling that type checked code statically? The obvious advantage will be that the generated bytecode will more closely resemble the bytecode created by the javac compiler itself, making statically compiled Groovy code as fast as plain Java, among other advantages. In the next section, we'll learn more about Groovy's static compilation.

### Static compilation

As we shall see in the following chapter about the JDK 7 alignments, Groovy 2.0 supports the new “invoke dynamic” instruction of the JVM and its related APIs, facilitating the development of dynamic languages on the Java platform and bringing some additional performance to Groovy's dynamic calls. However, unfortunately shall I say, JDK 7 is not widely deployed in production at the time of this writing, so not everybody has the chance to run on the latest version. So developers looking for performance improvements would not see much changes in Groovy 2.0, if they aren't able to run on JDK 7. Luckily, the Groovy development team thought those developers could get interesting performance boost, among other advantages, by allowing type checked code to be compiled statically.

Without further ado, let's dive in and use the new @CompileStatic transform:

```
import groovy.transform.CompileStatic

@CompileStatic
int squarePlusOne(int num) {
    num * num + 1
}

assert squarePlusOne(3) == 10
```

This time, instead of using `@TypeChecked`, use `@CompileStatic`, and your code will be statically compiled, and the bytecode generated here will look like `javac`'s bytecode, running just as fast. Like the `@TypeChecked` annotation, `@CompileStatic` can annotate classes and methods, and `@CompileStatic(SKIP)` can bypass static compilation for a specific method, when its class is marked with `@CompileStatic`.

Another advantage of the `javac`-like bytecode generation is that the size of the bytecode for those annotated methods will be smaller than the usual bytecode generated by Groovy for dynamic methods, since to support Groovy's dynamic features, the bytecode in the dynamic case contains additional instructions to call into Groovy's runtime system.

Last but not least, static compilation can be used by framework or library code writers to help avoid adverse interactions when dynamic metaprogramming is in use in several parts of the codebase. The dynamic features available in languages like Groovy are what give developers incredible power and flexibility but if care is not taken, different assumptions can exist in different parts of the system with regards to what metaprogramming features are in play and this can have unintended consequences. As a slightly contrived example, consider what happens if you are using two different libraries, both of which add a similarly named but differently implemented method to one of your core classes. What behaviour is expected? Experienced users of dynamic languages will have seen this problem before and probably heard it referred to as "monkey patching". Being able to statically compile parts of your code base — those parts that don't need dynamic features — shields you from the effects of monkey patching, as the statically compiled code doesn't go through Groovy's dynamic runtime system. Although dynamic runtime aspects of the language are not allowed in a static compilation context, all the usual AST transformation mechanisms work just as well as before, since most AST transforms perform their magic at compilation time.

In terms of performance, Groovy's statically compiled code is usually more or less as fast as `javac`'s. In the few micro-benchmarks the development team used, performance is identical in several cases, and sometimes it's slightly slower.

Historically, thanks to the transparent and seamless integration of Java and Groovy, we used to advise developers to optimize some hotspot routines in Java for further performance gains, but now, with this static compilation option, this is no longer the case, and people wishing to develop their projects in full Groovy can do so.

## **The Java 7 and JDK 7 theme**

The grammar of the Groovy programming language actually derives from the Java grammar itself, but obviously, Groovy provides additional nice shortcuts to make developers more productive. This familiarity of syntax for Java

developers has always been a key selling point for the project and its wide adoption, thanks to a flat learning curve. And of course, we expect Groovy users and newcomers to also want to benefit from the few syntax refinements offered by Java 7 with its “Project Coin” additions.

Beyond the syntax aspects, JDK 7 also brings interesting novelties to its APIs, and for a first time in a long time, even a new bytecode instruction called “invoke dynamic”, which is geared towards helping implementors develop their dynamic languages more easily and benefit from more performance.

### **Project Coin syntax enhancements**

Since day 1 (that was back in 2003 already!) Groovy has had several syntax enhancements and features on top of Java. One can think of closures, for example, but also the ability to put more than just discrete values in switch / case statements, where Java 7 only allows Strings in addition. So some of the Project Coin syntax enhancements, like Strings in switch, were already present in Groovy. However, some of the enhancements are new, such as binary literals, underscore in number literals, or the multi catch block, and Groovy 2 supports them. The sole omission from the Project Coin enhancements is the “try with resources” construct, for which Groovy already provides various alternatives through the rich API of the Groovy Development Kit.

#### *Binary literals*

In Java 6 and before, as well as in Groovy, numbers could be represented in decimal, octal and hexadecimal bases, and with Java 7 and Groovy 2, you can use a binary notation with the “0b” prefix:

```
int x = 0b10101111  
assert x == 175  
  
byte aByte = 0b00100001  
assert aByte == 33  
  
int anInt = 0b1010000101000101  
assert anInt == 41285
```

#### *Underscore in number literals*

When writing long literal numbers, it’s harder on the eye to figure out how some numbers are grouped together, for example with groups of thousands, of words, etc. By allowing you to place underscore in number literals, it’s easier to spot those groups:

```

long creditCardNumber = 1234_5678_9012_3456L
long socialSecurityNumbers = 999_99_9999L
double monetaryAmount = 12_345_132.12
long hexBytes = 0xFF_EC_DE_5E
long hexWords = 0xFFEC_DE5E
long maxLong = 0x7fff_ffff_ffff_ffffL
long alsoMaxLong =
9_223_372_036_854_775_807L
long bytes =
0b11010010_01101001_10010100_10010010

```

#### *Multicatch block*

When catching exceptions, we often replicate the catch block for two or more exceptions as we want to handle them in the same way. A workaround is either to factor out the commonalities in its own method, or in a more ugly fashion to have a catch-all approach by catching Exception, or worse, Throwable. With the multi catch block, we're able to define several exceptions to be catch and treated by the same catch block:

```

try {
    /* ... */
} catch(IOException | NullPointerException
e) {
    /* one block to handle 2 exceptions */
}

```

#### **Invoke Dynamic support**

As we mentioned earlier in this article, JDK 7 came with a new bytecode instruction called “invoke dynamic”, as well as with its associated APIs. Their goal is to help dynamic language implementors in their job of crafting their languages on top of the Java platform, by simplifying the wiring of dynamic method calls, by defining “call sites” where dynamic method call section can be cached, “method handles” as method pointers, “class values” to store any kind of metadata along class objects, and a few other things. One caveat though, despite promising performance improvements, “invoke dynamic” hasn't yet fully been optimized inside the JVM, and doesn't yet always deliver the best performance possible, but update after update, the optimizations are coming.

Groovy brought its own implementation techniques, to speed up method selection and invocation with “call site

caching”, to store metaclasses (the dynamic runtime equivalent of classes) with its metaclass registry, to perform native primitive calculations as fast as Java, and much more. But with the advent of “invoke dynamic”, we can rebase the implementation of Groovy on top of these APIs and this JVM bytecode instruction, to gain performance improvements and to simplify our code base.

If you’re lucky to run on JDK 7, you’ll be able to use a new version of the Groovy JARs which has been compiled with the “invoke dynamic” support. Those JARs are easily recognizable as they use the “-indy” classifier in their names.

#### *Enabling invoke dynamic support*

Using the “indy” JARs is not enough, however, to compile your Groovy code so that it leverages the “invoke dynamic” support. For that, you’ll have to use the --indy flag when using the “groovyc” compiler or the “groovy” command. This also means that even if you’re using the indy JARs, you can still target JDK 5 or 6 for compilation.

Similarly, if you’re using the groovyc Ant task for compiling your projects, you can also specify the indy attribute:

```
...
<taskdef name="groovyc"

  classname="org.codehaus.groovy.ant.Groovyc"
  classpathref="cp" />
...
<groovyc srcdir="{srcDir}"
  destdir="{destDir}" indy="true">
  <classpath>
  ...
  </classpath>
</groovyc>
...
```

The Groovy Eclipse Maven compiler plugin hasn’t yet been updated with the support of Groovy 2.0 but this will be the case shortly. For GMaven plugin users, although it’s possible to configure the plugin to use Groovy 2.0 already, there’s currently no flag to enable the invoke dynamic support. Again, GMaven will also be updated soon in that regard.

When integrating Groovy in your Java applications, with GroovyShell, for example, you can also enable the invoke dynamic support by passing a CompilerConfiguration instance to the GroovyShell constructor on which you access and set the optimization options:

```
CompilerConfiguration config = new  
CompilerConfiguration();  
config.getOptimizationOptions().put("indy",  
true);  
config.getOptimizationOptions().put("int",  
false);  
GroovyShell shell = new  
GroovyShell(config);
```

As invokedynamic is supposed to be a full replacement to dynamic method dispatch, it is also necessary to disable the primitive optimizations which generate extra bytecode that is here to optimize edge cases. Even if it is in some cases slower than with primitive optimizations activated, future versions of the JVM will feature an improved JIT which will be capable of inlining most of the calls and remove unnecessary boxings.

#### *Promising performance improvements*

In our testing, we noticed some interesting performance gains in some areas, whereas other programs could run slower than when not using the invoke dynamic support. The Groovy team has further performance improvements in the pipeline for Groovy 2.1 however, but we noticed the JVM isn't yet finely tuned and still has a long way to go to be fully optimized. But fortunately, upcoming JDK 7 updates (in particular update 8) should already contain such improvements, so the situation can only improve. Furthermore, as invoke dynamic is used for the implementation of JDK 8 Lambdas, we can be sure more improvements are forthcoming.

### **A more modular Groovy**

We'll finish our journey through the new features of Groovy 2.0 by speaking about modularity. Just like Java, Groovy is not just a language, but it's also a set of APIs serving various purposes: templating, Swing UI building, Ant scripting, JMX integration, SQL access, servlet serving, and more. The Groovy deliverables were bundling all these features and APIs inside a single big JAR. However, not everybody needs everything at all times in their own applications: you might be interested in the template engine and the servlets if you're writing some web application, but you might only need the Swing builder when working on a rich desktop client program.

### **Groovy modules**

So the first goal of the modularity aspect of this release is to actually split the original Groovy JAR into smaller modules, smaller JARs. The core Groovy JAR is now twice as small, and we have the following feature modules available:

- Ant: for scripting Ant tasks for automating administration tasks
- BSF: for integrating Groovy in your Java applications with the old Apache Bean Scripting Framework
- Console: module containing the Groovy Swing console
- GroovyDoc: for documenting your Groovy and Java classes
- Groovysh: module corresponding to the Groovysh command-line shell
- JMX: for exposing and consuming JMX beans

- JSON: for producing and consuming JSON payloads
- JSR-223: for integrating Groovy in your Java applications with the JDK 6+ javax.scripting APIs
- Servlet: for writing and serving Groovy script servlets and templates
- SQL: for querying relational databases
- Swing: for building Swing UIs
- Templates: for using the template engine
- Test: for some test support, like the GroovyTestCase, mocking, and more
- TestNG: for writing TestNG tests in Groovy
- XML: for producing and consuming XML documents

With Groovy 2, you're now able to just pick up the modules you're interested in, rather than bringing everything on your classpath. However, we still provide the "all" JAR which contains everything, if you don't want to complicate your dependencies for just a few megabytes of saved space. We also provide those JARs compiled with the "invoke dynamic" support as well, for those running on JDK 7.

## Extension modules

The work on making Groovy more modular also yielded an interesting new feature: extension modules. By splitting Groovy into smaller modules, a mechanism for modules to contribute extension methods has been created. That way, extension modules can provide instance and static methods to other classes, including the ones from the JDK or third-party libraries. Groovy uses this mechanism to decorate classes from the JDK, to add new useful methods to classes like String, File, streams, and many more — for example, a getText() method on URL allows you to retrieve the content of a remote URL through an HTTP get. Notice also that those extension methods in your modules are also understood by the static type checker and compiler. But let's now have a look at how you can add new methods to existing types.

### *Contributing an instance method*

To add new methods to an existing type, you'll have to create a helper class that will contain those methods. Inside that helper class, all the extension methods will actually be public (the default for Groovy but required if implementing in Java) and static (although they will be available on instances of that class). They will always take a first parameter which is actually the instance on which this method will be called. And then following parameters will be the parameters passed when calling the method. This is the same convention use for Groovy categories.

Say we want to add a greets() method on String, that would greet the name of the person passed in parameters, so that you could that method as follow:

```
assert "Guillaume".greets("Paul") == "Hi  
Paul, I'm Guillaume"
```

To accomplish that, you will create a helper class with an extension method like so:

```
package com.acme

class MyExtension {
    static String greets(String self, String
name) {
        "Hi ${name}, I'm ${self}"
    }
}
```

#### *Contributing a static method*

Static extension methods are defined using the same mechanism, but have to be declared in a separate class. The extension module descriptor then determines whether the class provides instance or static methods. Let's add a new static method to Random to get a random integer between two values, you could proceed as in this class:

```
package com.acme

class MyStaticExtension {
    static String between(Random selfType,
int start, int end) {
        new Random().nextInt(end - start + 1)
+ start
    }
}
```

That way, you are able to use that extension method as follows:

```
Random.between(3, 4)
```

#### *Extension module descriptor*

Once you've coded your helper classes (in Groovy or even in Java) containing the extension methods, you need to create a descriptor for your module. You must create a file called `org.codehaus.groovy.runtime.ExtensionModule` in the `META-INF/services` directory of your module archive. Four essential fields can be defined, to tell the Groovy runtime about the name and version of your module, as well as to point at your helper classes for extension methods with a comma-separated list of class names. Here is what our final module descriptor looks like:

```
moduleName = MyExtension  
moduleVersion = 1.0  
extensionClasses = com.acme.MyExtension  
staticExtensionClasses =  
com.acme.MyStaticExtension
```

With this extension module descriptor on the classpath, you are now able to use those extension methods in your code, without needing an import or anything else, as those extension methods are automatically registered.

#### *Grabbing an extension*

With the `@Grab` annotation in your scripts, you can fetch dependencies from Maven repositories like Maven Central. With the addition of the `@GrabResolver` annotation, you can specify your own location for your dependencies as well. If you are "grabbing" an extension module dependency through this mechanism, the extension method will also be installed automatically. Ideally, for consistency, your module name and version should be coherent with the artifact id and version of your artifact.

## Summary

Groovy is very popular among Java developers and offers them a mature platform and ecosystem for their application needs. But without resting still, the Groovy development team continues to further improve the language and its APIs to help its users increase their productivity on the Java platform.

Groovy 2.0 responds to three key themes:

- More performance: with the support of JDK 7 Invoke Dynamic to speed up Groovy for those lucky to have JDK 7 already in production, but also with static compilation for JDK 5 and beyond for everyone, and particularly those ready to abandon some aspects of dynamicity to shield themselves from the reach of "monkey patching" and to gain the same speed as Java.
- More Java friendliness: with the support of the Java 7 Project Coin enhancements to keep Groovy and Java as close syntax cousins as ever, and with the static type checker to have the same level of feedback and type safety as provided by the javac compiler for developers using Groovy as a Java scripting language
- More modularity: with a new level of modularity, Groovy opens the doors for smaller deliverables, for example for integration in mobile applications on Android, and allowing the Groovy APIs to grow and evolve with newer versions and newer extension modules, as well as allowing users to contribute extension methods to existing types.

## Groovy 2.1 release notes

With this new 2.1 release, Groovy:

- has **full support for the JDK 7 "invoke dynamic"** instruction and API,
- goes beyond conventional static type checking capabilities with a **special annotation for closure delegate**

**based Domain-Specific Languages** and **static type checker extensions**,

- provides **additional compilation customization** options,
- and features a **meta-annotation facility** for combining annotations elegantly.

## Table of Content

- [Full invoke dynamic support](#)
- [GPars 1.0](#)
- [@DelegatesTo annotation](#)
- [Type checker extensions](#)
- [Compile-time Meta-annotations](#)
  - [Alternate notation](#)
  - [Passing parameters](#)
  - [Custom processor](#)
- [Compilation customization](#)
  - [Custom base script class flag](#)
  - [Compiler configuration script](#)
  - [Source-aware customizer](#)
  - [Compiler customization builder](#)
- [Other Minor Enhancements](#)
  - [Additional DGM methods](#)
  - [GroovyDoc](#)
  - [Command-line](#)
  - [XML Processing](#)
  - [ConfigSlurper](#)
  - [@Delegate](#)
  - [@ToString](#)
  - [@EqualsAndHashCode](#)
  - [@Immutable](#)
  - [@AutoClone](#)

## Full invoke dynamic support

With Groovy 2.0, we introduced support for JDK 7's "invoke dynamic" bytecode instruction and API to benefit from the dedicated support and performance improvements for dynamic languages starting with JDK 7. Groovy 2.1 brings **full support for "invoke dynamic"** (aka "indy"), completing the work introduced in 2.0.

In Groovy 2.0, most method calls were using the "invoke dynamic" instruction, but there have been exceptions: constructor calls or "spread calls" (where you pass arguments with the "spread operator"). Groovy 2.1 completes the implementation started in 2.0. Now, code compiled with the "invoke dynamic" JAR on JDK 7 will not be using the old "call site caching" code which served us well for getting good performance for Groovy prior to JDK 7. If you are lucky enough to be using JDK 7 in production, be sure to use the Groovy 2.1 "indy" JAR to benefit from the full "invoke dynamic" support. The "indy" version is bundled with the binary download package and can be obtained via Maven (all JARs with "invoke dynamic" support are postfixed with "-indy").

You can [learn more about the invoke dynamic support](#).

## GPars 1.0

Groovy 2.1's distribution **bundles the recently released GPars 1.0**, the one-stop shop for all your concurrency needs. This new version comes with various enhancements in the asynchronous functions, [promises](#), parallel collections, actors, dataflow support, [Google App Engine support](#), etc.

Be sure to check the [release announcement](#) and read the “[what’s new section](#)” of the GParc user guide. You can also have a look at the detailed [JIRA release notes](#).

## @DelegatesTo annotation

Authoring Domain-Specific Languages (DSLs) has always been a sweet spot for Groovy, and the availability of closures and the malleable syntax of the language has allowed DSL implementors to build nice mini-languages like “builders”, to represent configuration or hierarchical data.

Thanks to the various delegation strategies of the `groovy.lang.Closure` class, a range of very powerful techniques can be used when building DSLs. Due to different implementation techniques, inferring type information within the DSL has not been straightforward. This is especially an issue when DSLs should have proper IDE support (e.g. code completion).

The very popular and powerful [Gradle](#) build automation system uses its own DSL for build script specifications. On the DSL implementation layer are various methods taking closures as arguments, and with special delegation strategies delegating to some other parameter passed to them. Providing good IDE support for Groovy DSLs — like the one in Gradle — has presented a few challenges. Hence the need for the `@DelegatesTo` annotation.

Groovy 2.1 introduces the `@groovy.lang.DelegatesTo` annotation as a **documentation mechanism** for DSL users and maintainers, as an **IDE hint for providing better coding assistance**, and as additional information that can be **taken into account by the static type checker and static compilation** introduced in Groovy 2.0. Let’s see that in action with some examples.

A closure delegate based method usage might look like the following:

```
exec {
    launch()
}
```

The `exec()` method takes a closure as parameter, and the actual `launch()` call inside that closure is delegated to some particular object (the closure delegate), instead of being dispatched to the enclosing class. The above code would only fail at runtime (not at compile-time!), as the `launch()` method can not be found in the closure context. In order to delegate method calls within the closure’s code block to another object instance, we need to set the closure delegate.

Setting a closure delegate is as easy as invoking `Closure#setDelegate(Object)`:

```
void exec(Closure c) {
    c.delegate = new Executor()
    c()
}
```

The delegate can be set to an arbitrary object instance (here, an instance of an `Executor` class that has a `launch`

( ) method). When the delegate is set accordingly, we can execute the closure code.

Note that usually, to avoid odd behavior if the closure is used in multiple threads, we tend to clone that closure.

The problem with delegate objects are IDEs not knowing about them. Given our example, most IDEs will underline the `launch()` method as being an unknown method in this context.

This is where `@DelegatesTo` comes into play. By adding the `@DelegatesTo` annotation to DSL methods like `exec(Closure)`, IDEs get the actual delegate type and other meta-data.

A future update might let GroovyDoc show the details about the annotation usage to help users know what methods they can call, what properties they can access, etc.

Here's what your `exec()` method will look like with the annotation:

```
void exec(@DelegatesTo(Executor) Closure c)
{
    c.delegate = new Executor()
    c()
}
```

Besides specifying the actual delegate type, `@DelegatesTo` can be used to hint at the actual resolve strategy. The resolve strategy determines the order in which non-closure method / property calls are looked up. In our example, `Closure.DELEGATE_FIRST` will be used. This indicates the closure will attempt to resolve against the given delegate object in first place, followed by the owner object:

```
import static groovy.lang.Closure.*
// ...
void exec(@DelegatesTo(strategy =
DELEGATE_FIRST, value = Executor) Closure c)
{
    c.delegate = new Executor()
    c.resolveStrategy = DELEGATE_FIRST
    c()
}
```

IDE support is not the only reason to use `@DelegatesTo`. The **static type checker and static compiler take the additional meta-data specified by the `@DelegatesTo` annotation into account**. If there is a typo in the closure code block, the type checker will complain. And if you use the static compilation capability introduced in Groovy 2.0,

the calls will be compiled statically.

Let's say we wouldn't call `launch()` but `launchr()` in the closure code block, we would get a message like:

```
[Static type checking] - Cannot find  
matching method  
DelegatesToSamples#launchr().  
Please check if the declared type is right  
and if the method exists.
```

**Static type checks for custom Domain-Specific Languages** is a very convenient feature in Groovy 2.1!

In addition, Groovy 2.1 features other abilities for even further type checking your DSLs, as you shall see in the following section.

Before moving on, let's mention a few closing details about `@DelegatesTo`.

`@DelegatesTo` allows to specify the receiver calls are delegated to. For instance, when a delegate calls a method or property on another method parameter. Imagine our `exec()` method taking the `Executor` argument instance as delegate:

```
void exec(Executor ex,  
@DelegatesTo(Executor) Closure c) {  
    c.delegate = ex  
    c()  
}
```

In this example, the information is lost that the call is delegated to the `ex` parameter. Thanks to the `@DelegatesTo.Target` annotation we can specify `ex` as target for being the delegate object:

```
void exec(@DelegatesTo.Target Executor ex,  
@DelegatesTo Closure c) {  
    c.delegate = ex  
    c()  
}
```

What if we had several `Executor` parameters, how would we differentiate which one we're targeting?

```
void exec(  
    @DelegatesTo.Target('param1') Executor  
    ex,  
    @DelegatesTo(target = 'param1') Closure  
    c) { ... }
```

The delegation “target” can be specified with an arbitrary id. In the example above it is `param1`.

One last very nice little feature: if you are using static type checking, you can omit the type of the parameter and `@DelegatesTo` combined with “flow typing” (the ability of following the current type of an untyped variable) would still know if method calls are valid:

```
void exec(@DelegatesTo.Target ex,  
@DelegatesTo Closure c) {  
    c.delegate = ex  
    c()  
}  
  
class Executor {  
    void launch() {}  
}  
  
def ex = new Executor()  
  
exec(ex) {  
    launch()  
}
```

We've seen that the `@DelegatesTo` helps documenting, tooling, and checking Domain-Specific Languages in the specific context of closure delegate based methods, but we hinted at the fact we can go beyond, in terms of static type checking for your DSLs.

For more details take a look at the [@DelegatesTo documentation](#).

## Type checker extensions

Static type checking was introduced in Groovy 2.0, but Groovy 2.1 goes beyond built-in type checks and offers a way to **create type checker extensions**. This is great news for Groovy scripts, configuration files, or Domain-Specific Languages implementations as they can be “type checked” with more advanced, domain-specific rules. As an example, it would be possible to create a custom DSL type checker that throws compilation errors when certain verbs of the DSL are not recognized, or tells this other noun is allowed even if it’s a dynamic name bound at runtime, or type checks literal strings containing SQL code to see if the syntax is correct, and more.

Imagine a script, where we define a small robot class and instantiate it:

```
class Robot {  
    void move(String dist) { println "Moved  
$dist" }  
}  
  
robot = new Robot()
```

And we want to operate our robot in the `operate()` method, but we want this method to be type checked:

```
@groovy.transform.TypeChecked  
void operate() {  
    robot.move "left"  
}  
  
operate()
```

The static type checker will complain as it doesn’t understand where the `robot` variable is coming from, as it’s going through the binding of the script — note that we could teach the type checker to figure out binding-bound variables. It will throw an error telling us that the `robot` variable was undeclared.

But by utilizing type checker extensions, we can hook into the type checking process to teach it how to handle unresolved variables! In order to do that, we’ll specify an extension script through the newly introduced `extension` s annotation parameter of the `@TypeChecked` annotation:

```

@TypeChecked(extensions =
  'RobotMove.groovy')
void operate() {
    robot.move "left"
}

```

Now it's time to define the type checker extension script called `RobotMove.groovy`. The type checker extension script is written by applying a new DSL — the “type checking DSL”. The DSL provides various hooks for type checker extensions to register to. Going back to the example above, we register for unresolved variables using the `unresolvedVariable` hook:

```

unresolvedVariable { VariableExpression var
->
    if ('robot' == var.name) {
        def robotClass =
context.source.AST.classes.find { it.name ==
'Robot' }
        storeType(var, robotClass)
        handled = true
    }
}

```

The type checker extension script needs to be on the classpath. If this is the case, the script gets notified during compile-time when the static type checker encounters an unresolved variable. The unresolved variable closure is handed over a `VariableExpression`.

The `VariableExpression` is an object directly from Groovy's AST (Abstract Syntax Tree). It is a representation of the unresolved variable expression. The script checks if the variable is named `robot`, if this is the case, we lookup a `ClassNode` representing the `Robot` class, and store the type of that variable back in the AST. At the end, the `handled` property is set to `true`, to indicate the type checker already managed that variable. As a consequence, you won't get the compilation error about that undeclared variable.

To continue the journey, let's consider the case where the user enters a wrong direction string. We could of course use an enum or some other class containing direction constants, but for the sake of the example, we'll have a look at how we can teach the type checker to inspect strings and how you can actually throw your own compilation errors.

For that purpose, let's say a robot can only move left, right, forward and backward. And now, let's change our robot move instruction to:

```
robot.move "sideways"
```

The robot is not allowed to move sideways, so we should instruct the type checker to throw a compilation error if it encounters a direction the robot will not be able to understand. Here's how we can achieve our goal, by adding a new event handler to our `RobotMove.groovy` script:

```
afterMethodCall { MethodCall mc ->  
    def method = getTargetMethod(mc)  
    if (mc.objectExpression.name == 'robot'  
&& method.name == 'move') {  
        def args = getArguments(mc)  
        if (args &&  
isConstantExpression(args[0]) &&  
args[0].value instanceof String) {  
            def content = args[0].text  
            if (!(content in ['left',  
'right', 'backward', 'forward'])) {  
  
                addStaticTypeError("'${content}' is not a  
valid direction", args[0])  
            }  
        }  
    }  
}
```

This handler receives a `MethodCall` expression. We are using the `getTargetMethod()` utility method to retrieve the corresponding `MethodNode`. We check that the method call is a call to our `robot`, and that the name of the method corresponds to the `move` method. Then, we fetch the arguments passed to that method call, and if we're passed a direction in the form of a string constant, we are checking that the direction is an actual allowed direction. If this is not the case, we are adding a new static typing compilation error into the mix, so that the compiler will yell at the poor user because he used a direction which is forbidden and not understood by our robot.

This second example is also interesting in a way that it shows how you can even add compilation checks on things like literal strings on a domain-specific level, paving the way for possible checks on sprintf strings, on SQL or HQL code in strings, etc, allowing you to go even further than what the Java compiler actually checks.

The extension script can make use of various event oriented extension points and utility methods coming from the `TypeCheckingExtension` class from Groovy, such as:

- `unresolvedVariable`
- `unresolvedProperty`
- `unresolvedAttribute`
- `methodNotFound`
- `incompatibleAssignment`
- `beforeVisitMethod`
- `afterVisitMethod`
- `beforeVisitClass`
- `afterVisitClass`
- `beforeMethodCall`
- `afterMethodCall`
- `onMethodSelection`
- `setup`
- `finish`

The two examples are just the tip of the iceberg, but we will work out more complete documentation of the various extension points and utility methods going forward.

For more details take a look at the [type checking extensions documentation](#).

## Compile-time Meta-annotations

Annotations are a great way to add supplementary meta-data to classes, methods, fields, and other source code elements, thus frameworks, libraries, and even Groovy's homegrown AST transformations can take advantage of them to do some special treatments to the corresponding AST nodes. Every now and then the use case arises to reuse a combination of annotations, potentially at the expense of a galore of at-signs that obscure the general intent of that particular combination.

To group annotations together, to make the intent clearer or to streamline your code, Groovy 2.1 offers a **meta-annotation system, which allows to combine other annotations into one "alias" annotation**.

Imagine we are using some annotations defining constraints on properties of your class, like `@NotNull`, `@Length`, or `@Pattern`, which could be defined as follows:

```

@interface NotNull {}

@interface Length {
    int value() default 0
}

@interface Pattern {
    String value() default ".*"
}

```

An example of how to annotate an ISBN property with those annotations could look like this:

```

class Book {
    @NotNull
    @Length(10)
    @Pattern(/\d{9}(\d|x)/)
    String isbn10
}

```

For a single property, that's quite a bit of annotation overload! And it could be the case of other domain classes with properties having the same validation rules as the ISBN property, where we would need to duplicate that pattern.

As of Groovy 2.1, `@groovy.transform.AnnotationCollector` can be used to solve code duplication for this use case. `@AnnotationCollector` can be specified on annotation types and acts as meta-annotation. Whenever an annotation marked with it is found, it is replaced with its own annotations. Let's illustrate this with our ISBN example.

We will create a new annotation combination for the 13-digit ISBN standard, but this time, using the `@AnnotationCollector` meta-annotation:

```
@NotNull
@Length(13)
@Pattern(/\d{12}(\d|x)/)
@groovy.transform.AnnotationCollector
@interface ISBN13 {}
```

@ISBN13 as a single annotation can now be applied on code elements, instead of applying the entire annotation gang::

```
class Book {
    // ...
    @ISBN13
    String isbn13
}
```

What is particularly interesting with such **meta-annotations** is that they are actually **replaced at compilation time with the *real* annotations**. So if you counted the number of annotations on the `isbn13` property, you would count 3 (`@NotNull`, `@Length` and `@Pattern`). Thus, your underlying framework doesn't need to know about that meta-annotation solution and act accordingly.

#### *Alternate notation*

In our example above, we annotated our meta-annotation with the annotations that are then combined together. But for annotations for which you don't need to specify arguments, you could have also passed the names of the annotations to combine as parameters to the annotation collector:

```
import groovy.transform.*
@groovy.transform.AnnotationCollector([ToString, Singleton])
@interface ChattySingleton {}
```

In the above case, we combine the `@Singleton` and `@ToString` transformation into a meta-annotation called `@ChattySingleton`.

#### *Passing parameters*

If you need to pass some specific parameter to one of the underlying annotations which are combined, you can still do so by passing the parameter to the meta-annotation.

Let's assume we need to combine the following annotations:

```
@interface Service {}  
  
@interface Transactional {  
    String propagation() default "required"  
}
```

We define the meta-annotation combining both the above annotations:

```
@groovy.transform.AnnotationCollector([Service,  
Transactional])  
@interface TransactionalService {}
```

But we want to change the propagation strategy for the underlying `@Transactional` annotation, we do so by passing the parameter to the meta-annotation:

```
@TransactionalService(propagation =  
"mandatory")  
class BankingService { }
```

Note that if two combined annotations share the same parameter name, the last annotation declared wins and gets the parameter passed to the meta-annotation.

### *Custom processor*

If you need even more flexibility, meta-annotations allow you to define custom processors. The role of the custom processor is to go beyond the simple exchange of the meta-annotation with the combined annotations, to further customize the logic of that transformation.

Custom processors must be precompiled to take action, so we'll create our processor, and then evaluate our final example with `GroovyShell`, but first, let's talk about the use case.

We have two validation annotations for defining a minimum and maximum value for an integer property:

```
@interface Min {  
    int value() default 0  
}  
  
@interface Max {  
    int value() default 100  
}
```

If we want to define a range of values, with a lower and an upper bound, we could define a new annotation and implement the associated validation logic, or we could use custom meta-annotation processors to replace a range annotation with a minimum and a maximum one.

So instead of writing:

```
class Room {  
    @Min(1)  
    @Max(4)  
    int numberOfPersons  
}
```

We could write:

```
class Room {  
    @Range(from = 1, to = 4)  
    int numberOfPersons  
}
```

With the normal replacement logic, there's no way we can map the lower and upper bound values to the minimum and maximum annotation element default values. That is where custom processors come into play.

Our meta-annotation definition will look like this:

```

@Min @Max
@AnnotationCollector(processor =
    'RangeAnnotationProcessor')
@interface Range {}

```

Notice how we specify that the `@Range` annotation is a combination of `@Min` and `@Max`, and more importantly, how we pass a processor parameter to the `@AnnotationCollector` to instruct it about our custom meta-annotation processing logic.

In order to create a custom processor, you have to extend the `AnnotationCollectorTransform` class and override the `visit()` method:

```

import
org.codehaus.groovy.transform.AnnotationColl
ectorTransform
import org.codehaus.groovy.ast.*
import
org.codehaus.groovy.control.SourceUnit

class RangeAnnotationProcessor extends
AnnotationCollectorTransform {
    List<AnnotationNode>
visit(AnnotationNode collector,

AnnotationNode usage,
                                     AnnotatedNode
annotated,
                                     SourceUnit
src) {

        def minExpr =
usage.getMember('from')

```

```
def maxExpr = usage.getMember('to')

def (minAnno, maxAnno) =
getTargetAnnotationList(collector, usage,
src)

minAnno.addMember('value', minExpr)
maxAnno.addMember('value', maxExpr)

usage.members.remove('from')
usage.members.remove('to')
```

```
        return [minAnno, maxAnno]
    }
}
```

A few words about the parameters : the `collector` corresponds to the `@Range` annotation definition, `usage` to the actual usage of the `@Range` annotation, `annotated` is the annotated class, and `src` is script being compiled.

We start our implementation of the processor by retrieving the numeric expressions of the bounds defined as the `from` and `to` annotation parameters, because we'll pass those values back to the underlying `@Min` and `@Max` combined annotations. In order to do that, we retrieve the `@Min` and `@Max` combined annotations thanks to the `getTargetAnnotationsList()` method. We then set the values of the `@Min` and `@Max` annotations to the expressions we've retrieved before. We remove the `from` and `to` bounds from the `@Range` meta-annotation since those parameters aren't really defined on a real annotation but on a meta-annotation. And last, we return the two `@Min` and `@Max` annotations. If you wanted the Groovy compiler to do its usual replacement logic, you could have also called `super.visit(...)`, but in our case it wasn't needed.

The full example can be found in this Gist on Github: <https://gist.github.com/4563430>

Additional details can be found in the [meta-annotations documentation](#).

## Compilation customization

### *Custom base script class flag*

When integrating and evaluating Groovy scripts in an application for business rules or Domain-Specific Languages, it is often valuable to define a base script class, in order to add various utility methods, properties, or interception mechanisms for missing methods or properties.

The `CompilerConfiguration` object, that you can pass to `GroovyShell` and other integration mechanisms, allows you to specify a base script class with the `setScriptBaseClass()` method.

As of Groovy 2.1, we introduce the ability to define a base script class reference for your scripts via an additional command-line option `-b / --basescript` for the `groovyc` command, as well as for the `groovy` command.

Here's an example using a script called `businessRule.groovy`:

```
assert lookupRate(EUR, USD) == 1.33
```

In the above script, we notice two things: the usage of a `lookupRate()` method, and two undeclared variables: `EUR` and `USD`. Neither the method, nor the variables have been defined in our script. Instead, they are provided by a base script class, which can look like the following `ExchangeRateBaseScript.groovy` class:

```

abstract class ExchangeRateBaseScript
extends Script {
    def lookupRate(String currency1, String
currency2) {
        if (currency1 == 'EUR' && currency2
== 'USD')
            return 1.33
        else return 1
    }

    def getProperty(String name) { name }
}

```

The `lookupRate()` method used in our script is declared in the the base class, and the two currencies are retrieved via the `getProperty()` method.

Now it's time to wire them together, by instructing the `groovyc` compiler or the `groovy` command line launcher to use our base script class for all `groovy.lang.Script` descendants:

```

groovy --basescript
ExchangeRateBaseScript.groovy
businessRule.groovy

```

### *Compiler configuration script*

Similarly to the `--basescript` flag, there's another new option for the `groovy` and `groovyc` commands: the `--configscript` flag. Its purpose is to let you further configure the compiler, in a configuration script, by parameterizing the `CompilerConfiguration` object used for the compilation.

With a `CompilerConfiguration`, you can customize the various aspects of the Groovy compilation process. For example, you can specify various compilation customizers introduced in Groovy 1.8. Imagine you want to add a new default import to your classes, like importing all `java.lang.Math` functions and constants, so that your scripts and classes don't have to prefix those functions and constants all the time, and to avoid having to do that import wherever needed. Here's how you can proceed.

At first, your script, `mathFormula.groovy`, contains the following lines:

```
import static java.lang.Math.*  
  
assert sin(PI/2) == 1
```

For evaluating such math expressions, you wish to make the static import implicit, so that the final script will actually look like this:

```
assert sin(PI/2) == 1
```

If you'd run it as is, you'd get an error message saying:

```
No such property: PI for class: mathFormula
```

We need to use `CompilerConfiguration` to do add an `ImportCustomizer`. We'll create an `importConfiguration.groovy` script with the content below:

```
import  
org.codehaus.groovy.control.customizers.ImportCustomizer  
  
def importCustomizer = new  
ImportCustomizer()  
importCustomizer.addStaticStar('java.lang.Math')  
  
configuration.addCompilationCustomizers(importCustomizer)
```

We import and then instantiate an `ImportCustomizer`, on which we ask for a static star import of the methods and constants of the `java.lang.Math` class. Eventually, we pass that customizer to the `configuration` variable, which is an instance of `CompilerConfiguration` that will be used for the compilation of your math formula.

Now, we are able to execute your formula with the following command-line:

```
groovy --configscript  
importConfigurer.groovy mathFormula.groovy
```

### *Source-aware customizer*

If you use the groovy compiler to compile all your classes, one drawback of the approach above is that the customization applies globally to all classes that are going to be compiled. You may want to add certain default imports only in certain classes (ie. scripts containing math), but you might want to do something different for other classes, like adding a `@ToString` transformation to all the domain classes of your application. For that purpose, a new customizer was created, the `SourceAwareCustomizer`, to let you filter which classes should be impacted by particular compilation customizations, such as filtering by class name, by file extension, or by a custom logic.

Coming back to our previous example, let's add the default import to our `mathFormula.groovy` script, but add a `@ToString` transformation to the `MyDomain.groovy` class:

```

import
org.codehaus.groovy.control.customizers.*
import groovy.transform.ToString

def importCustomizer = new
ImportCustomizer()
importCustomizer.addStaticStar('java.lang.Math')

configuration.addCompilationCustomizers(
    new SourceAwareCustomizer(new
ASTTransformationCustomizer(ToString)) {
        boolean acceptBaseName(baseName) {
baseName ==~ 'MyDomain' }
    },
    new
SourceAwareCustomizer(importCustomizer) {
        boolean acceptBaseName(baseName) {
baseName ==~ 'mathFormula' }
    })

```

### *Compiler customization builder*

The more complex the customization becomes, the more cumbersome the above configuration becomes to write too, that's why Groovy 2.1 also provides a builder for building these types of configurations.

The builder allows you to use a familiar declarative syntax and saves you from adding manually various imports. Let's adapt our example above with the builder:

```

withConfig(configuration) {
    source(basenameValidator: {
it.endsWith('MyDomain') }) {
        ast(ToString)
    }
    source(basenameValidator: {
it.endsWith('mathFormula') }) {
        imports {
            staticStar 'java.lang.Math'
        }
    }
}
}

```

The configuration code is easier to read and maintain, thanks to the clarity brought by the builder approach. But we've only seen a couple examples of customization, and you should have a look at the other customizations available in the `org.codehaus.groovy.control.customizers.builder` package to learn more about them.

More details can be found in the [advanced compiler configuration documentation](#).

## Other Minor Enhancements

### *Additional DGM methods*

There are now `leftShift` and `withFormatter` methods for `Appendable` objects.

There are now methods for creating temporary directories and determining the total size of all files in a directory.

There is now a `collectMany` for maps (has been backported to earlier versions of Groovy too).

There is now a `closeStreams()` method for `Process` objects.

### *GroovyDoc*

You can now explicitly set a file encoding.

### *Command-line*

There is support for using a "jar:" prefix when running a script from a URL, in addition to the "file:" and "http:".

### *XML Processing*

There is a method for escaping / encoding XML entities in Strings.

There is a convenience method for serializing `Elements` objects.

You can now clone `Node` and `NodeList` objects.

The `name()` method now works for all `Node` objects, not just `Element` objects.

### ***ConfigSlurper***

Multiple environments blocks are now supported and merged.

### ***@Delegate***

Can now carry over annotations if desired for methods and method parameters.

### ***@ToString***

You can now cache the `toString` value. This is useful for immutable objects.

### ***@EqualsAndHashCode***

You can now cache the calculated `hashCode` value. This is useful for immutable objects.

### ***@Immutable***

You can now specify `knownImmutables`. This is useful when you know you are using an immutable object, but its type isn't one of the known immutable types.

### ***@AutoClone***

There is now a `SIMPLE AutoCloneStyle` which avoids some annoyances with Java's cloning behavior from `Object`. Those who need to clone Grails domain objects might find this useful.

## **News and Further Information**

- [Books](#)
- [Groovy 1.0 parties](#)
- [Groovy Series](#)
- [latest posts from our mailing-lists](#)
- [News](#)
  - [Release candidate of Groovy 2.0 available](#)
- [PLEAC Examples](#)
- [GroovyBlogs](#): A JavaBlog-like news aggregator for the Groovy and Grails mailing-lists feeds, and many feeds from famous bloggers spreading the Groovy and Grails love.

## **Books**

### **Groovy in Action, Second Edition**

The [Manning book page](#) leads to all the online resources about the book like

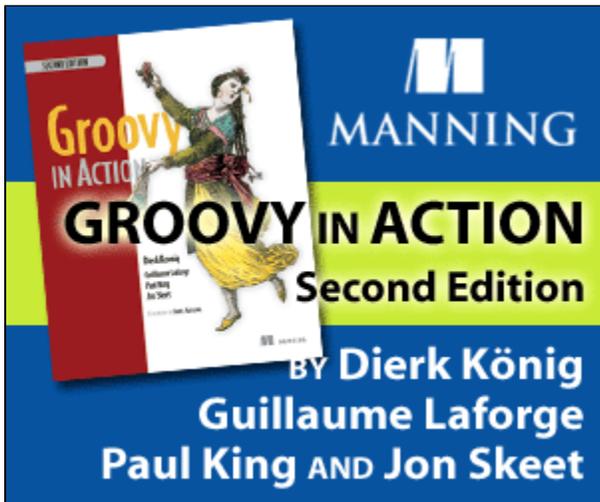
- the table of contents
- free chapters
- reader's forum
- errata

The [Amazon page](#)

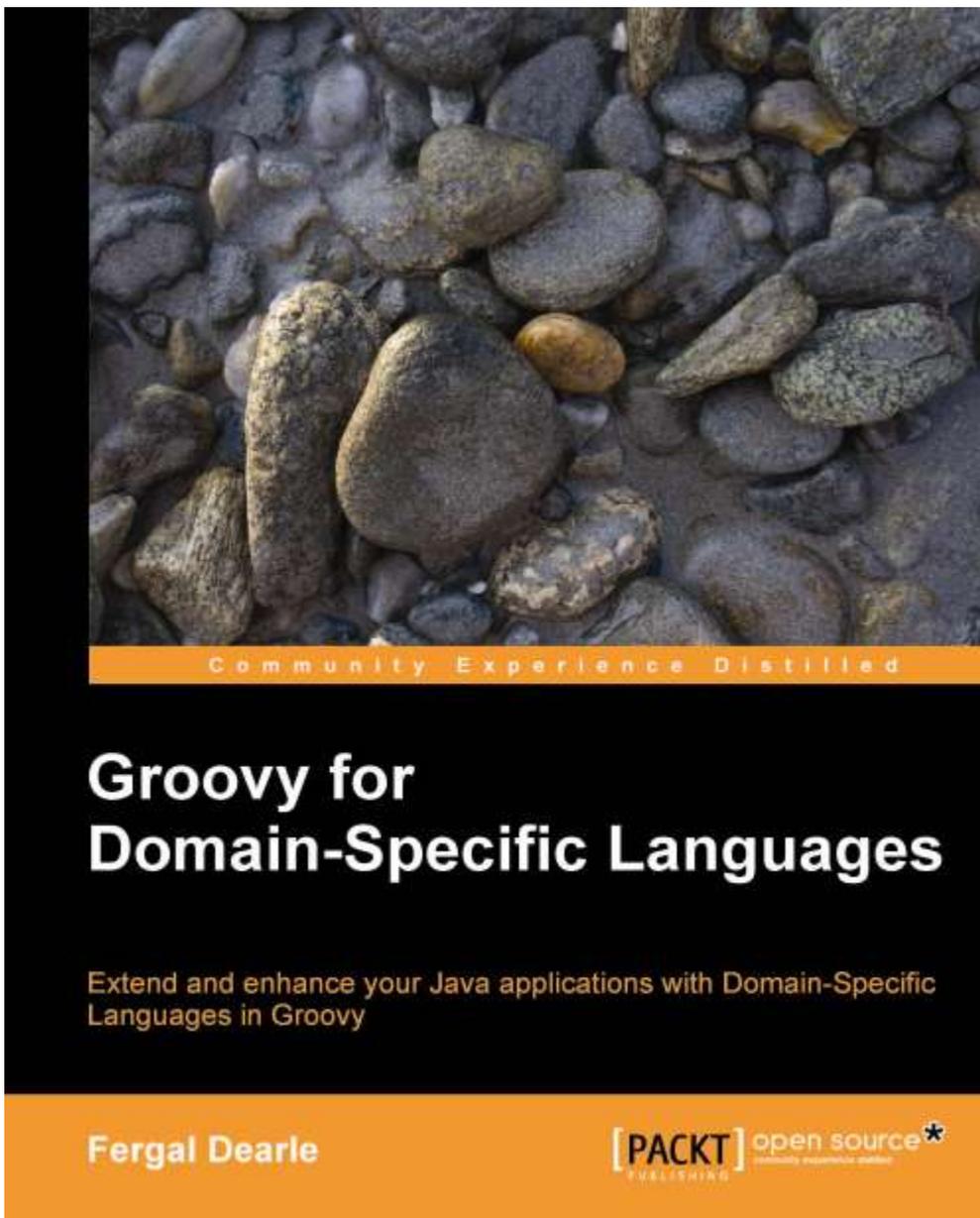
allows for quickly buying the book.

There also is a German edition of the book called "Groovy im Einsatz".  
You can find it at [Amazon DE](#) and on the [Hanser publishing site](#) where you can also get the PDF/ebook version.

The Japanese edition is available at [Amazon JP](#) and on the [publisher site](#).



## Groovy for Domain-Specific Languages



[Fergal Dearle's book](#) helps users to extend and enhance your Java applications with Domain Specific Languages in Groovy.

What you will learn from this book :

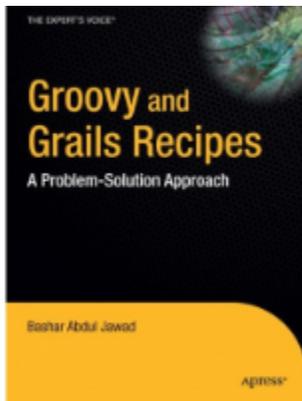
- Learn Groovy scripting designed to get Java programmers using Groovy quickly
- Use the meta-programming features in Groovy to build your own DSLs
- Learn everything you need to know about Groovy markup and build your own Groovy builders to simplify your application development
- Build effective DSLs using operator overloading and Groovy categories
- Work with Groovy closures
- Get to grips with the working of Groovy on the JVM and integrate Groovy with your Java applications

## Programming Groovy

- [Authors' book page](#)
- [Amazon](#)



## Groovy and Grails Recipes



- [Author's Book Page](#)
- [Publisher's Page](#)
- [Amazon Page](#)

*Groovy and Grails Recipes* is the busy developers' guide for developing applications in Groovy and Grails. Rather than boring you with theoretical knowledge of "yet another language/framework," this book delves straight into solving real-life problems in Groovy and Grails using easy-to-understand, well-explained code snippets. Through learning by example, you will be able to pick up on Groovy and Grails quickly and use the book as an essential reference when developing applications.

### **What you'll learn**

- Discover elegant and efficient solutions to common programming problems and web development tasks.
- Get and reuse practical examples for both Groovy language and Grails framework, using the latest stable versions of each.
- Perform a wide range of development tasks that cover all of the web development tiers, from View Layer to Service Layer to Domain Layer.
- Access the wide range of available Grails framework plug-ins.
- Obtain the recipes to integrate Spring, Hibernate, SiteMesh, and more with the Grails web framework.

### **Who is this book for?**

This book is for Java and web developers who are interested to learn more about Groovy and/or Grails and are looking for real-life, working examples of how to achieve common programming tasks in Groovy and Grails.

## Groovy Series

The Groovy Series is an audio lecture on Groovy and is part of the Grails Podcast. The episodes are produced by Dierk König (Lead author of the book [Groovy in Action](#)) and [Sven Haiges](#). You will find complementary information about the topics below in Dierk's Book and we mention the page numbers at the beginning of each series to help you find the right chapters.

### **Code Examples at [snipplr.com](http://snipplr.com)**

The code examples that we discuss in the podcasts are posted to [snipplr.com](http://snipplr.com), you can [find all code snippets here](#). Please note that we cannot change the code snippets once they are published (we would run into problems with mentioned line numbers in the podcast), but you are invited to post comments. Simply log in to [snipplr](http://snipplr.com) and leave us some feedback.

### **Current Groovy Series Planning**

This is a rough plan and we might switch topics, add or delete at any time. The Groovy Series Episodes will be part of the [Grails Podcast](#) and published between regular episodes. You can subscribe to the [podcast feed](#) via: <http://han.samann.podspot.de/rss>. Once the podcast episodes are released, you can click the title links to download the mp3 files manually, too.

1. Strings & GStrings ([direct mp3](#))
  - a. Snippet 1/2: <http://snipplr.com/view/2047/groovy-series-string--gstring-12/>
  - b. Snippet 2/2: <http://snipplr.com/view/2089/groovy-series-string--gstring-22/>
2. Regular Expressions ([direct mp3](#))
  - a. Snippet 1/3: <http://snipplr.com/view/2090/groovy-series-regular-expressions-13/>
  - b. Snippet 2/3: <http://snipplr.com/view/2091/groovy-series-regular-expressions-23/>
  - c. Snippet 3/3: <http://snipplr.com/view/2092/groovy-series-regular-expressions-33/>
3. Numbers ([direct mp3](#))
  - a. Snippet 1/1: <http://snipplr.com/view/2093/groovy-series-numbers/>
4. Ranges ([direct mp3](#))
  - a. Snippet 1/1: <http://snipplr.com/view/2197/groovy-series-ranges/>
5. Lists ([direct mp3](#))
  - a. Snippet 1/1: <http://snipplr.com/view/2198/groovy-series-lists/>
6. Maps ([direct mp3](#))
  - a. Snippet 1/1: <http://snipplr.com/view/2199/groovy-series-maps/>
7. Background on Groovy Typing ([direct mp3](#)) (no code for this part)
8. Groovy Control Structures ([direct mp3](#))
  - a. Snippet 1/2: <http://snipplr.com/view/2498/groovy-series-groovy-control-structuresgroovy-truth/>
  - b. Snippet 2/2: <http://snipplr.com/view/2499/groovy-series-groovy-control-structures/>
9. Closures ([direct mp3](#))
  - a. Snippet 1/1: <http://snipplr.com/view/2866/groovy-series-closures/>
10. GroovyBeans (incl. Expando) ([direct mp3](#))
  - a. Snippet 1/1: <http://snipplr.com/view/2945/groovy-series-groovybeans--expando/>
11. GPath & Co
12. MOP
13. Builders
14. GDK - Working with Objects, Files & I/O, Threads and Processes)
15. Java Integration
16. DB Programming with Groovy
17. XML (Standalone & "Distributed" - eg- RSS, ATOM, REST)

## PLEAC Examples

[PLEAC](#) presents a suite of common programming problems from the [Perl Cookbook](#) in various programming languages. Groovy contains a complete set of examples. Here are links to the these examples:

- Manipulating [Strings](#), [Numbers](#), [Dates](#), [Arrays](#), and [Maps](#)
- [Pattern matching and text substitutions](#)
- [File Access](#)
- [File Contents](#)
- [Directories](#)
- [Subroutines](#)
- [References and Records](#)
- [Packages, Libraries, and Modules](#)
- [Classes and Objects](#)
- [Database Access](#)
- [User Interfaces](#) including screen addressing, menus, and graphical applications
- [Process Management and Communication](#)
- [Sockets](#)
- [Internet Services](#) including mail, news, ftp, and telnet
- [CGI Programming](#)
- [Web Automation](#)

## FAQ

- [Class Loading](#)
- [FAQ - Classes and Object Orientation](#)
- [FAQ - Closures](#)
- [FAQ - Collections, Lists, etc.](#)
- [FAQ - GSQL](#)
- [FAQ - RegExp](#)
- [General](#)
  - [How can I edit the documentation](#)
- [Language questions](#)
  - [Can I break a Groovy statement into multiple lines anyway I want?](#)
  - [How can I dynamically add a library to the classpath](#)
  - [Why does == differ from Java](#)
- [Learning about Groovy FAQ](#)
- [License Information](#)
- [Runtime vs Compile time, Static vs Dynamic](#)

## Class Loading

**I'm getting an "unable to resolve class [My Class](#)" error when I try to use a class contained in external .groovy file.**

If the problem goes away when you apply groovyc to the .groovy file (compiling it), and you're running on windows, the problem is probably spaces in the current directory structure. Move your files to a path without spaces (e.g., c:\source rather than c:\documents and settings\Administrator\My Documents\source).

## How do I load jars and classes dynamically at runtime?

Use the groovy script's classLoader to add the jar file at runtime.

```
this.class.classLoader.rootLoader.addURL(new URL("file:///path to file"))
```

Then, use `Class.forName` to load the class.

```
def cls = Class.forName("com.mysql.jdbc.Driver").newInstance();
```

## FAQ - Classes and Object Orientation

### Classes and Object Orientation

*How do you include groovy classes within other classes?*

Groovy classes work exactly like java classes. For example, to include the class "TestClass" in your program, ensure that it is in a file called "TestClass.groovy," and in a path seen by your CLASSPATH environment variable (or command line). Remember that JAR files need to be included explicitly by name.

## FAQ - Closures

### Closures

*What problem do closures solve? Why have closures?*

At one level they just allow internal iterators instead of external ones. This alone is really nice because looping is a lot cleaner. With iterators for example you do all the work, because if you want to execute the loop of the body, you have to take responsibility for `hasNext()` and `next()`.

So its basically providing the body of a loop or a callback etc which will execute within the original scope.

Anonymous classes can seem like they are in fact closures, but they have limitations that cause annoying things like having to declare all your variables as `final`. The compiler just creates a synthetic constructor that takes any variables you are going to reference.

For me the main benefit of closures is that they allow you to write code for collections with a lot less boilerplate.

```
accounts.findAll { it.overdrawn &&
!it.customer.vip }.each { account ->
    account.customer.sendEmail("Pay us now!!")
}
```

## FAQ - Collections, Lists, etc.

### Collections, Lists, etc.

*Why don't return statements work when iterating through an object?*

The `{...}` in an `each` statement is not a normal Java block of code, but a *closure*. Closures are like classes/methods, so returning from one simply exits out of the closure, not the enclosing method.

*How do I declare and initialize a list at the same time?*

Syntax:

```
def x = [ "a", "b" ]
```

*How do I declare and initialize a traditional array at the same time?*

Syntax:

```
String[] x = [ "a", "qrs" ]
```

or

```
String[] x = [ "a", "qrs" ] as String[]
```

or

```
def x = [ "a", "qrs" ] as String[]
```

*Why does myMap.size Or myMap.class return null?*

In Groovy, maps override the dot operator to behave the same as the index[] operator:

```
myMap["size"]="ONE MILLION!!!";  
println myMap.size           // outputs 'ONE  
MILLION!!!'
```

```
//use the following:
```

```
println myMap.@size         // '1'  
println myMap.size()       // '1'  
println myMap.getClass()   // 'class  
java.util.HashMap'
```

### *Why is my map returning null values?*

Chances are, you tried to use a variable as a key in a map literal definition. Remember, keys are interpreted as literal strings:

```
myMap = [myVar:"one"]  
assert myMap[ "myVar" ] == "one"
```

Try this (note the parentheses around the key):

```
myMap = [ (myVar) : "one" ]
```

## **FAQ - GSQL**

### **GSQL**

*Show me a simple GSQL application?*

Here you are:

```
import groovy.sql.Sql

sql = Sql.newInstance("jdbc:hsqldb:mem",
"sa","","org.hsqldb.jdbcDriver")
sql.execute('create table TableA (FirstName
varchar(40),&nbsp; LastName varchar(40))')
sql.execute('INSERT INTO TableA
(FirstName,LastName) values
(?,?)', \['Stan', 'Juka' \])

sql.eachRow('select * from TableA') {
    println "TableA row: ${it.firstName},
${it.lastName}"
}
```

The output should be:

```
TableA row: Stan, Juka
```

*Why does my statement doesn't work? (GString Special Treatment)*

Why the INSERT in this code fails?

```
...
values = "'Stan', 'Juka'"
insertSQL = "INSERT INTO TableA
(FirstName,LastName) values ($values)"

sql.execute(insertSQL)
```

Because the *insertSQL* is a GString. if you make it a String like this

```
String insertSQL = "INSERT INTO TableA
(FirstName,LastName) values ($values)"
```

it will work. Or you can do it like this:

```
firstName = 'Stan'
lastName = 'Juka'
insertSQL = "INSERT INTO TableA
(FirstName,LastName) values
($firstName,$lastName)"
```

In the last code snippet, GSQL will automatically create a prepared statement for you and run it.

---

#### ***Can you make this work with an Oracle Database?***

I modified the code to get it to work with Oracle and for readability.

The `Sql.newInstance` will connect to an Oracle database SID called XE installed on localhost at port 1521. The database user is "username" and the password is "password".

In order to get this to work you will have to install the the latest jdbc.jar file from Oracle's website. You need to install the .jar file into the

lib directory in your GROOVY\_HOME directory:

```

import groovy.sql.Sql

sql = Sql.newInstance(
    "jdbc:oracle:thin:@localhost:1521:XE",
    "username", "password",
    "oracle.jdbc.OracleDriver" )

createTable = 'CREATE TABLE TableA (
    FirstName varchar2( 40 ), LastName varchar2(
    40 ) )'
insertIntoTable = 'INSERT INTO TableA (
    FirstName, LastName ) values ( ?, ? )'
selectStatement = 'SELECT * FROM TableA'

sql.execute( createTable )
sql.execute( insertIntoTable, [ 'Bill',
    'Lyons' ] )

// the eachRow method is an Iterator
sql.eachRow( selectStatement ) {

    println "TableA row: ${ it.firstName }, ${
    it.lastName }" }
}

```

The code does the following:

1. Creates TableA in username's schema
2. Inserts a row into TableA
3. Performs a SELECT \* FROM TableA;
4. Returns "TableA row: Bill Lyons"

**FAQ - RegExp**

## RegExp

`matcher.matches()` returns false

Why does this code fail?

```
def matcher = "/home/me/script/test.groovy"
=~ /\.groovy/
assert matcher.matches()
```

Because you think you do something like "Oh dear it contains the word!", but you're confusing *matches* with *find*

From Javadoc: [http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Matcher.html#matches\(\)](http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Matcher.html#matches())

public boolean matches()

Attempts to match the **entire** region against the pattern.

...

So  `/\.groovy/` is just a subsequence.

You must use

```
def matcher = "/home/me/script/test.groovy"
=~ /\.*.groovy/
```

*What is the difference between  `=~`  and  `==~`  ?*

- `~`  is the [Pattern](#) symbol.
- `=~`  means [matcher.find\(\)](#)
- `==~`  means [matcher.matches\(\)](#)

Pattern, Matcher ?

A pattern is not very useful alone. He's just waiting input to process through a *matcher*.

```
def pattern = ~/groovy/
def matcher = pattern.matcher('my groovy
buddy')
```

Matcher can say a lot of thing to you:

- if the *entire* input sequence matches the pattern, with [matcher.matches\(\)](#) ;
- if just a *subsequence* of the input sequence matches the pattern, with [matcher.find\(\)](#).

A matcher with `/groovy/` pattern **finds** a matching subsequence in the 'my groovy buddy' sequence. On the contrary the whole sequence doesn't **match** the pattern.

```
def m = c.matcher('my groovy buddy')
assert m.find()
assert m.matches() == false
```

Application: to filter a list of names.

```
def list = ['groovy', '/a/b/c.groovy',
'myscript.groovy', 'groovy.rc', 'potatoes',
'groovy. ']
// find all items whom a subsequence matches
/groovy/
println list.findAll{ it =~ /groovy/ } //
=> [ "groovy", "/a/b/c.groovy",
"myscript.groovy", "groovy.rc", "groovy." ]

// find all items who match exactly /groovy/
println list.findAll{ it ==~ /groovy/ } //
=> [ "groovy" ]

// find all items who match fully
/groovy\..*/ ('groovy' with a dot and zero
or more char trailing)
println list.findAll{ it ==~ /groovy\..*/ }
// => ["groovy.rc", "groovy." ]
```

A little tilde headache ? Remember like this

~

the pattern

<code>≈</code>	roughly as the pattern (easy to write)
<code>≈≈</code>	more than roughly, <b>exactly</b> as the pattern (think hard...)

## General

- [How can I edit the documentation](#)

## How can I edit the documentation

The entire Groovy website is stored in this [wiki](#)

The home page is called [Home](#) then the navigation links on the left are on a magic page called [Navigation](#) and the top right navigation links are on a magic page called [QuickLinks](#).

Hopefully now anyone can contribute to the documentation.

If you ever edit a page and wonder why its not yet been updated on the <http://groovy.codehaus.org/> site well it could be cached. To view a latest greatest page just add the 'refresh=1' to your URL. e.g.

<http://groovy.codehaus.org/?refresh=1>

wiki

## Language questions

- [Can I break a Groovy statement into multiple lines anyway I want?](#)
- [How can I dynamically add a library to the classpath](#)
- [Why does == differ from Java](#)

## Can I break a Groovy statement into multiple lines anyway I want?

The simple answer is no. For example, a code

```
aaa = 7
    -1
assert aaa == 6
```

will fail. See more details about new lines in the [Groovy Language Specification](#) (section 3.11.1 Significant Newlines).

## How can I dynamically add a library to the classpath

Use `getRootLoader().addUrl([Some URI])`

See [How to get a RootLoader](#)

See [RootLoader javadoc](#)

*Sample: Dynamic JDBC Driver Loading*

```
import Groovy.sql.Sql
this.class.classLoader.rootLoader.addURL(
new URL("file:///d:/drivers/ojdbc14.jar") )
def
driver="oracle.jdbc.driver.OracleDriver";
def sql =
Sql.newInstance("jdbc:oracle:thin:@hostname:
port:schema", "scott", "tiger", driver);
```

### Why does == differ from Java

This is described [here](#).

Basically in Java == with primitive types means use equality. For object types == means test identity.

We found when working with Groovy that if we kept those semantics and folks used dynamic typing as follows

```
def x = 2 * 2
if (x == 4) {
...
}
```

They would get surprising results, as they often mean equality based on value, such as in the above, rather than identity. Indeed folks rarely ever use identity comparisons.

So to avoid many common gotchas and confusions, we've made == mean equals, the meaning most developers use, and we use this for both primitive types and for object types and across both static and dynamic typing to simplify things.

Currently if you really want to compare identities of the objects, use the method is(), which is provided by every object.

```
if (x.is(4)) {
... // never true
}
```

The above condition is never true, since the Integer object in x (which is the result of the computation above) is not identical to the Integer object with value 4 that has been created for the comparison.

## Learning about Groovy FAQ

### This FAQ hopes to answer common questions for users of Groovy

#### *What is Groovy?*

- [Groovy](#) is trying to provide a high level language (like Ruby, Python or Dylan) that maps cleanly to Java bytecode.
- It needs to work with Java objects, and the root of all the object trees is [java.lang.Object](#).
- The syntax will be Java friendly, but doesn't have to be backwards compatible.
- Groovy will sit on top of [J2SE](#).

#### *Where can I get more information on Groovy?*

The current user documentation for Groovy is available from <http://groovy.codehaus.org>

#### *What if the documentation is wrong?*

Anybody can change these pages, just click on the little **Edit** link on the right of each page (you then have to signup/login if you haven't already).

#### *How can I get a binary version of Groovy?*

Download latest distribution as a [zip](#) or [tgz](#) file and then follow the [installation instructions](#)

#### *How do I embed Groovy in my own programs?*

Download latest [groovy-all.jar](#) and place it in your classpath.

#### *How can I grab the sources?*

See [Building Groovy from Source](#).

## License Information

Groovy is licensed under the Apache 2 license:

<http://www.apache.org/licenses/LICENSE-2.0.html>

## Runtime vs Compile time, Static vs Dynamic

### Runtime vs Compile time, Static vs Dynamic

#### *I misspelled a method call, but it still compiled. What gives?*

Take this simple script as an example:

## Greet.groovy

```
class Greet {
    def salute( person ) { println "Hello
    ${person.name}!" }
    def welcome( Place location ) { println
    "Welcome to ${location.state}!" }
}

g = new Greet()
g.salude()                //misspelling
g.welcome( 123 )         //wrong argument
type
```

Note that running `groovyc Greet.groovy` does not produce any errors. Instead, a `MissingMethodException` is thrown at runtime.

This is because Groovy is a dynamic language. Several other things could be happening to make this code valid at runtime. Using the `MetaClass`, you could add a `salude()` method to the `Greet` class at runtime. You could also add a `state` property to `Number`, which would make the `welcome(..)` call valid. See [ExpandoMetaClass](#) and [TMPGroovy Categories](#).

Type checking to the rescue

If all you need is a scripting language and that you don't rely on the dynamic behaviour, since Groovy 2.0.0, you can add an annotation that will activate type checking:

## Greet.groovy

```
@groovy.transform.TypeChecked
class Greet {
    def salute( person ) { println "Hello
    ${person.name}!" }
    def welcome( Place location ) { println
    "Welcome to ${location.state}!" }
}
```

The compiler will report errors at compile time instead of runtime. See [GEP 8 - Static type checking](#) for details.

***Will I get a performance boost if I statically type all of my fields and methods?***

Actually, no. The way Groovy method selection is done, it actually takes **longer** if you provide lots of static type information. This could possibly change in the future, but as of Groovy 1.1 this is not the case. See [this thread](#) for more info.

***But can't you add warnings so I can see missing methods at compile time?***

In theory, we could. It would *only* work for methods available at compile time, and *only* for fields and parameters that you have strongly typed. But as we mentioned above, that hurts performance! Plus, there are a number of frameworks that rely heavily on dynamic methods (i.e. [GORM](#)). In this case, you would get gobs of warnings, and likely just start ignoring them because it is just noise.

It might be scary to do away with all of your static typing and compile time checking at first. 😊 But many Groovy veterans will attest that it makes the code cleaner, easier to refactor, and, well, more *dynamic*. You should make all efforts to use unit tests to verify your intended behavior. Also keep in mind that Groovy also offers a slew of [features](#) to make unit testing easier as well.

***Can't we provide some sort of "NotDynamic" flag to speed up method invocation?***

Since Groovy 2.0.0, you can annotate your code with `@CompileStatic`. Be warned that while this will improve performance, it changes the semantics of your code. See [GEP 10 - Static compilation](#) for details.

## Roadmap, Discussions and Proposals

Groovy has an active community from which many ideas and discussions arise. Initial discussions or clarifications occur within the [Mailing Lists](#). Very specific suggestions can often be included in the Groovy [Issue Tracker](#). Ideas which require wider or on-going discussion or greater formalism are documented here.

- *Discussions* are ideas or potential changes in their early formation.
- *Proposals* are ideas that have reached a greater level of maturity but still require discussion or wider communication.
- The *Roadmap* outlines things we have committed to do or hope to do soon.

- [Discussions](#)
  - [Feature Wish List](#)
  - [New Groovy Console Wish List](#)
- [Proposals](#)
  - [Annotation usage with properties](#)
  - [AST Macros and Annotations](#)
  - [C Sharp Groovinator](#)
  - [GJIT - Groovy Just-In-Time Compiler](#)
  - [Grapes and grab\(\)](#)
  - [Groovy realtime archive internet lookup](#)
  - [JSR-295 Binding Builder](#)
  - [Macro Use Cases](#)
  - [MetaClass Redesign \(by blackdrag\)](#)
  - [MetaClass Redesign \(by blackdrag\) part 2](#)
  - [MOP 2.0 ideas](#)
  - [Multiple Assignment Proposal](#)
    - [Multiple Assignment](#)
  - [Multiproject builds](#)
  - [Proposed Website Revamp](#)
  - [Web based GroovyShell](#)
  - [Miscellaneous Feature Requests and Enhancements](#)
  - [Groovy 2.0 modularization](#)
- [Roadmap](#)
  - [notes on JDK 7 and JDK 8 future](#)
  - [Not Yet Documented](#)
    - [Gldapwrap - Usage](#)

## GEP

Current formal proposals:

**Could not access the content at the URL because it is not from an allowed source.**

<http://jira.codehaus.org/sr/jira.issueviews:searchrequest-rss/temp/SearchRequest.xml?component=12771&pid=10242>

You may contact your site administrator and request that this URL be added to the list of allowed sources.

## Discussions

### Current Discussions

- [Feature Wish List](#)
- [New Groovy Console Wish List](#)

### Feature Wish List

Please use this page to document all the ideas and wishes you'd like to see in Groovy.

---

#### *Non-breaking Switch Enhancements*

Objective: get rid of a lot of if then else statements; eliminate unnecessary syntax; no impact on "java-like" groovy

code.

*Multiple parameters/targets*

```
// multiple parameter switch
// '?' == true/any
def x = { a, b -> switch(a,b) {
  case (0, 0) : return 0
  case (0, 1), (1, 0) : return 1
  case (1, ?), (?, 1) : return 2
  default : return 3
}}
```

*Alternate '->' Syntax, Optional 'default', Assign Result*

```
// in a switch '->' implies break,
// 'default' is optional,
// and they can have a result
def x = { a ->
  switch (a) {
    case 0 -> println 'a'
    case 1 -> println 'b'
    -> println 'c'
  }

  def b = switch(a) {
    case 0 -> 'a'
    case 1 -> 'b'
    -> 'c'
  }
}
```

```
// a short form of the above for function
pattern matching
def x = { a, b
  case (0, 0) -> 0
  case (0, 1),
        (1, 0) -> 1
  case (1, ?),
        (?, 1) -> 2
  default -> 3
}
```

Here the '->' form implies 'return' instead of 'break' (same result at this level)

Another example:

```
// case works as it does currently,
executing the closure or calling isCase()
def qsort = { ls
  case {ls.size() < 2} -> ls
  ->
  def (x, xs) = [ls.head(), ls.tail()]
  def smaller, larger = []
  xs.each { it<=x ? smaller<<it : larger<<it
  }
  qsort(smaller) + x + qsort(larger)
}
```

Here you can see the special "case" is executed when the list size is less than 2, otherwise it looks like a normal closure.

The above proposal does not break any existing code, does not add any keywords, does not break compatibility with java ("paste java in to groovy and it work", etc).

The multi-parameter switch statement and '->' in a switch so it has a result is where the core of the work would be. The 'case' statement on a closure definition for pattern matching like capability is just syntactical sugar.

--krsmes (Apr 2010)

---

- **New function for integers to reverse bits.**
- **Concatenate string with null object.** 'Test'+null will produce 'Test'.
- **Function/Global method 'NVL'. This method avoid null/empty string value of object.** E.g. NVL(null,'Test')='Test' and NVL("",'Test')='Test'
- **Joining of list without null elements.**

e.g. [null,'Hello','world',null].join(' ') will produce 'Hello world'

- **Simple and *Robust* way of executing external processes instead of using ProcessBuilder.** Make the convenient string.execute() handle blocked IO streams / threading so users are not required know or understand about ProcessBuilder
- **More Enumerable methods** for lists, arrays and maps (like map, pluck, invoke, ...)
- Make **logical operators** (||, &&...) **return the value** instead of the boolean equivalent eg.

```
def x = '' || 'test'
assert x == 'test'

def x = null || 15
assert x == 15
```

- **Make map creation more versatile.** Add constructors that allow creation of a map from 2 collections "HashMap(keys, values)" or a list of entries "HashMap(itemlist)".
- **Make a list and create an auto mapping:**

eg.

```
//Today:
def list = ["key":"value", 1:1, 2:2,
"noAuto":"noAuto"]
print(list[1]) // will print 1
//Would like to have:
```

```
def autoMapList = ["key":"value", 1, 2,
"auto"]
print(autoMapList[1]) // will print 1
```

- would like methods to return more than one return value (a la ruby)

```
//Today:
def mytest { return [1,2] }

c = mytest()
a = c[0]
b = c[1]

wanted behavior:

def mytest { return 1,2 }

a,b = mytest()
```

### ***Property reference operator***

Groovy 1.0 already has the method reference operator:

```
class X {  
    def doSomething() { println "hello" }  
}  
  
def obj = new X()  
  
def methRef = obj.&doSomething  
methRef()
```

However we do not have anything similar for properties. i.e. we should be able to get a reference to a wrapper object for any object property, and be able to get and set the property via the reference, and get the original property name and owning bean:

```

class X {
    def someValue
}

def obj = new X()

def propRef = obj.&someValue // same syntax,
but runtime knows it is a property
println propRef.value
propRef.value = 999
println propRef.name // outputs "someValue"
assert propRef.object == obj

// static property equivalent
def propRef = X.&someValue
// Now you need to pass in an instance
propRef.setValue( new X(), 123)

```

This would be useful in Grails and other apps that need the user to specify in their code a reference to another property, for example in GORM declaring the list of "embedded" properties:

```
class AddressBookEntry
```

Property access in groovy is not problematic using GPath or subscript operator anyway, so this may not seem so useful. However if we could get some compile-time checking of the validity of the property that would be a nice win. Perhaps combining with the @ operator:

```
class AddressBookEntry {  
    Address home  
    Address work  
    String name  
  
    static embedded = [ &@home, &@work ]  
}
```

Altho a tad heiroglyphic, this would allow the compiler to ensure that "home" and "work" do exist as declared properties on the class or its ancestors, and hence fail fast.

#### *named parameters everywhere*

the proposal is that for functions defined inside groovy or where the debug information is available can be called with the map construct to provide named parameter calling everywhere

```
void helloworld(name, title) {println "hello  
$title $name"}  
can be called as  
helloworld( title:"mr.", name:"koen" )
```

this feature would largely improve general readability of code and imho very often enough information should be available to perform the matching (certainly for groovy functions, very often for java binaries with appropriate debug information)

#### *smarter nulls*

```
def hello = "hello"  
def test = null  
println "$hello$test"
```

should print "hello" iso "hellonull"

#### *variable constraints*

Keep dynamic typing while adding real type safety with constraints on variables that all must be true. Though not all these constraints are actually useful they are there for illustrative purposes.

```
percent : as Integer, >= 0, <= 100  
total : != 0  
...  
percent = num/total
```

#### *Embedded XML*

The programming language [Scala](#) has built-in support for XML. You can create something like this (taken from the Scala overview document):

```

val labPhoneBook =
  <phonebook>
    <descr>Phone numbers of<b>XML</b>
hackers.</descr>
    <entry>
      <name>Burak</name>
      <phone where="work"> +41 21 693
68 67 </phone>
      <phone where="mobile"> +41 78
601 54 36 </phone>
      </entry>
    </phonebook>;

```

#### *Decoupling of static and instance methods*

Also taken from the [Scala](#) programming language. It doesn't have static methods. All the static methods should be placed in a singleton. Static members are not a part of instance variables, so you shouldn't have to declare them in the same class.

#### *New way of declaring Expando*

Could be just like in anonymous types in C#. For example:

```

def foo = new (prop: "value")
foo.bar = "bar"

```

#### **First Class Support for User-Defined Boolean Types**

There is already quite good support for user-defined numeric data (e.g. classes that act like numeric data in expressions) using the current support for operator overloading. But if one needs to define a class that operates in expressions like a Boolean, it can't be done. The following changes would make this possible and these seem consistent with the overall idea and design of Groovy:

1. Enable operator overloading for logical AND, logical OR, logical NOT. For logical AND and logical OR, the deferred evaluation of the right operand can be maintained by passing a Closure to the method. The signatures might be something like this:

```
Boolean land(Closure)
Boolean lor(Closure)
Boolean lnot()
```

2. Enable assignment operator overloading. This is needed to allow a semantic like

```
MyBooleanClass bool = true
```

Note that overriding the assignment operator makes it significantly easier to replace numeric types too since one can handle cases like

```
MyNumericClass num = 10
```

which is not possible today.

3. Today due to "Groovy Truth" processing, any non-null object reference (that is not a Boolean) will evaluate to true. To enable user-defined Boolean type data, Groovy Truth would need to be modified to optionally "unbox" a non-null object reference into a Boolean value if it has a `booleanValue()` method. This would allow one to do something like this

```
MyBooleanType bool = something(); if bool || somethingElse() { ... }
```

In such a case, the `bool` instance can be either false or true and can be directly used for control flow and other operations that normally expect a Boolean.

With these changes, user-defined boolean replacement classes would get first-class support in the language.

### ***Smalltalk-like syntax for methods***

I've seen that in Groovy 1.5 it was possible to avoid using parenthesis when calling a method. It was made to simplify the reading in a more natural way...

Nevertheless I'm not sure that the goal is today reached... The groovy syntax was primarily settled to make the step from Java easier for java coders... Along the versions, some syntactic sugar was added to fulfill new needs... to reach the point where going from jav to groovy is not so straightforward... Personally it is not an issue, but I would like to point out that if we differ more and more from java, why not adopting a syntax which is more accurate and simpler?

Personally, I find the Smalltalk syntax simpler and more elegant... I think that we could add some syntactic "tournures" of Smalltalk in Groovy 2.0, especially for defining new methods....

**definition example :**

```
def print: anObject on: aStream {  
  ....  
}
```

or

```
def print: Customer aCustomer on: FileStream  
aStream {  
  ...  
}
```

or better

```
def print: aCustomer as Customer on:  
aStream as FileStream {  
  ...  
}
```

use example :

```
def aCustomer = new Customer();  
WriteStream aStream = new FileStream();  
print:aCustomer on: aStream ;
```

*Expect for Groovy*

[What is expect?](#)

Expect has been already ported to other languages:

[Expect for Perl](#)

Why can't we have it built in?

My proposal:

[Expect for Groovy](#)

#### ***Automatic constructor for immutable Groovy Beans***

Automatically create a constructor for all final properties of a Groovy Bean.

```
class Data {  
    final Integer id  
    final String name  
}
```

And then:

```
def myData1 = new Data(5, "John")  
def myData2 = new Data(8, "Jane")
```

There should be no setters generated, and not be possible to change the values of id or name after object creation.

It would be even better if you could use named parameters to the constructor as well:

```
def myData1 = new Data(id: 5, name: "John")  
def myData2 = new Data(name: "Jane", id: 8)
```

### **New Groovy Console Wish List**

This page is dedicated to the discussion of what the Groovy Shell (aka. `groovysh`) and Groovy Console (aka. `groovyConsole`) should be able to do in the future. So please go on and add suggestions.

#### **Wishlist for the command-line Groovy Shell (aka. `groovysh`)**

- keep imports (**DONE** see `import` and `show imports` commands)
- remove the requirement for the "go" command (**DONE**)
  - proposal: read lines until one is entered WITHOUT a trailing semicolon or as long as there are open parentheses; change prompt for continuation lines
- the value of last expression evaluated is made available in a variable. The name of the variable is short, e.g. "res". It is also automatically printed to the console (including type?). (**DONE** The last result is saved as the `{_}` variable)
  - alternate name: `"_"` or `"_r"`

from Dierk

- code completion for known methods, DGM, and symbols used so far
- dump transcript to file (**DONE** see `record`).
- nice to have: dump classes and methods that were previously defined (**DONE** See `show classes` and `show variables`, methods are wrapped in a closure and bound to variables)
- nice to have: run code in new jdb console

from Guillaume

- all the above, and...
- possibility to discard imports (since we can keep imports, let's flush them too) (**DONE** See `purge imports`).
- Crazy idea: I'm wondering whether we could leverage ANSI colours in the shell so that we can have some text in colours (perhaps even syntax highlighting???) (**DONE** sans syntax highlighting right now)

from Jürgen:

- learn from Python and especially IPython (`sys.displayhook` etc.) (**DONE** There is a `errorHook` and `resultHook` now).
- (GNU) readline support (**DONE** current trunk uses `jline` for rich buffer editing and history).

---

additionally: enhance ObjectBrowser to execute selected methods and change the "object under inspection" to the result of that method call. Keep an expression of how to get to that object (e.g. `GPath`). Let ObjectBrowser return this expression and make `groovysh` and `groovyConsole` aware of it such that `groovysh` user can use the `inspect` command to interactively construct such expressions.

**Wishlist for the Swing Console (aka. `groovyConsole`)**

- basic syntax highlighting in source code (**DONE** in 1.5\*)\*
- line numbers (be able to turn on / off)
- be able to specify some `.jar` files that will be included into the classpath (**DONE** in 1.6)
- enable standard-de-facto `ctrl+z` for undo and `ctrl+y` for redo
- tabs
- code formatting

Debug features:

- step-by-step execution of statements
- improve variables insight for clarity

## Proposals

### All Proposals

- [Annotation usage with properties](#)
- [AST Macros and Annotations](#)
- [C Sharp Groovinator](#)
- [GJIT - Groovy Just-In-Time Compiler](#)
- [Grapes and grab\(\)](#)
- [Groovy realtime archive internet lookup](#)
- [JSR-295 Binding Builder](#)
- [Macro Use Cases](#)
- [MetaClass Redesign \(by blackdrag\)](#)
- [MetaClass Redesign \(by blackdrag\) part 2](#)

- [MOP 2.0 ideas](#)
- [Multiple Assignment Proposal](#)
- [Multiproject builds](#)
- [Proposed Website Revamp](#)
- [Web based GroovyShell](#)
- [Miscellaneous Feature Requests and Enhancements](#)
- [Groovy 2.0 modularization](#)

## Proposals By State

### *Draft*

[Grapes and grab\(\)](#)

---

### *Accepted*

No content found for label(s) gep-accepted.

---

### *Withdrawn*

No content found for label(s) gep-withdrawn.

---

## Proposals By Groovy Version

### *Groovy 1.6*

No content found for label(s) gep-v1-6.

---

### *Groovy 1.7*

[Grapes and grab\(\)](#)

---

### *Groovy 2.0*

No content found for label(s) gep-v2-0.

---

## Annotation usage with properties

### Background

Groovy's property mechanism provides a short-hand which simplifies defining typical *JavaBean-style* classes. You simply define the property with no explicit visibility and it is replaced at the bytecode level by a private backing variable, a public setter and a public getter. To the Java world, all three parts are visible. In the Groovy world, conceptually it is OK to just think about the property - Groovy's property notation comes in to play so that using the setter or getter looks like using the property as if it was a field.

This works well but introduces a slight complication with annotations. When using the non-property approach, there are three places in the source code where an annotation may be added (the field, the setter and the getter). With the property approach (idiomatic Groovy) there is just one place available in the source to place an annotation.

### Current Behaviour

Currently if an annotation is applied to a property, the compiler will attempt to apply the annotation to the field, the setter and the getter. If any of these fails (e.g. some annotations may apply only to methods or only to fields) the code won't compile.

**PROPOSAL: Smarter annotation assignment**

Use the `java.lang.annotation.Target` of any annotations to determine where the annotation will be included in the final class.

If the `@Target` of the used annotation marks the annotation as usable only on `FIELD` level, then the annotation will be attached to the generated field.

If the `@Target` of the used annotation marks the annotation as usable on `METHOD` level, then the annotation will be attached to the generated getter and setter.

If the `@Target` of the used annotation marks the annotation as usable on both `FIELD` and `METHOD` level, then the annotation will be attached to all the generated class elements.

Obviously, if multiple annotations are used, there is the possibility of the annotations being split across the generated class elements, e.g. I can use both a field and a method annotation. The field annotation will be attached to the field (and ignored for the getters and setters) while the method annotation will be attached just to the methods.

#### **PROPOSAL: Additional annotation classification mechanism**

Sometimes it is desirable to be more selective with annotation than is possible even with smarter annotation assignment, e.g. placing an annotation on just one of the setter or getter. Sometimes an annotation might be possible on both fields and methods but in a particular example we want to apply it to just one. To achieve this we need additional syntax to express how we want to classify an annotation.

One way to do this is to use a synthetic annotation as a prefix to the real annotation. A `@Target` annotation appears in Groovy code at the source level, but it is only synthetic. It has no real definition. It will be like a compiler flag only, and completely ignored in the output.

The `@Target` annotation allows the user to further refine where the compiler should attach an annotation. After it has been used by the compiler, it is completely discarded. The Groovy `@Target` annotation will further specify if an annotation will be attached to a `FIELD`, `GETTER` or `SETTER`, but will never contradict the Java `@Target` annotation. In case such a contradiction occurs the Groovy compiler must emit an error.

Last, but not least I should mention that there will be no naming conflict between the Java `java.lang.annotation.Target` annotation and the Groovy `Target` annotation (probably `groovy.lang.annotation.Target`), because they will be living in completely different context.

An example:

```
@Target ( [ FIELD ] )  
@AutoPopulate ( defaultValue = "bar" )  
@Target ( [ GETTER ] )  
@ManagedAttribute ( currencyTimeLimit = 15 )  
@Target ( [ SETTER ] )  
@ManagedAttribute ( persistPolicy = "OnUpdate" )  
int base = 10
```

As an alternative, we could come up with some other syntax to achieve the same result, e.g. here might be some

other ways to write the above:

```
@[FIELD]AutoPopulate(defaultValue="bar")  
@[GETTER]ManagedAttribute(currencyTimeLimit=  
15)  
@[SETTER]ManagedAttribute(persistPolicy="OnU  
pdate")  
int base = 10
```

Or this:

```
@AutoPopulate[FIELD](defaultValue="bar")  
@ManagedAttribute[GETTER](currencyTimeLimit=  
15)  
@ManagedAttribute[SETTER](persistPolicy="OnU  
pdate")  
int base = 10
```

Or this (using a synthetic annotation parameter):

```
@AutoPopulate(groovyTarget='field',  
defaultValue="bar")  
@ManagedAttribute(groovyTarget='getter',  
currencyTimeLimit=15)  
@ManagedAttribute(groovyTarget='setter',  
persistPolicy="OnUpdate")  
int base = 10
```

AST Macros and Annotations



## Work In Progress

Please post your comments on the Groovy developer list!

## Introduction

This text here is based on ideas from Jochen Theodorou (see [chitchat-with-groovy-compiler](#) and [ast-macros-and-mixins](#))

Right now, there is no good macro preprocessor for Java. Annotations somehow come close but they don't really fit the bill. In the Java VM, annotations are a runtime feature. You cannot enhance an existing class; only create new ones. This means that you cannot add setters and getters to a class.

When you look at OR Mappers, they even do this at runtime, so there is no way to see what is actually happening: When the error happens, the code which is executed can be completely different than what you see in the source. Even decompiling the class file will not help anymore because the information isn't there, yet. It's only added when the classloader reads the file.

So from a certain point of view, Sun's solution is the worst of all worlds: Your code is changed at a point in time when you can't see it anymore and you cannot move the modification in the compile cycle because the API simple doesn't allow it.

## The Goal

To know where you want to go, you must have a goal. The goal here is to reduce the amount of code to write for a certain feature. Specifically, the idea is to be able to move common, repeated code into a single place and be able to reference it easily.

The code must be more flexible than a method call and easier to manage than cut&paste.

### *Example 1: Bound Properties*

A bound property in a Java bean is a field which sends notifications to listeners when it is changed. This means it is made up of these parts:

- There is a list of listeners who are interested in changes
- The field itself
- Methods to add and remove listeners
- Setter code which changes the value and notifies the listeners if the value has changed

### *Example 2: Merge Java and SQL*

OR Mappers will only get you so far. While they will solve many or all problems, they also introduce new ones:

- You have to learn to use the mapper
- You must still understand how a database works
- The mapper will try to connect the Java semantics to the Database semantics. This is not always straightforward. For example, a table might define a non-unique primary key. There is no way to map this to a Java map where primary keys must be unique or you will overwrite objects.
- When a mapper doesn't support a specific corner case, you're in trouble. Mappers are often quite monolithic

and they shove their tendrils in many places. Usually, you can't split it apart, injecting your own code in certain places.

So what do we expect from AST Macros in this case?

- They shouldn't get in the way of the developer. If she chooses to use an OR Mapper, she should be free to do so.
- It should help to manage all the cases which the OR Mapper doesn't handle well.
- It should give aid to interface the Groovy code with the OR Mapper.
- It should allow to write code which directly interfaces the DB (for example, when you have to execute some special SQL which the OR Mapper simply can't do).

Some simple examples:

- counting rows
- loading objects from a DB

## Examples

Before we look at solutions, let's look at what the code ought to do in the end.

### *Example 1: Bound Properties*

```
class A {
    @BoundProperty int x
}
```

should become

```
import javax.beans.*;

class A {
    // the following is added only once per
class
    PropertyChangeSupport
propertyChangeSupport

    void
addPropertyChangeListener(PropertyChangeListener
```

```
ner listener) {

propertyChangeSupport.addPropertyChangeListener(listener)
}

void addPropertyChangeListener(String property, PropertyChangeListener listener) {

propertyChangeSupport.addPropertyChangeListener(property, listener)
}

void
removePropertyChangeListener(PropertyChangeListener listener) {

propertyChangeSupport.removePropertyChangeListener(listener)
}

void removePropertyChangeListener(String property, PropertyChangeListener listener) {

propertyChangeSupport.removePropertyChangeListener(property, listener)
}

PropertyChangeSupport[]
getPropertyChangeListeners() {
return
```

```
propertyChangeSupport.getPropertyChangeListener  
ners  
    }  
  
    // the following is added per each  
annotated proeprty  
    private int x  
  
    void setX (int x) {  
propertyChangeSupport.firePropertyChanged('x  
, this.x, this.x = x)  
    }  
  
    int getX() {
```

```
        return x;
    }
}
```

specifically:

- When using code completion, the additional fields and methods should be visible.
- The added code should be lean and fast
- ~~It should not get in the way of existing user code, for example, it should be possible to define a custom setter which gets wrapped by the code above~~
- ~~If a custom setter exists, it should be possible to invoke it before the comparison (so you can define setters which convert the value before assigning it)~~

This leads to a couple of demands which an AST Macro Processor (AMP) must met:

- It must be able to see all fields and methods, no matter if the user supplied them or an AMP added them.
- It must be able to add new methods and fields and static code to an existing class and presumably also to classes created by an AMP
- It must be able to reorder source code or at least AST Nodes.

In a perfect world, an AMP should be able to modify the code on a source level and pass it back to an IDE, for example, so that I can see (and debug) what is actually compiled (instead of only seeing the Annotation).

#### ***Example 2: Merge Java and SQL***

SQL enhanced code is pretty similar to bound properties but more code is generated. The first step is to define the class which maps a database table to a Java object:

```
@Entity
class Foo
{
    @Id
    @Column (type:java.sql.Types.INTEGER)
    int id
    @Column (type:java.sql.Types.CHAR,
size:20)
    String name
}
```

After this is compiled, I want to see a special field "SQL" which I can use to build database queries like so:

```
def columns = [Foo.SQL.value, Foo.SQL.name]
def cond = Sql.WHERE () { Foo.SQL.id >= 5 &&
  Foo.SQL.name != null }
def list = Sql.SELECT (columns,
  table:Foo.SQL.TABLE, where:cond,
  class:Foo.class)
```

This gets converted by the compiler into:

```
def list = []
def _sql = "SELECT id, name FROM foo WHERE
id >= ? AND name IS NOT NULL"
_sql = Sql.eachRow (_sql, [5]) {
  Foo o = new Foo ()
  o.id = it[0]
  o.name = it[1]
  list << o
}
```

The SQL object in Foo also gives access to the standard DAO methods like loading an object by its primary key:

```
def foo = Foo.SQL.load (5)
```

In addition to the simple bound property example, the AMP must also be able to note the usage of an annotated object, so it can convert the Groovy code into SQL at compile time (and possibly check it for mistakes).

## Open Issues

### *Java 1.4/5*

Groovy 1.x must run on Java 1.4. We must decide what to do with non-macro annotations, whether we want to support a switch to generate Java 5 classfiles (so Groovy can generate code for third party APTs like Hibernate)

It seems that it is possible to write annotations into Java 1.4 classfiles (see [Commons Attributes](#)). But the questions is: Is this futile? There are only a few tools which support annotations **and** Java 1.4.

In this light, it makes more sense to add a switch to allow Groovy to write Java 5 classfiles, so users stuck to 1.4 can still use it and Java 5 users can upgrade when they want to.

### *Expand or Pass On*

The compiler needs a way to decide what to do with an official Java 5 annotation like `javax.persistence.Entity` which is defined in EJB3: Expand it as a macro or pass it on into the class file so a third party library/tool can process it later.

Here, the user might want to decide differently per class (i.e. handle most of these cases with Hibernate and some corner cases with her own AST macro).

For Groovy-specific macros, the solution is to add a marker interface to the macro annotation.

### Options

- Add a config file to the compiler
- Users have to use a different annotation which implements both Groovy's marker interface and `javax.persistence.Entity`

### Links

- [Java 6 Annotations](#)
- [Getting Started With APT \(Java 6\)](#)
- [Annotation processing in Core Java \(7th Edition\)](#)
- [OOPSLA paper on which Annotations are based](#)

## C Sharp Groovinator

One aspect of Groovy ( and Ruby, Python , etc ) I really like is the combination of closures and support for them in the Groovy DK .

As an example, consider iterating through every line in a text file :

### *Groovy*

```
file = new File("c:/dynamic.ini");
file.eachLine{
    println(it)
}
```

Very elegant and easy on the old eyes.

*Java*

```
BufferedReader input = null;
try {
    input = new BufferedReader (new
FileReader("c:/dynamic.ini"));
    String line;
    while ((line = input.readLine()) !=
null) {
        System.out.println(line);
    }
} finally {
    if(input != null){
        input.close();
    }
}
```

Oh dear.

*Java with some help*

```
Iteration.eachLine("c:/dynamic.ini", new
LineHandler() {
    public void handleLine(String line) {
        System.out.println(line);
    }
});
```

and the supporting code :

```
public static void eachLine(String path,
LineHandler handler) throws IOException {
    FileReader file = null;
    try {
        file = new FileReader(path); // Open the
file.
        BufferedReader input = new
BufferedReader(file);
        String line;
        while ((line = input.readLine()) != null)
        {
            handler.handleLine(line);
        }
    } finally {
        if (file != null) {
            file.close();
        }
    }
}
```

Apologies for my C#, I have not been writing it for very long...

```
string line;  
using (StreamReader reader =  
File.OpenText(@"c:\dynamic.ini")) {  
    while((line = reader.ReadLine()) !=  
null){  
        Console.WriteLine(line);  
    }  
}
```

No need to escape path string is nice

#### *C# 2.0 with some help*

With C# 2.0 and the 2.0 .NET framework the addition of anonymous delegates we can produce a very concise and readable solution

```
Iterators.EachLine(@"c:\dynamic.ini",  
delegate(string it) {  
    Console.WriteLine(it);  
});
```

similar in elegance to the Groovy one.

This depends on the following utility class

```

public static class Iterators {

    public delegate void Closure<T>(T it);

    public static void EachLine(string
path, Closure<string> closure) {
        string line = null;
        using (StreamReader reader =
File.OpenText(path)) {
            line = reader.ReadLine();
            while (line != null) {
                closure(line);
                line =
reader.ReadLine();
            }
        }
    }
}

```

### *Groovinating C#*

You will notice that I have started implementing the Groovy DK set of collection iterators (each, findAll, find , etc ) for C# 2.0.

I do wish Microsoft and Sun would have built this kind of simplicity in....

There are 2 possible approaches to the Collections :

1. Subclass and add the grooviness
2. Add the grooviness as a set of static methods you pass the collection to

Here is some preliminary code and tests of both types.

This subclasses List<T> and adds the groovination, and offers the same support in a static class

### *C# 2.0 Code*



```
#region Using directives

using System;
using System.Collections.Generic;
using System.IO;

#endregion

namespace Groovinator {

    #region Deleगतros
    public class Deleगतros {
        public delegate void Closure<T>(T
it);
        public delegate T Functor<T>(T it);
        public delegate bool Predicate<T>(T
it);
    }

    #endregion

    #region Collection Iterators

    public class List<T> :
System.Collections.Generic.List<T> {
        public void
Each(Deleगतros.Closure<T> closure) {
            foreach (T it in this) {
                closure(it);
            }
        }
    }

}
```

```

    public List<T>
    FindAll(Delegatros.Predicate<T> predicate) {
        //this is a bit poor
        List<T> result = new List<T>();
        foreach (T it in this) {
            if (predicate(it)) {
                result.Add(it);
            }
        }
        return result;
    }

```

```

    public T
    Find(Delegatros.Predicate<T> predicate) {
        foreach (T it in this) {
            if (predicate(it)) {
                return it;
            }
        }
        return default(T);
    }

```

```

    public void
    Map(Delegatros.Functor<T> functor) {
        List<T> result = new List<T>();
        for(int i = 0;i<
this.Count;++i){
            this[i] = functor(this[i]);
        }
    }
}

```

```

public static class Iterators {

    // Collections
    public static void
Each<T>(IEnumerable<T> enumerable,
Delegatros.Closure<T> closure) {
        foreach (T it in enumerable) {
            closure(it);
        }
    }

    public static List<T>
FindAll<T>(IEnumerable<T> enumerable,
Delegatros.Predicate<T> predicate) {
        //this is a bit poor
        List<T> result = new List<T>();
        foreach (T it in enumerable) {
            if (predicate(it)) {
                result.Add(it);
            }
        }
        return result;
    }

    public static T
Find<T>(IEnumerable<T> enumerable,
Delegatros.Predicate<T> predicate) {
        foreach (T it in enumerable) {
            if (predicate(it)) {

```

```

        return it;
    }
}
return default(T);
}

    public static List<T>
Map<T>(IEnumerable<T> enumerable,
Delegatros.Functor<T> functor) {
    List<T> result = new List<T>();
    foreach (T it in enumerable) {
        result.Add(functor(it));
    }

    return result;
}

}

#endregion
#region File Iterators
public static class IOIterators{
    // File related iterators
    public static void EachLine(string
path, Delegatros.Closure<string> closure) {
        string line = null;
        using (StreamReader reader =
File.OpenText(path)) {
            line = reader.ReadLine();
            while (line != null) {

```

```
closure(line);  
line =  
reader.ReadLine();  
    }  
    }  
}
```

```
    }  
    #endregion  
}
```

#### *Some Usage Examples*

```
// open, print all lines of a text file ,  
close  
IOIterators.EachLine(@"c:\dynamic.ini",  
delegate(string it) {  
    Console.WriteLine(it);  
});  
  
List<string> list = new List<string>();  
  
// populate list  
  
// print a list of string  
list.Each(delegate(string it) {  
    Console.WriteLine(it);  
});  
  
System.Collections.Generic.IEnumerable<string> res;  
  
// find all items matching the predicate  
item.Contains("8")  
res = list.FindAll(delegate(string it) {  
    return it.Contains("8");  
});
```

```
// find the first item where  
Item.Equals("10")  
string ten = list.Find(delegate(string it) {  
    return it.Equals("10");  
});
```

```
List<int> nums = new List<int>();  
for (int j = 1; j < 11; ++j) {  
    nums.Add(j);  
}
```

```
// apply a functor ( multiply by 2 ) to a  
list of ints
```

```
nums.Map(delegate(int num) {  
    return num * 2;  
}));
```

## GJIT - Groovy Just-In-Time Compiler

### Background

Groovy has a performance problem, compared with Java, due to its dynamic nature, while there are several requests for performance improvement.

### Proposed Solution

Implementing just-in-time compiler is a proper way to help boosting performance of Groovy. It would be great to have a Groovy module (in this case a JVMTI agent) to help transform/redefine Groovy classes during runtime.

#### *Primitive Operations*

As of Groovy 1.6-beta series, its performance has been improved but there are still problems occurred from primitive operations. All operand and results are heavily boxed and unboxed when primitive variables are used. The current implementation of GJIT provides the algorithm to unwrap almost binary operations back to be primitive. Thus, it can speed Groovy performance up when dealing with arithmetic calculation.

#### *Implementation*

This is the completely re-written version of GJIT. Basically, GJIT can be thought as a Hotspot-like compiler for Groovy. However it is working outside JVM, and not a part of the virtual machine. GJIT uses JVMTI, the virtual machine tool interface, to:

1. Transform a class when it is being loaded, based on the assumption that the default metaclass provides the default behaviour.
2. Redefine the class with the original one, when the metaclass of this class is modified.

Due to the use of JVMTI, GJIT supports Java 5 or later.

### Usage

Add

```
-javaagent:gjit.jar
```

to your Java command line to use GJIT.

### Missing Features and Known Bugs

- Primitive casts have not been processed properly.
- Class redefinition is not built in yet, it needs to tweak MetaClassImpl.

## Grapes and grab()

### Abstract

Grape (The Groovy Adaptable Packaging Engine or Groovy Advanced Packaging Engine) is the infrastructure enabling the `grab()` calls in Groovy, a set of classes leveraging [Ivy](#) to allow for a repository driven module system for Groovy. This allows a developer to write a script with an essentially arbitrary library requirement, and ship just the script. Grape will, at runtime, download as needed and link the named libraries and all dependencies forming a transitive closure when the script is run from existing repositories such as Ibiblio, Codehaus, and java.net.

### TimeFrame

The prototype is ready for inclusion into the trunk immediately, however the principal dependent library used ([Apache Ivy](#)) is a SVN snapshot. We should at least wait for a Beta 2 release and preferably for an RC release of the library before merging into trunk. That makes it a possible 1.6 candidate, and more likely a 1.7 candidate.

### General Requirements

- There must be a means to cause all calls to the Grape system to become no-ops and rely strictly on the JVM classpath.
- Grape should support bringing in packages from Maven 2 repositories.
- There should be some notion of 'endorsed modules' that are shortcutted in for easier use. For example: Scriptom, GroovyWS, SwingXBuilder. These can be hosted as regular Maven2 POMS or some other repository mechanism can be used.
- There should be command line tools similar to RubyGems, to add, remove, enumerate, and update versions of Grapes stored in the local cache.
- The implementation should not prevent movement into a container style module system such as integrating OSGi or Java 7 Super Packages. (this requirement should not be construed as an endorsement of either)

### Specification

#### *Module versioning model.*

Grape follows the Ivy conventions for module version identification, with naming change.

- `group` - Which module group the module comes from. Translates directly to a Maven groupId or an Ivy Organization. Any group matching `/groovy[x][\.\.*]^/` is reserved and may have special meaning to the groovy endorsed modules.
- `module` - The name of the module to load. Translated directly to a Maven artifactId or an Ivy artifact.
- `version` - The version of the module to use. Either a literal version '1.1-RC3' or an Ivy Range '[2.2.1,)' meaning 2.2.1 or any greater version).

#### *Code Level Support*

##### Annotation

One or more `groovy.lang.Grab` annotations can be added at any place that annotations are accepted to tell the compiler that this code relies on the specific library. This will have the effect of adding the library to the classloader of the groovy compiler. This annotation is detected and evaluated before any other resolution of classes in the script, so imported classes can be properly resolved by a `@Grab` annotation.

```

import com.jidesoft.swing.JideSplitButton
@Grab(group='com.jidesoft',
module='jide-oss', version='[2.2.1,)')
public class TestClassAnnotation {
    public static String testMethod () {
        return JideSplitButton.class.name
    }
}

```

An appropriate `grab(...)` call will be added to the static initializer of the class of the containing class (or script class in the case of an annotated script element).

*TODO: How do we add a grab annotation to straight scripts? The current workaround is to annotate something in the script that can accept an annotation, like a method or parameter.*

#### Method call

Typically a call to `grab` will occur early in the script or in class initialization. This is to insure that the libraries are made available to the `ClassLoader` before the groovy code relies on the code. A couple of typical calls may appear as follows:

```

// random maven library
groovy.grape.Grape.grab(group:'com.jidesoft'
, module:'jide-oss', version:'[2.2.0,)')
groovy.grape.Grape.grab([group:'org.apache.i
vy', module:'ivy', version:'2.0.0-beta1',
conf:['default', 'optional']],
    [group:'org.apache.ant', module:'ant',
version:'1.7.0'])

// endorsed Groovy Module
// FUTURE grab('Scriptom')

```

\* ~~`grab(Object self, String module)`~~, `grab(Object self, Map attrs)`, and `grab(Object self, Map attrs, Map... dependencies)` will be added to the DGM so that references to the backing Grape

classes will not be needed. These will be proxies to the main Grape class calls.

- Multiple calls to `grab` in the same context with the same parameters should be idempotent. However, if the same code is called with a different `ClassLoader` context then resolution may be re-run.
- `grab` is disabled by default. Starting calling `Grape.initGrape()` will enable `grab`. Any calls to `grab` before `initGrape()` is called will be ignored. Hence Grape managed classloading is opt in only.
- If the `args` map passed into the `grab` call has an attribute `noExceptions` that evaluates true no exceptions will be thrown.
- `grab` requires that a `RootLoader` or `GroovyClassLoader` be specified or be in the `ClassLoader` chain of the calling class. By default failure to have such a `ClassLoader` available will result in module resolution and an exception being thrown (if `initGrape()` has been called).
  - The `ClassLoader` passed in via the `classLoader:` argument and its parent classloaders.
  - The `ClassLoader` of the object passed in as the `referenceObject:` argument, and its parent classloaders.
  - The `ClassLoader` of the class issuing the call to `grab`
- ~~`grab(String)` is a shortcut for endorsed groovy modules and will be equivalent to `grab(group:'groovy-  
endorsed', module:<the module>, version:<the version of groovy being run>)`~~

*TODO: all discussion of `grab(String)` are purely hypothetical as it hasn't been prototyped or the needed infrastructure set up*

#### **`grab(HashMap)` Parameters**

- `group:` - `<String>` - Which module group the module comes from. Translates directly to a Maven groupId. Any group matching `/groovy(|\..*|x\..)` is reserved and may have special meaning to the groovy endorsed modules.
- `module:` - `<String>` - The name of the module to load. Translated directly to a Maven artifactId.
- `version:` - `<String>` and possibly `<Range>` - The version of the module to use. Either a literal version '1.1-RC3' or an Ivy Range '[2.2.1,)' meaning 2.2.1 or any greater version).
- `conf:` - `<String>`, default 'default' - The configuration or scope of the module to download. The default `conf` is `default:` which maps to the maven `runtime` and `master` scopes.
- `force:` - `<boolean>`, defaults true - Used to indicate that this revision must be used in case of conflicts, independently of
- `conflicts manager`
- `changing:` - `<boolean>`, default false - Whether the artifact can change without its version designation changing.
- `transitive:` - `<boolean>`, default true - Whether to resolve other dependencies this module has or not.

There are two principal variants of `grab`, one with a single `Map` and one with an arguments `Map` and multiple dependencies map. A call to the single map `grab` is the same as calling `grab` with the same map passed in twice, so `grab` arguments and dependencies can be mixed in the same map, and `grab` can be called as a single method with named parameters.

There are synonyms for these parameters. Submitting more than one is a runtime exception.

- `group:`, `groupId:`, `organisation:`, `organization:`, `org:`
- `module:`, `artifactId:`, `artifact:`
- `version:`, `revision:`, `rev:`
- `conf:`, `scope:`, `{{configuration:}}`

*Grape may want support simple ranges ('2.2.1..'\*) or '2.2.1'..'<3.0.0' meaning any version greater than 2.2.1 but no 3.x version. But there are implementation problems, trying either of these ranges creates a runtime error having to do with enumerating the list.*

## Arguments Map arguments

- `classLoader`: - `<GroovyClassLaoder>` or `<RootClassLoader>` - The `ClassLoader` to add resolved Jars to
- `refObject`: - `<Object>` - The closest parent `ClassLoader` for the object's class will be treated as though it were passed in as `classLoader`:
- `validate`: - `<boolean>`, default `false` - Should poms or ivy files be validated (`true`), or should we trust the cache (`false`).
- `noExceptions`: - `<boolean>`, default `false` - If `ClassLoader` resolution or repository querying fails, should we throw an exception (`false`) or fail silently (`true`).

### Tool level support

#### GroovyStarter

`GroovyStarter` has been enhanced with an additional `--grab` option. All three main arguments are required (group, module, version). Resolved jars will be added to the `RootLoader`. These modules will be resolved prior to the main class call.

#### LoaderConfiguration conf files

`LoaderConfiguration` has been enhanced with an additional 'grab' line option, where the option is followed by all three arguments (group, module, version). Resolved jars will be added as though they were specifically mentioned by a 'load' line option. These lines are similarly subject to property replacement as 'load' lines are.

#### <groovyc> support

The `GroovyC` ant task has been enhanced with `<grab>` child elements. The attributes `group`, `module`, and `version` specify the module to grab.

| *TODO: currently this doesn't work when fork='true'*

### File Level View

The downloaded modules will be stored according to Ivy's standard mechanism with a cache root of `~/ .groovy/grape`

### Command Line Tools

```
grape install <groupId> <artifactId>  
[<version>]
```

This installs the specified groovy module or maven artifact. If a version is specified that specific version will be installed, otherwise the most recent version will be used (as if `**` we passed in).

```
grape list
```

Lists locally installed modules (with their full maven name in the case of groovy modules) and versions.

```
grape resolve (<groupId> <artifactId>  
<version>)+
```

This returns the file locations of the jars representing the artifacts for the specified module(s) and the respective transitive dependencies.

## Motivation

The motivation for introducing Grape into groovy is to resolve the library and module proliferation problem we are encountering. As a general purpose language Grape is suitable for a lot of wildly variable tasks. Providing groovy ways to do those things can create 'bloat' because a large number of classes that are

## Rational

Credit for coming up with the name Grape goes to Jörg Staudemeyer and grab Andres Almiray.

### *Use of Maven Repositories*

Maven is big, there are tons of Java libraries that are already deployed in a well-known infrastructure. Because almost all  of these are straight Java they are all usable in Groovy out of the box.

### *Use of Ivy*

- Ivy is licensed under the ASL2, which is the same license as Groovy
- Ivy is focused strictly on Repository Management. Maven has a strong Repository Management portion, but also has other aspects not germane to the needs of the Grape system, namely build management and project management. If the goal is to provide a means to drop unneeded code we shouldn't bring in unneeded code.
- Ivy provides for an extensible repository resolution system. If we decide to roll our own Groovy Repository we have the infrastructure to do it, we can also support other non-maven repository systems if needed.

## Backward Compatibility

GroovyStarter does do some existing library management, namely loading every single jar in `~/groovy/lib` into the RootLoader. With Grape installed such behavior can continue, and is merely duplicate effort with no adverse implications. In the 2.0 timeframe we may want to look into deprecating or removing the auto-include of the `~/groovy/lib` directory.

## Reference Implementaion

[A 0.1.0 prototype is available](#). This shows the `grab(...)` method, the `@Grab` annotation, and command line tools. Examples are available in subversion in [src/examples](#), except for the annotation, where the best examples are in the [test class](#).

This version may conflict with Gant, only one of the Ivy files should exist under `$/GROOVY_HOME/libs`. If the beta-1 jar is left in place then Grape may not work.

## Future Directions

We may want to increase the kind of repositories that Grape can handle from simple Ivy and Maven repositories to other areas such as OSGi Repository Bundles, JNLP (WebStart) files and JSR-277 JAM files.

## Groovy realtime archive internet lookup

What is Grail?

Groovy Realtime Archive Internet Lookup == GRAIL

This "grail" ideas are not related to the [Grails web application framework](#).

What does it do?

Check IRC log below

The idea is to have something like RubyGems or CPAN for Groovy automating the resolution of dependencies on external jars, and downloading them automatically 😊, the end of jar hell.

A list of potential features

- Dynamic class loading without using import statements...
- Maven repository support, with local/mirrors support,
- virtual maven repos (a repository that does not exist, but is presented as such by an index that can point to any jar, i.e. for Sourceforge based files, or other non ibiblio based jars)
- Support for version resolution, scoping
- Jar integrity verification
- the base for doing it: <http://www.pmilne.net/launcher/>

Going forward

After a chat with Phil Milne, the author of Launcher.java, we could use his code as a base. He is also in! The project could be called Jail (Java Archive Internet Lookup) instead of Grail, because it would be for any java environments, not only Groovy.

The license would be a simple MIT style license.

Maybe hosted at Codehaus?

Grail Genesis : the Groovy 1.0 Beta 7 [Release Process](#) IRC logs...

08:36 <tug> Jstrachan: can you put the embeddable jar on the ftp download area, once built.

08:37 <jstrachan> happy

08:37 <Guillaume> we should also distribute this artifact as well

08:38 <jstrachan> you mean in the dist.codehaus.org/groovy/jars area?

08:40 <Guillaume> I guess so along the groovy.jar so that ppl can just choose one single jar with everything included

..

08:43 <Guillaume> perhaps a sub-project, like xml-rpc I guess (for the xml-rpc.jar) with reactor?

08:43 <jstrachan> good idea

08:43 <jstrachan> hmm, no the subproject only gets copied into the binary distro not as a separate jar AFAIK

...

08:44 <tug> After build it creates target/embeddable/groovy-all-xxx.jar

08:44 <tug> Include that jar file on it's own, in the ftp area, for ease of fetching...

...

08:44 <pombreda> Guillaume: instead of having one big jar, what about something like maven/ibiblio integration? something like RubyGems <http://rubygems.rubyforge.org/wiki/wiki.pl> or CPAN?

08:45 <jstrachan> we've got the repo, maven, we just need to use in a clever way

08:45 <jstrachan> I friend of mine has done a cool class loader to do this... <http://www.pmilne.net/launcher/>

08:45 <Guillaume> pombreda, the big jar includes the asm jars into one big groovy+asm

08:46 <tug> Jez says: Can we put the bloglines example in the groovy src examples dir...

08:46 <Guillaume> tug-jez, good idea!

...

08:46 <Guillaume> we'd need to add the http-client jar to the dependencies

08:47 <jstrachan> lets do that next release - its been chugging for 30 mins already

...

08:46 <pombreda> Guillaume: but for stuffs like xml-rpc getting based on Maven... integrated in Groovy... would be way cool.

08:48 <jstrachan> it'd be cool to just type : x = new com.foo.something.Whatever()

08:48 <jstrachan> and for it to just work, downloading whatever it needs

08:48 <pombreda> Yehaa! happy)) that would rock!

...

08:48 <Guillaume> what if there are two com.foo.something.Whatever ? with three different releases?

08:49 <jstrachan> you setup rules for versions

08:49 <Guillaume> yeah, nobody reads the reports... useless

08:49 <jstrachan> e.g. use latest production relases or for 'foo' use version '1.2.3' latest production release is fine for most people plus we should include metadata in the manifest of the versions we depend on so dependent jars should match the release of each project

08:51 <Guillaume> and if groovy requires a specific version, but the script you're running would require another one?

08:51 <jstrachan> things could certainly get nasty

08:51 <pombreda> I would rather get a ganifest than an ugly xml manifest?

08:51 <jstrachan> but you could do a nice simple

08:52 <jstrachan> versionScope

## Unknown macro: { // do something with version 1 of foo }

08:52 <jstrachan> versionScope

## Unknown macro: { // do something with version 2 }

08:52 <jstrachan> etc

08:52 <jstrachan> versionScope

## Unknown macro: { it.setMinimumRequirement("foo", 2.0); .... }

08:52 <jstrachan> etc

08:52 <Guillaume> hmmm... with a specific classloader for the scoped stuff?

08:52 <jstrachan> yep

08:52 <Guillaume> in its own thread,

08:53 <jstrachan> threads don't matter, its the classloader that does

08:53 <Guillaume> interesting idea

08:53 <jstrachan> all the Launcher is, above, is a classloader that auto-downloads stuff it needs

08:53 <jstrachan> would be trivial to add to groovy

08:53 <jstrachan> **expecially** for use in IDEs / consoles

08:53 <jstrachan> where you just wanna script something quick

08:53 <jstrachan> and not spend most of the time setting the goddamned classpath

08:54 <jstrachan> I'm sure it'd be easy to make a versionScope() thingy introspectable, so you could enquire the versions your using etc

08:54 <pombreda> and include some auto Md5 verif

...

08:56 <pombreda> jstrachan, Guillaume: this idea rocks... the end of jar hell unhappy(

08:59 <pombreda> what about a name for the version/jar loader? GRAIL

08:59 <pombreda> Groovy Realtime Archive Internet Lookup == GRAIL  
08:59 <precipice> ha!  
08:59 <Guillaume> pombreda, that sounds like the French verb "grailer" happy  
09:00 <jstrachan> happy  
09:00 <jstrachan> or just 'groovy' happy  
09:00 <pombreda> Guillaume, it would eat jars, would it? "grailer"== eating in slang  
09:00 <brianm> WOOT  
09:01 <brianm> you talking the go-get-jar-from-maven at runtime thing?  
09:01 <jstrachan> yes  
09:01 <jstrachan> happy  
09:01 <brianm> GRAIL is perfect  
09:01 <jstrachan> I'm sold!  
09:01 <brianm> heh, better than "go get" use a remote classloader  
09:01 <Guillaume> the holy grail cheesy  
09:01 <jstrachan> happy  
09:01 <Guillaume> who's going to implement taht?  
09:01 <brianm> Sun already did  
09:01 <Guillaume> wasn't there someone supposed to have been working on it?  
09:02 <jstrachan> it shouldn't take long to wire the launcher into the shell / console etc  
09:02 <brianm> parsing out maven paths will be tricky =)  
09:02 <jstrachan> it does this already  
09:02 <brianm> the package -> project mapping is doable though  
09:02 <jstrachan> its already done, check the link above  
...  
09:05 <pombreda> about GRAIL: some issue: ibiblio/maven is so unreliable... it is down every now and then ...  
09:06 <jstrachan> mirrors  
09:06 <brianm> should check local maven repo first  
09:06 <jstrachan> maven supports mirroring of repos, or have your own local mirror of projects you're interest in etc  
09:06 <pombreda> good point.  
09:08 <pombreda> that could call for a project.groovy rather than a project.xml...  
...  
09:44 <pombreda> brian: about GRAIL, you said earlier : Sun already did. could you elaborate?  
09:44 <brianm> the remote classloader  
09:44 <brianm> all we need to do is figure out url for jars  
09:44 <jstrachan> the Launcher implementation trawls the maven repo to build up indices AFAIK  
09:47 <pombreda> but does'nt remote class loader runs has security constraints?  
09:53 <pombreda> briaim: AFAIK: remote class loader can only originate jars from the same code base as the app.  
09:53 <pombreda> Check <http://java.sun.com/j2se/1.4.2/docs/guide/extensions/spec.html>  
...  
19:52 <pombreda>Guillaume: about GRAIL, I will start a page in the Groovy space, and post an edited IRC log. OK?  
19:53 <Guillaume>okey dokey  
19:53 <Guillaume>you want to work on that?  
19:54 <pombreda>yep. why not? we call it GRAIL or Grail?  
19:54 <Guillaume>not too many caps 😊  
19:54 <Guillaume>you're welcome to work on anything you like 😊  
19:54 <pombreda>So? which one?  
19:56 <oni> what is GRAIL?  
19:58 <pombreda>oni: Groovy Realtime Archive Internet Lookup == GRAIL  
19:59 <oni> pombreda and what does it do?  
20:00 <pombreda>oni: check today's IRC log at : <http://servlet.uwyn.com/drone/log/hausbot/groovy>  
20:00 <pombreda>the idea is to have something like RubyGems or CPAN for Groovy  
20:01 <pombreda>automating the resolution of dependencies on external jars 😊, the end of jar hell

## JSR-295 Binding Builder

**This proposal is deferred and not under active development or consideration**

The reason for deferral is that the JSR-295 API is too unstable. Since this posting it has gone under a major re-design. It will not be revisited any earlier than the posting of the first proposed final draft for that reason.

Based off of an example in Scott Violet's [last blog](#) at sun, a BindingBuilder in practice would look something like this:

```
// code from elsewhere...
public class Bug {
    @BoundProperty String ID
    @BoundProperty int priority
    @BoundProperty String synopsis
    @BoundProperty String type
}

List<Bug> bugs = [...] // assuming generics
make it into 1.1

bugTable = theSwingBuilder.bugTable
summaryLabel = theSwingBuilder.summaryLabel
idTF = theSwingBuilder.idTF
descriptionTF =
theSwingBuilder.descriptionTF
typeTF = theSwingBuilder.typeTF
prioritySlider =
theSwingBuilder.prioritySlider

// the actual bindings
bindings = new BeansBindingBuilder()
context = bindings.context() {
    binding(source: bugs, target: bugTable,
property: 'elements') {
        binding(value: '${ID}') {
```

```

        tableColumn(0)
    }
    binding(value: '${priority}') {
        tableColumn(1)
    }
    binding(value: '${synopsis}') {
        tableColumn(2)
    }
}
binding(
    source:bugTable,
    value: '${bb:listSize(selectedElements)}
of ${bb:listSize(elements)} are selected',
    target:summaryLabel,
    property:'text')
    binding(source:bugTable,
value: '${selectedElements.ID}', target:idTF,
property:'text') {
    textChangeStrategy(CHANGE_ON_TYPE)
    concatenatingCondensor(['"', "'", ','])
}
    binding(source:bugTable,
value: '${selectedElements.synopsis}',
target:descriptionTF, property:'text') {
    textChangeStrategy(CHANGE_ON_TYPE)
    concatenatingCondensor(['"', "'", ','])
}
    binding(source:bugTable,
value: '${selectedElements.type}',
target:typeTF, property:'text') {
    textChangeStrategy(CHANGE_ON_TYPE)

```

```
concatenatingCondensor(['"', '"', ','])  
}  
binding(source:bugTable,  
value:'${selectedElements.priority}',  
target:prioritySlider, property:'value')
```

```
}
```

```
context.bind()
```

Once I get a better feel for the existing code I can mock up what all the possible children would be.

Here's what the java snippets from the post are

```
context = new BindingContext();
```

```
List<Bug> bugs = ...;
```

```
Binding tableBinding = context.addBinding(
```

```
    bugs,          // Source for the binding,
```

```
the List of bugs in this case.
```

```
    null,          // Expression, relative to  
the source, used in obtaining the property.
```

```
        // For this example it's  
null, meaning use bugs as is.
```

```
    bugTable,     // Target of the binding, a  
JTable in this case.
```

```
    "elements"); // The property of the  
target to bind to.
```

```
tableBinding.addBinding(
```

```
    "${ID}",      // Expression evaluated  
relative to each Bug.
```

```
        // In this case, it's  
treated as bug.getID().
```

```
    null,          // Target value (I'm not  
going to get into this parameter now)
```

```
TableColumnParameter, 0); // Specifies
the binding applies to the first column
```

```
tableBinding.addBinding("${priority}", null,
TableColumnParameter, 1);
tableBinding.addBinding("${synopsis}", null,
TableColumnParameter, 2);
```

```
Binding textFieldBinding =
context.addBinding(
    bugTable, // Source of
the binding, the JTable in this case.
    "${selectedElements.ID}", // Expression
relative to the source. Evaluates to the
// the id
property of each of the selected elements
    idTF, // Target of
the binding, a JTextField here.
    "text", // The target
property to bind to.
    // The next line specifies the 'text'
property should change as you type.
    // The default is to change the property
on enter/focus leaving.
    TextChangeStrategyParameter,
TextChangeStrategy.CHANGE_ON_TYPE);

textFieldBinding.setListCondenser(ListConden
ser.concatenatingCondenser(
    "\\\"", // The string placed before each
element
```

```
    "\\\"",    // The string paced after each
element
    ", ")); // The string that separates
each element.
```

```
context.addBinding(
    bugTable, // The source of the binding,
the table in this case.
    // The expression evaluated relative to
the source. Notice this makes use
    // of the function "listSize", that
returns an size of the list supplied to it.
    "${bb:listSize(selectedElements)} of
${bb:listSize(elements)} are selected",
    summaryLabel, // The target of the
binding, a JLabel here.
    "text");      // The target property to
```

```
bind to
```

```
context.bind();
```

## Macro Use Cases

Use this page to list use cases for AST-level macros in Groovy.

How do you want to use such macros? Show examples of their usage (without showing how they're implemented). Add to the bottom of the page.

### *Increase code readability*

#### *Rearrange code portions*

The end-weight principle for natural languages says the longer stuff should be at the end of a sentence. In Groovy/Java, while and if statements are usually heavy in the block/statement, and light in the condition. When the statement/block is short and the condition long, it can be more readable to put the block last. Eg:

```
do{ i++; return c }if(  
    ( severance.taxAmount >=  
TaxYear2006.taxAmounts.severanceLimit[  
    'dependent' ] ||  
    ... ..  
    ... ..  
)
```

is transformed to:

```
if( ... .. ) { i++; return c }
```

We could use 'unless' and 'until' if using 'if' and 'while' caused problems.

### *Common modifier*

When a long list of methods and/or fields have the same modifier/s, it may be more readable to apply them to a whole block, eg:

```
static{  
  def a(){ ... }  
  def b(){ ... }  
}
```

becomes

```
static a(){ ... }  
static b(){ ... }
```

*Transform statements*

Eg:

```
def genUniqueScriptName(){ static int i=1;  
  'Script_' + i++ }
```

becomes

```
class UniqueScriptName{ static int i=1;  
  static gen(){ 'Script_' + i++ }
```

triggered by the 'static' modifier of a field in a function.

*Names for operators*

Use names instead of symbols for operators, eg:

```
if( a and b )
```

becomes

```
if(a && b)
```

or change their names to something shorter, eg:

```
if( a of A )
```

becomes

```
if( a instanceof A )
```

#### *Equivalence of block and statement*

Forgoing curlies for all statements, making a statement and block equivalent always:

Eg:

```
try A.call()  
catch(e) if(e instanceof ABCException) throw  
new BusinessException()
```

#### *Use other natural languages with Groovy*

Enable fuller internationalization of Groovy, eg:

```
mientras( i < 10 ){ ... .. }
```

would convert to

```
while( i < 10 ){ ... .. }
```

if a Spanish option for Groovy was loaded.

#### *More Seamless Integration with Java and Java-like languages*

Embed Java code within Groovy without quoting it, and without needing to bind variables to the same name, ie, no `binding.setVariable('i', i)`.  
eg:

```
int i= 0  
java('jdk1.5.0_07'){ System.out.println i;  
class ... .. }
```

### ***Simplify the Antlr lexer/parser***

Many existing syntactic sugars could be re-implemented as macros in a standard macro library. This might simplify the Antlr lexer/parser, enabling better maintenance of and extensions to it.

Eg, properties where:

```
String reading= 'OK'
```

expands to:

```
private String reading= 'OK'  
public String getReading(){ this.xxxx }  
public void setReading( String s ){  
this.xxxx= s }
```

Other examples:

- `for( i in ... )` expanding to code using iterator
- shorthand notation for invoking method with closure/s at last parameter/s

### **MetaClass Redesign (by blackdrag)**

This page is dedicated to the new Design of the MetaClass.

#### ***General Problem***

It seems that we need 2 parts for MetaClass, one for the implementation and one for the user to customize the behavior. The reason for this is, that the customization is done by subclassing the implementation part. This means it is impossible to stay in the specification part (classes from `groovy.*`) to do a all day thing. Additionally the method signatures must be fixed to not to break code.

#### ***Current Behavior***

Currently we do something like:

```
class MyMetaClass extends MetaClassXY {  
    public Object invokeMethod(Object o,  
String name, Object[] args) {  
        preProcessing();  
        super.invokeMethod(o, name, args);  
        postProcessing();  
    }  
}
```

- preProcessing might change the arguments, name or call object, or might just do some logging.
- postProcessing might evaluate the return value of the method call, call additional methods, or just do logging
- every part here might end the invokeMethod method, that means super.invokeMethod might not be called

This is very flexible, it allows interception and logging as well as transformation at the same time.

### ***Basic Design***

If we want to separate the two cases (runtime MetaClass, user defined MetaClass) in two classes, then one must call the other. As the user part should stay in the specification, it can't know the exact class of the runtime, or which method to call there. This means the user part must be called from the MetaClass in the runtime. Since we are not able to call "super" then we need an additional Object, that we make the call on:

### **Call Chain**

```
class MyMetaClass extends MetaClassXY {  
    public Object invokeMethod(Object o,  
String name, Object[] args, CallChain chain)  
    {  
        preProcessing();  
        chain.invoke(o, name, args);  
        postProcessing();  
    }  
}
```

```

    }
}

interface CallChain {
    Object invoke(Object base, String name,
Object[] args)
}

class RuntimeMethodClass {
    ...
    Object invokeMethod(Object base, String
name, Object[] args, ....) {
        process(args)
        MetaClass mc = getMetaClass()
        if (mc!=null) {
            return mv.invokeMethod(base,name,args,
new CallChain() {
                Object invoke(Object base, String
name, Object[] args) {
                    return
doRealInvoke(base,name,args...)
                }
            }
        } else {
            return doRealInvoke(base,name,args...)
        }
    }
}

```

```
    ...  
}
```

That's a bit like the continuation passing style.

*Disadvantages*

- A custom MetaClass results in creating a ChainCall object each time an method of that class is invoked.

*Advantages*

- The old behavior can be preserved and guranteed

**Multiple Behaviors controlled by Class**

If we want to split the behavior, then we could have an interface for each of the cases

```
interface MetaClassInterceptor {  
    // throws MissingMethodException if no  
    interception is done, which results in  
    normal method invocation  
    Object invokeMethod(Object base, String  
name, Object[] args);  
}  
  
interface MetaClassLogger {  
    // throws never MissingMethodException  
    and does not change the arguments  
    void invokeMethodEntry(Object base,  
String name, Object[] args);  
    void invokeMethodExit(Object base, String  
name, Object[] args);  
}  
  
interface MetaClassTransformer {  
    // basically the same as above
```

```
    Object invokeMethod(Object base, String
name, Object[] args, CallChain chain);
}
```

```
class RuntimeMetaClass {
    ...
    Object invokeMethod(Object base, String
name, Object[] args, ....) {
    process(args)
    MetaClass mc = getMetaClass()
    if (mc!=null) {
        Object result = null;
        MetaClassLogger logger=null
        MetaClassInterceptor interceptor=null
        MetaClassTransformer transformer=null
        if (mc instanceof MetaClassLogger)
logger = (MetaClassLogger ) mc
        if (mc instanceof MetaClassInterceptor)
interceptor = (MetaClassInterceptor) mc
        if (mc instanceof MetaClassTransformer
) interceptor = (MetaClassTransformer ) mc
        try {
            if (logger!=null)
logger.invokeMethodEntry(base,name,args)
            if (transformer!=null) {

transformer.invokeMethod(base,name,args, new
CallChain() {
                Object invoke(Object base, String
name, Object[] args) {
                    return
```

```
invokeWithInterceptor(interceptor,
base,name,args...)
    })
    } else if (interceptor!=null) {
        return
invokeWithInterceptor(interceptor,
base,name,args...)
    }
    } finally {
        if (logger!=null)
logger.invokeMethodExit(base,name,args)
    }
    } else {
        return doRealInvoke(base,name,args...)
    }
}
```

```
    . . .  
}
```

Maybe it is a bit overkill, since a transformer can act as interceptor, but a logger is surely more lightweight than an interceptor or transformer.

#### **Disadvantages**

- the common case still requires a ChainCall object each time a method of that class is invoked

#### **Advantages**

- the logging case does not influence the invocation so much

#### **General Disadvantage**

The general disadvantage of the separation is that we need to associate the custom MetaClass and the runtime MetaClass. Again I see two ways:

1. let the runtime get the custom MetaClass:

```
class RuntimeMetaClass {  
    Class theClass;  
    MetaClass getMetaClass() {  
        return registry.getMetaClass(theClass)  
    }  
    ...  
}
```

A null entry in the registry would then mean, that no custom MetaClass is set, in case of GroovyObject, the logic must be changed a little because null here means just no per instance MetaClass. As there is only one runtime MetaClass and several behaviors this means that it is not possible to do

```
metaClass.getProperties()
```

or something alike. Again we could add an interface and make the runtime MetaClass ask the custom MetaClass or to provide an additional set of Properties and Methods at initialization, but then we get another set of interfaces and the runtime MetaClass must keep track of the changes in case a new custom MetaClass is set.

2. let the runtime assign the runtime MetaClass to the custom MetaClass:

```

class MyCustomMetaClass extends MetaClass {
    MyCustomMetaClass(MetaClass rmc, MetaClassRegistry mcr){
        super(rmc,mcr)
    }
    ...
}

interface MetaClass {
    MetaClass getChainedMetaClass()
    List getProperties()
    ...
}

class MetaClass {
    ...
    List getProperties(){return runtimeMetaClass.getProperties()}
    MetaClass getInnerMetaClass() {return runtimeMetaClass}
}

```

So this works much like our ProxyMetaClass. The behavior is much like the old MetaClass, just the signature of invokeMethod for example changes from (Object,String,Object[]) to (Object,String,Object[],ChainCall). It is then no longer possible to invoke a method using invokeMethod directly, it is then only for internal use. Which is no problem, as long as we keep an alternative way for Java as we currently have with InvokerHelper. Multiple chained MetaClasses are also possible this way, the implementation must just keep that in mind and iterate through all of them when making a call. This again is very unlike the ProxyMetaClass we currently have. I would strongly suggest to make MetaClass no interface then and to remove ProxyMetaClass, because the MetaClass then fulfills already the requirements of ProxyMetaClass. Another solution would be to not to let it act as proxy, but to require an additional method:

```

class MetaClass {
    QueryableMetaClass
    getQueryableMetaClass() {
        MetaClass mc = this
        while (!(mc instanceof
    QueryableMetaClass)) {
            mc = mc.getChainedMetaClass()
        }
        return mc;
    }
    ...
}

```

Where getChainedMetaClass behaves like above, meaning, it returns the next MetaClass, the last MetaClass is then the runtime implementation. This version means the usage is

```

metaClass.queryableMetaClass.getProperties()

```

instead of the old

```

metaClass.getProperties()

```

but it means also, that we can keep the MetaClass interface/class very clean and don't need to add other methods but the methods required for chaining, which we could even make final. Any additional behavior is then controlled by the interfaces I have shown above. This way we could make an MetaClass that intercepts methods, but not properties or fields. Or a version that does logging only on Properties, but does not interfere with normal method invocation. I am aware that this means to have many interface.. 9 then (Interceptor + Transformer + Logger for each of Property, Field and Method) or 6 (if Transformer and Interceptor are unified)

1. The custom behavior is unable to tell the difference between a call to super, a call from inside the class, or a

normal call from a context outside (could be solved with additional different interfaces)

#### General Advantage

- implementation and custom behavior are separated
- we can split the runtime MetaClass in a mutable and immutable part and reconstruct the immutable part on demand
- custom MetaClasses becomes more lightweight, since no full initialization step is required for each custom MetaClass and the heavy part can be shared by multiple custom MetaClasses

#### *More efficient custom MetaClasses*

Additionally to the traditional Design we could also have a new Design, where we don't take part in the CallChain, but produce a MetaMethod (or property).

```
interface MetaClass2 {  
    MetaMethod getMetaMethod(Object base,  
String name, Object[] args)  
}  
  
class MyMetaClass extends MetaClass  
implements MetaClass2 {  
    MetaMethod getMetaMethod(Object base,  
String name, Object[] args) {  
        if (name.equals("foo")) return  
myFooMetaMethod()  
        return null  
    }  
}
```

which means this method is called in case of a cache miss, and not every time. Returning a MetaMethod we can control if we would like to add the method to the cache or not through the isCachable() method on MetaMethod. We could also change the interface a little and force the call to getMetaMethod even if there is no cache miss. The advantage is, we can still intercept methods, but in a more efficient way, because we don't need to throw a MissingMethodException. And we can use the cache directly if we do not force method selection through this class each time, which makes it very fast.

#### *Proposal*

So my proposal is to chain the MetaClasses as shown above, to have a more or less empty MetaClass class which

does only the chaining and to have interfaces controlling all aspects of the method invocation (or for properties/fields) like I have shown with logger and transformer or MetaClass2. Having `metaClass.queryableMetaClass.getProperties()` means to have less methods on MetaClass and since these MetaClasses do normally not add real methods. It would make sense to do so, but `metaClass.getProperties()` is not really a problem.

### ***Variations***

- Instead of chaining the MetaClasses like above, we could also have a central MetaClass and just add behavior using interface like the ones I shown (transformer, interceptor...). This would mean to change GroovyObject, because `get/setMetaClass` becomes meaningless then. If we use a proxy like mechanism, then we end in something that looks almost as the one above, but even more complicated.
- I tried to find a way to keep the call to super, but I was unable to find out how without using a ThreadLocal workaround, which I would like to avoid.

## **MetaClass Redesign (by blackdrag) part 2**

In [MetaClass Redesign \(by blackdrag\)](#) I tried to show some ideas of how such a redesign might look like and what the advantages and disadvantages might be.

### ***groovy.lang.MetaClass***

In this part I want to show a more detailed version of one idea. that is a implementation based MetaClass is in front and the user can add custom behaviors. This limits the basic MetaClass interface to mere administrative methods

```

package groovy.lang;

public interface MetaClass {
    // MOP
    void addMOP(MetaClassObjectProtokoll mop);
    MetaClassObjectProtokoll
removeMOP(MetaClassObjectProtokoll mop);

    // add methods and properties
    void addMethod(MetaMethod mm);
    void addProperty(MetaProperty mp);
    void addConstructor(MetaMethod mc);

    // listener for MetaClass changes
    void addMetClassListener(MetaClassListener
mcl);
    void
removeMetaClassListener(MetaClassListener
mcl);
    MetaClassListener[]
getMetClassListeners();
}

```

So the central point is addMOP, which allows to customize the method invocation for example. Besides this there are some methods for mutating the MetaClass (adding properties and such), a standard listener interface that allows to react to additions of properties for example. The implementation of this is not relevant for the user as it is thought as specification. The implementation depends on the Groovy runtime and might differ from other runtimes if there are ever such runtimes.

#### *Usage of groovy.lang.MetaClass*

If the user wants to influence the method invocation process for example, then he has to implement the MetaClassObjectProtokoll interface, which I will show later. In this section I want to show how to create a per instance MetaClass and how to use it.

*changing a MetaClass in the registry*

```
//old
coderegistry.setMetaClass(theClass,myMetaClass)
// new code
metaClass =
registry.getMetaClass(theClass).add
Behavior(myMOP)
// or in one line
registry.getMetaClass(theClass).addMOP(myMOP)
)
```

So the one-line code becomes a more complex two-line code, unless we decide to change MetaClassRegistry to make that by itself.

*creating and changing a per instance MetaClass*

The old code was like this:

```
groovyObject.metaClass = myMetaClass
```

The new code will look like this:

```
groovyObject.metaClass.addMOP(myMOP)
```

So the change here is minimal. The code to create a per instance MetaClass is hidden by the implementation and might differ. This of course means that the class GroovyObjectSupport, which is part of groovy.lang has implementation knowledge inside. So a implementator must change this class to his version of the runtime.

*Using the Listener to MetaClass changes*

(add interface)

...

*Consequences*

MOP 2.0 ideas

## A small collection of ideas for MOP 2

### *ideas by Jochen:*

- we should no longer directly invoke methods like equals, setProperty and such.(see GROOVY-2635 for example)
- we should think about changing GroovyObject into a pure marker interface (if setProperty is needed in the logic we have problems)
- the MetaClass should be able to overwrite all methods including setProperty (this implies that we do not directly call setProperty)
- Comparable needs a rethought. This interface is not well to be used if you have to handle objects of different types
- it would be nice to be able to define only one operand like < or > instead of having to define both. At last being able to define these additionally
- we have to test if creating direct method call classes in call sites is doable. This would be like Reflector, but only for a single method, allowing the JIT to inline the method without much hazzle. currently reflection means to have a border where the JIT can not really inline the call. Maybe we should also have different modes here, to allow users to still use reflection if this version has a problem on certain VMs or in certain scenarios
- as a design goal I would like to transform the MetaClass from a directly method invoking to a request oriented method returning class. This would be especially interesting for call site caching. The user could still write his invokeMethod method, but instead of calling that method directly in the Metaclass we would let the MetaClass return a method wrapper that can do the call to invokeMethod.
- I would also like to add a Listener to each MetaClass which allows other parts of the program to be informed when a MetaClass is changed in any kind. This could probably also be used to invalidate call site caches.
- A version information added to the MetaClass allows a very fast check for changes. We could also think about adding an additional version cache for method names. the idea here is to have a simple array and to let a call site cache the index to the array, which means the index is equal to a complete method signature or only to name. By checking the information in there(for example an referential identity check with the cached method) would allow the call site to quickly know if it needs to update or not. Such an array would grow other time if it needs to reflect dynamically added methods as well. So we should do this maybe only for the default methods.
- another design goal should be that if a code modifies a Metaclass, then as much changes as possible should be done by the thread modifying the metaclass. that means that this thread will for example call all the listeners, probably update caches, reorganize structures.. what ever
- The allow call sites faster access without going through the registry it would be good to have some kind of handle that can be used to directly access the MetaClass. This means that either the real MetaClass must not be replaced (only on the implementation side of course) or that the handle is update if a replacement happens.
- I would also like to add a module concept based on different meta class registries. For this a class using a module will contain some kind of information that will cause all code directly in this class to use only the meta classes from the registry and probably no per instance meta class. As some kind of namespace is required to differentiate one module from another we could couple this with the package "package foo with module" or by using a package like reference to a constant "module Foo.myRegistry" (Foo is a normal class, myRegistry is a final  public static field of type MetaClassRegistry or an alibi type. Annotations seem not to be fitted for this, as they would have to use enums, classes, primitives or strings and none but strings seem to fit that. enums won't work as I would have to define all possibilities in the runtime not allowing the user to define his own ideas of how it should work. Using classes would work, but where is the connection here to make it clear what we mean? primitives lack memorization, the user would have not only to remember a number, he would also have to make sure that it is not used somewhere else too. Strings are better, but there might still a name conflict. An Annotation would be doable if we want to make the module per class, but I am not sure this makes much sense.
- we should also look if we can not reduce the amount of MOP methods. We need one for each non public method, true, but we possibly do not need all these super\$ methods
- as Java still does direct calls we should think about adding the ability to methods to check if their MetaClass

has a new version for this method. This way Java could easily access the meta class stored method. Of course we then have to also find a way to access the original method as well, as this method may be needed

- we should also think if we should add mechanisms we add for method calls should also be added for property access.
- we need to look at method caching in builders.... or maybe this is not needed as a builder might not be executed often
- for the meta class implementation we should also think about having special method selectors like I have in ClosureMetaClass.
- if we do call site caching, then we should probably remove the method caching inside the MetaClass, especially if we add specialized method selectors
- a point of discussion is if a method added to the parent classes metaclass should also affect the current metaclass or not.
- we need also to test if rewriting the method itself at runtime is possible and what it will cost.
- we need to investigate if inlining primitive type operations at compile time is possible. This can help avoiding boxing and direct usage of primitive types. Of course before doing something like that we would have to check the Metaclasses and all operations if this is allowed. If we do the idea with the method signature arrays we could also define constants for certain operations like +, -, \*, / to do the check very fast. If I do for example 1+1 I could compile this as: "if (Integer.metaclass.version==0 && Integer.metaclass.ops\_1==0) {iload 1; iload 1; iadd;}" 0 would show it is the default version and 1 is a numeric constant to see if the operation + is still the default or not.
- maybe the module concept can be used to enable calling private methods/fields. With this we could disable access to private from outside the class per default.. of course only after we solved the Closure problem (information loss with getProperty)
- invokeMethod should loose its double role for being a method missing and at the same time a direct invocation way from Java

## Multiple Assignment Proposal

### Superceded

Note that an initial implementation of multiple assignment is already built in to Groovy 1.6 and above. This document is kept for historical purposes and because it contains suggestions for advanced features not yet implemented. See [Multiple Assignment](#) for more details of the current implementation.

### Introduction

This page looks at proposed changes to the Groovy language to support multiple assignment. Such support would allow for code such as:

```
a, b = b, a    // will transparently handle  
any temporary storage
```

```
list = [1, 2, 3]  
def (c, d, e) = list  
assert c == 1 && d == 2 && e == 3
```

It is meant to be a discussion document, not a formal definition, we will work out the impact on the grammar etc. once we clarify our intention.

#### Single Assignment

Some existing declarations involving single assignment:

```
def a = 'a' // for examples below
def b = 'b' // for examples below
def c = 'c' // for examples below

//----- Existing: Java style, simple case
(Proposal leaves unchanged)
def x1,y1,z1 = a // only z1 is equal to a,
the others are undefined

assert x1 == null
assert y1 == null
assert z1 == a

//----- Existing: Java style, list case
(Proposal leaves unchanged)
def x2, y2, z2 = [a,b,c] // only z2 is equal
to [a,b,c], the others are undefined

assert x2 == null
assert y2 == null
assert z2 == [a,b,c]

//----- Existing: Java style, multiple
equals case (Proposal leaves unchanged)
def x3 = a, y3 = b, z3 = c // obvious
assert x6 == a
```

```
assert y3 == b
```

```
assert z3 == c
```

```
//----- Existing: Java Java style,  
multiple lists case (Proposal leaves  
unchanged)
```

```
def x4 = [a,b], y4 = [b,c], z4 = [a,c] //  
obvious
```

```
assert x4 == [a,b]
assert y4 == [b,c]
assert z4 == [a,c]
```

#### Variable declarations

In order to not conflict with existing definitions, any variable declaration which wishes to make use of multiple assignments must surround the 'tuple' of variable declarations in round brackets. Whenever such a round bracket is found, the compiler will match the tuple with a list. The list will be expected to be where it would be in the case of single assignment. The compiler will 'unpack' the list into the tuple of variables.

```
def (x5,y5,z5) = [a,b,c] // common
type goes before round brackets
def (x6,y6) = [a,b], z6 = c //
x,y,z/5,6,7 must all be undefined
def list = [b, c]
def x7 = a, (y7, z7) = list // list can
be explicit or implicit

assert x5 == a && x6 == a && x7 == a
assert y5 == b && y6 == b && y7 == b
assert z5 == c && z6 == c && z7 == c
```

Nested variations are also supported (this may be deferred initially):

```
def ((x8,y8),z8) = [[a,b],c]
assert x8 == a && y8 == b && z8 == c
```

Differing sized tuples and lists are catered for by ignoring extra list members and leaving any untargeted variables in the tuple set to null:

```
def (x9,y9,z9) = [a,b]
assert x9 == a && y9 == b && z9 == null

def (x10,y10,z10) = [a,b,c,d] // d unused
assert x10 == a && y10 == b && z10 == c
```

There can be at most one [at the end like varargs?] 'tuple spread' operator which takes the rest of the list:

```
def (first11,*rest11) = [a,b,c,d]

assert first11 == a
assert rest11 == [b, c, d]
```

Setting a bunch of variables to the same (or related) values is accomplished using normal list conventions:

```
def (x12,y12,z12) = [a] * 3
assert [x12, y12, z12] == [a, a, a]

def (x13,y13,z13) = 0..2
assert [x13, y13, z13] == [0, 1, 2]

def string = 'abcdefghi'
def (x14,y14,z14) = (1..3).collect{
string[0..it*2]}
assert [x14, y14, z14] = ["abc", "abcde",
"abcdefg"]
```

The tuple list is strictly a list (apart from the nesting seen earlier) and may not contain assignments itself:

```
// INVALID  
def (x15 = c, y15, z15) = [a,b,c]
```

Example involving nulls:

```
def (x16,y16,z16)  
assert x16 == null  
assert y16 == null  
assert z16 == null
```

Examples involving real world use:

```
def (name, address1, address2, postcode) =  
myData.tokenize("\n").toList()  
// acts like first/head/top  
def (name) = myData.tokenize("\n").toList()  
// equivalent to above but 'self  
documenting'  
def (name, *ignored) =  
myData.tokenize("\n").toList()
```

#### Use in Expressions

The left hand side (LHS) of an expression involving the simple assignment operator, =, can be replaced with a tuple. In this case, the variables will be predefined or binding variables and the result of the RHS of the expression will be a (possibly coerced) list which will be unpacked into the tuple.

```
def list = [a, b]
def x17, y17
(x17, y17) = list
assert [x17, y17] == [a, b]
```

Where it is not ambiguous, the round brackets can be removed from the tuple list when used in such expressions, so the above could be written as:

```
x17, y17 = list // LHS turned into (x17,
y17)
```

If the right hand side after any assignment operator contains a comma separated list of subexpressions, they will be assumed to make up a list (i.e. automatically turned into a list), so the above could be written as:

```
x17, y17 = a, b // RHS turned into [a, b]
```

[Note: the above probably requires lots of pouring over the grammar!]

If you find you need a tuple with different typed items inside, use an expression form rather than the variable declaration form which requires the types to be the same, e.g.:

```
def myMonth(){ [2000, "Jan", 1] }
String month
int day, year
(year, month, day) = myMonth()
```

Unlike with declarations, the LHS doesn't need to simply be a variable:

```
def map = [a:1, b:2]
(map.a, map.b) = [3, 4]
assert map == [a:3, b:4]

a = 10..15
i = 3
i, a[i] = i+1, 20
assert i == 4 && a[3] == 20 && a[4] == 14
```

Other examples which need to work:

```
x = 0
a, b, c = x, (x += 1), (x += 1)
assert [a, b, c] == [0, 1, 2]

year, month, day = "2007-03-20".split('-')
```

Other assignment operators

These may not be implemented initially, but we attempt to define them here. It includes these:

```
*= /= %= += -= <<= >>= >>>= &= ^= |=
```

First, a tuple on the LHS of a special assignment operator:

```

a, b *= 3
// form list [a, b] (tuple spread operator
// ignored here if present)
// call multiply() operator with arg 3
// unpack result into (a, b) tuple (spread
// operator applies to unpacking)

// example
def x18 = a, y18 = b
x18, *y18 *= 3
assert x18 = a
assert y18 = [b, a, b, a, b]

```

Now, an example with special treatment on the RHS:

```

yourObject *= 1, 2
// calls the multiply() method on yourObject
// with the list [1, 2]

```

As an alternative to the semantics described here, we could define 'a, b \*= 3' to be the same as 'a \*= 3; b \*= 3'. This seems more useful but then if the semantics were to be consistent, then 'a, b = 3' should mean 'a = 3; b = 3'.

#### Use in Method declarations

No change, no grouping of method parameters is supported, i.e. the following is invalid:

```

// INVALID
def myMethod((a, b), c) { ... }

```

See Also

<http://jira.codehaus.org/browse/GROOVY-158>

## Multiple Assignment

### Multiple Assignment

Groovy supports multiple assignment, i.e. where multiple variables can be assigned at once, e.g.:

```
def (a, b, c) = [10, 20, 'foo']
assert a == 10 && b == 20 && c == 'foo'
```

You can provide types as part of the declaration if you wish, e.g.:

```
def (int i, String j) = [10, 'foo']
```

As well as used when declaring variables (as above) it also applies to existing variables, e.g.:

```
def nums = [1, 3, 5]
def a, b, c
(a, b, c) = nums
assert a == 1 && b == 3 && c == 5
```

The syntax works for arrays as well as lists, as well as methods that return either of these, e.g.:

```
def (_, month, year) = "18th June
2009".split()
println "In $month of $year" // => In June
of 2009
```

## Overflow and Underflow

If the left hand side has too many variables, excess ones are filled with null's, e.g.:

```
def (a, b, c) = [1, 2]
assert a == 1 && b == 2 && c == null
```

If the right hand side has too many variables, the extra ones are ignored, e.g.:

```
def (a, b) = [1, 2, 3]
assert a == 1 && b == 2
```

Notes:

- currently only simple variables may be the target of multiple assignment expressions, e.g. if you have a `pers` on class with `firstname` and `lastname` fields, you can't currently do this:

```
(p.firstname, p.lastname) = "My name".split()
```

- currently multiple assignment cannot be used for declaring multiple class fields and initializing them as:

```
class Foo {
    def (i,j) = [1,2]
}
```

## Multiproject builds

I think the biggest challenge for our build system is how to implement multiproject builds. It is likely that most of users will do multiproject builds.

### Vision

- Gant enables DRY for multiproject builds. This is were Ant has massively failed so far.
- Gant offers an intuitive way to write a multiproject build.
- Gant offers out of the box functionality for standard Java build tasks, which will grow from release to release.
- Gant offers maximum flexibility for customization. This is were Maven has massively failed.

### Directory structure for multiproject builds

I think it is good to be very flexible regarding the possible directory structures. For Steven's use case it was good enough to assume that all projects are toplevel folders of the root folder. But there are many situations were this structure is not expressive enough and therefore should not be imposed.

In my last project we had a structure like:

- shared
- client
- ...

- services
  - services-shared
  - service1
  - ...
  - serviceN

And I'm sure there are much more complicated project structures out there which can't be changed or people don't want to change them. I propose to have only one assumption. There is a root folder where we (might) have a top level buildfile and all projects folder must be contained somewhere in this root folder.

#### **Determining the projects to be build**

Even if I do not consider pathological cases, automatically determining the projects to be build is not easy or at least not cheap. The rule would be that everything is a project which contains a buildfile. If we look at the projects layout described in the section above (nested projects), we would have to scan many, many folders to look for the buildfiles (e.g. `src/main/java/org/codehaus/.....`). We could try to be smart and have the policy not to look in folders with name `src`. But the name `src` is just a convention and could be changed by the user. We could read the configuration of a found project first and then continue our search. But there might be some folder in a project with many files and subfolder which just sits there (e.g. not in use any longer but people don't want to delete it). For every build that is triggered we would have to scan this folder to look for buildfiles. I think an easy way out of this problem is that people have to declare the projects to be build in a top level build file. This has also the advantage to have a static file containing the projects layout.

#### **Implementation**

I've been thinking a lot about this in the last days. I think I have a solution I'm happy with now. It is very much inspired from looking at the code of `buildr`.

#### **Endresult**

After processing all the build scripts, our new Gant will produce one classic Gant build script (in memory). So the complete multiproject build is finally described by exactly one flat list of targets which are related by dependency relations.

#### **How to get there**

Every subproject is represented by an instance of a `Project` object. This `Project` object is amongst other things a container for target object. For each `Project` object we create a set of default target instances (e.g. `compile`, `test`, ...) which we add to the `Project` object. Every `Project` must have a unique name (`parent-project-name + project-name`). The target instances of a `Project` object have the name: `project-name+target-name`.

#### **Customization of the default targets**

The default targets are not simple target objects. I think about creating for example a class `Compile` which extends the class `Target`. That means the `compile` target is a target but also a special `compile` object, that offers methods for customization and of course the action for actually doing the compile.

In our build script we could write for example `target("compile").with classpath`. Or even better `compile.with classpath`.

#### **Adding new behavior**

For this feature I guess we have to extend the classic Gant a little bit. We might make it a bit more Rakish. Basically I have two additional methods for a target in mind:

- `target("compile").dependsOn(otherTarget1, otherTarget2, ...)`

- `target("compile").addAction { ... }`

To add behavior we can now either create a new target in our build script and make a default target dependent on it. Or we can simply add behavior to an existing target, which gets executed after the default behavior is executed.

When I talk about the build script I have in mind that every subproject has its build script. That means the behavior changes described above only apply to the build of this particular subproject. If we want a more general change, we have a couple of possibilities. In the build script of a parent project we can say:

- something like `this.getChildren().each{add behavior}`
- `addRecursiveTarget(target)`

#### Interproject dependencies

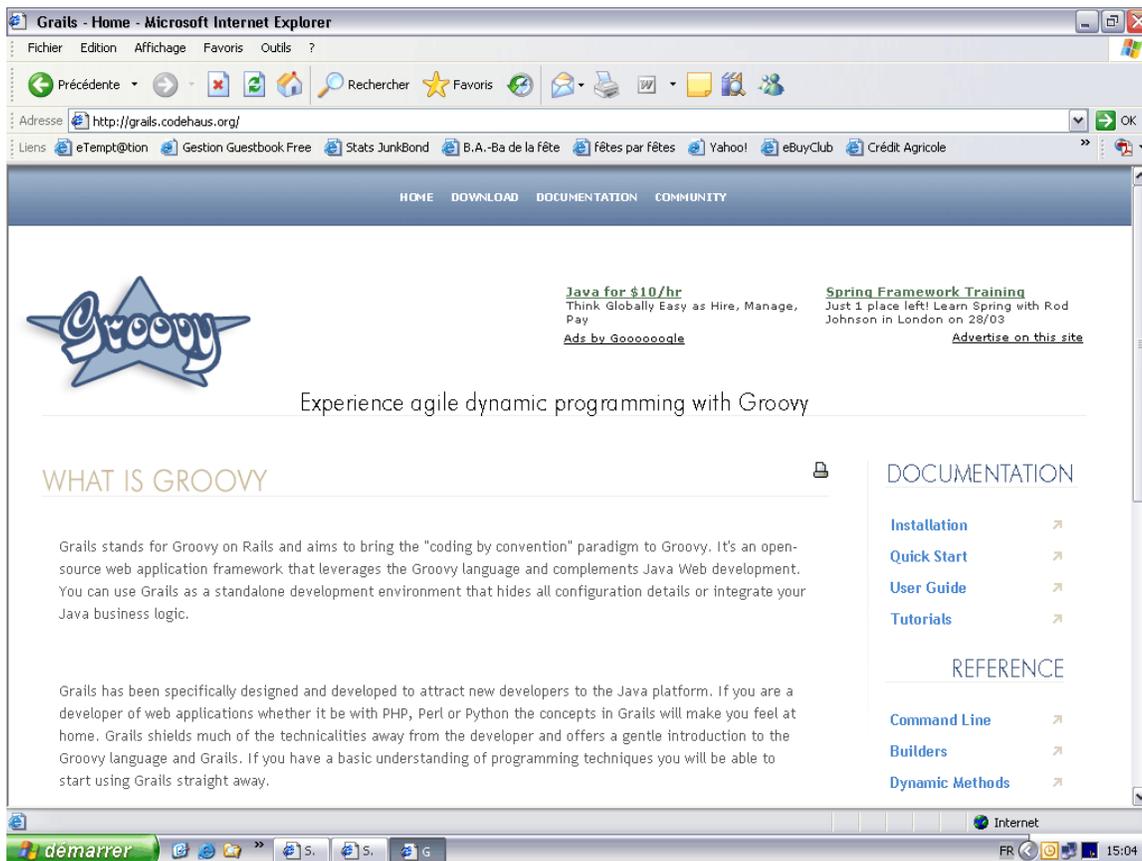
I think we don't have to explicitly declare them. If there is such a dependency, it means somewhere in the build script of the project that depends on the other project, it refers to this other project anyway. For example `compile.with project("otherProject").classpath`.

#### Conclusions

Of course there are many details to hammer out. But I'm confident enough now to start implementing something based on the above ideas if we all think this makes sense.

## Proposed Website Revamp

### *New web page mockup*



### *New logos in "grails" spirit*



**Archives**



1.



2.



3.



4.



5.



6.



7.



8.



9.



10.



11.



12.



13.



14.

### **Web based GroovyShell**

Hi All,

This is my first post at Groovy forum. Last couple of days I am learning groovy. I like this language a lot. Previously I had explored BeanShell. BeanShell provides online Web based BeanShell Servlet for executing beanshell code online. I find same for Groovy, but I am not able to find it. So I had written very primitive Web Based Groovy Shell (essentially Groovy Server Page with Groove Shell capabilities). Might be it is useful to others also like me. Mainly I am using all these web based scripting shell for debugging web based application. Typically changing code, session, request variables from it and debugging the application.

[Web based Groovy Shell:](#)

## Miscellaneous Feature Requests and Enhancements

Legacy enhancements and feature requests which haven't yet been accepted for the current version of Groovy or earmarked for a known future version of Groovy. Further progress will require additional discussion or ideally creation of a GEP.

- GROOVY-54 Add generator syntax similar to Python's
- GROOVY-1077 operator overloading

## Groovy 2.0 modularization

### Rationale

Groovy has become a pretty complete package with more than just the language itself, but also with several useful APIs and wrappers to simplify the life of developers, dealing with Ant, Swing, XML, JDBC, command-line, and more. The main groovy-all JAR has reached over 4MB, preventing, for example, usage in an applet, as the time to load the JAR is too long. Recognizing that nobody needs everything in Groovy, it is time we rework the Groovy source organization, deliverables, project build, and more. However, this is a "big bang" kind of change that needs a good level of discipline to get through properly, as several key steps will have to be made.

### Key steps

Having a modular system means that:

- **sources will have to be reorganized:** this is where a DVCS system will be especially useful
- **modules will have to be created:** as source code is being reorganized, modules will be created, but will need to be built using a multi-module approach

A good approach will be to follow some baby steps:

- providing a new build which will support multiple modules in the future
- migrating to a DVCS system to ease the migration
- progressively migrate one module at a time, to avoid any breakage

### *Build System*

The major options for modular build systems are:

- gradle
- mvn (+gmaven)

A consensus is emerging that Gradle should be our new build infrastructure. A Gradle replacement for the current Ant build will be created over the next few weeks. This can then be evolved into a multi-project build once the module structure is decided.

We'll also need to ensure that our toolset will properly support a Gradle build infrastructure:

- IDEs: IntelliJ IDEA supports Gradle out of the box, for instance
- CI: continuous integration servers should be able to build a Gradle-powered build of Groovy

### *Source reorganization and repository*

**Move to a DVCS system: Git**

To help with external contributions, and also to ease the work of source code reorganization, it'll be interesting to

move to a different source code repository, moving from Subversion to a distributed source control system. Git is the choice made for DVCS at Codehaus, so we'll migrate to Git as part of the process.

Here again, we need to ensure that tool support is okay with Git:

- IDEs: IntelliJ IDEA offers a great support of Git, the command-line is also usable, of course. Eclipse has some support for Git.
- FishEye: FishEye is pretty handy for navigating in the source's history. We need to check that the issues of FishEye support for Git are fixed.
- CI: the continuous integration servers should be able to checkout the source code from Git. Codehaus' Bamboo already has support for Git.

#### **Base structure**

The current SVN structure does not really follow standards. For example, groovy-core, a separately release component (at the moment, hope to change), is located at: <http://svn.codehaus.org/groovy/trunk/groovy/groovy-core/> This should really be something like:

The modules listed above would be listed under: <http://svn.codehaus.org/groovy/core/trunk/<module>>

And they would be release together. Completely optional projects, like the native support, ide support, etc. should be located at: <http://svn.codehaus.org/groovy/<project>/trunk>

Such a structure will ensure the core modules follow the same lifecycle as groovy-core, whereas external modules can still be evolved independently, and don't need to be released alongside Groovy releases.

There should perhaps be three levels of module:

1. Core
2. Non-core, centrally supported
3. Others

Core modules are those needed for the core set of functionality to work. Whether these are shipped as a single jar or a set of jars is up for debate. Everything in this category should be held in the central Git master repository.

Non-core, centrally supported should be in separate Git repositories, not in the Groovy master. They should though be managed from Codehaus in tight-knit collaboration with the master core repository.

Others are modules held by whom and wherever they are held. These might be managed using Bazaar, Mercurial, or Git – there is no need to impose Git as a tool unless they are to migrate into one of the two above centrally managed in which case they must migrate to use Git.

There should be a central index of core modules, non-core centrally managed modules and of any other modules people care to register with the index.

#### ***Other considerations***

##### **OSGi**

Since we've made Groovy's deliverables be proper OSGi bundles, we need to pursue our efforts, and make sure the core modules are also valid OSGi bundles, also marking dependencies between core modules, and external dependencies.

##### **DefaultGroovyMethods registration**

Some modules (swing, sql) also provide their own DefaultGroovyMethods. We'll need to figure out a way for these modules to register their own DGM methods, such as a META-INF/services technique.

##### **Capabilities support**

An old JIRA issue we may consider: <http://jira.codehaus.org/browse/GROOVY-2422>

The idea is to provide a "capabilities" mechanism to Groovy, to know which features are supported in the current Groovy version.

Inspired by this issue, we could at least provide some utility class (for instance in GroovySystem) where we could discover the existing modules available on the classpath.

## Modules

### *Proposed structure*

Here's a module structure we could follow. We shouldn't multiply the number of modules to a useless high number, but focus on core functionality, and ensure a pure core will be small enough to be easily embeddable and downloadable (for applets, mobile devices such as Android, etc.)

- groovy-core
- modules
  - test
  - jmx
  - swing
  - xml
  - sql
  - web
  - template
  - integration
    - bean-scripting-framework
    - javax.script
  - tools
    - groovydoc
    - shell
    - console
    - java2groovy

### *Issues to be aware of*

Certain classes would naturally belong to certain modules, but are sometimes used in core, etc.

A good example of this is GroovyTestCase and GroovyShellTestCase. These two classes would naturally go into the test module, but it means the core module would depend on test, but test would already depend on core. So, so far, in the following approach, I've kept these base classes (not very heavyweight anyway) in core, but we'll have to figure out a good way to migrate them in their respective module.

Another example is the java2groovy tool (not very much used, may perhaps be discarded at some point). It belongs to its own module, under some tools meta-module, but the trick with this tool is that it also contributes some batch scripts so one can run the tool from the command-line easily. For the normal distribution, anyway, we can embed those scripts, so that shouldn't be critical.

An important thing to consider is DefaultGroovyMethods. There are already some work done in that area to split methods related to SQL/JDBC and for Swing. So these additional DGM methods will naturally go in their respective modules (ie. sql and swing), but that means we need to provide a mechanism for discovering and registering such methods, for example with some META-INF/services discovery mechanism.

The modules proposed here are more or less coarse-grained modules, in order to keep things simple, and to try to regroup things by big functionality. But some things may be split furthermore. For instance, we could put CliBuilder

into its own class, and the same for AntBuilder (although those two are really just one class each!). It may be interesting to do so, also because these classes need additional dependencies (respectively commons-cli and ant). Speaking of splitting those builders into their respective modules, we could even regroup such builders into some meta-builder module.

The groupings below are done per functionality, so even if some utility class like groovy.util.GroovyMBean is in groovy.util, I moved it to the jmx module. Ultimately, it'd certainly be nicer to move such classes into their proper packages, following a deprecation strategy.

### ***More detailed regroupment of classes into each module***

I've created a little Groovy script to list / group all the Java and Groovy classes per package. Afterwards, I've moved the packages and some individual classes to form modules according to their respective feature set.

---

Module: groovy-core

## **Package: groovy.beans**

- Bindable.java
- BindableASTTransformation.java
- Vetoable.java
- VetoableASTTransformation.java

## **Package: groovy.grape**

- GrabAnnotationTransformation.java
- Grape.java
- GrapeEngine.java
- Grapelvy.groovy

## **Package: groovy.io**

- EncodingAwareBufferedWriter.java
- GroovyPrintStream.java
- GroovyPrintWriter.java
- PlatformLineWriter.java

## **Package: groovy.lang**

- AdaptingMetaClass.java
- BenchmarkInterceptor.java
- Binding.java
- Buildable.java
- Category.java
- Closure.java
- ClosureException.java
- ClosureInvokingMethod.java
- Delegate.java
- DelegatingMetaClass.java
- DeprecationException.java
- EmptyRange.java
- ExpandoMetaClass.java
- ExpandoMetaClassCreationHandle.java
- Grab.java
- GrabConfig.java
- GrabExclude.java

- GrabResolver.java
- Grapes.java
- GroovyClassLoader.java
- GroovyCodeSource.java
- GroovyInterceptable.java
- GroovyLogTestCase.groovy
- GroovyObject.java
- GroovyObjectSupport.java
- GroovyResourceLoader.java
- GroovyRuntimeException.java
- GroovyShell.java
- GroovySystem.java
- GString.java
- IllegalPropertyAccessException.java
- Immutable.java
- IncorrectClosureArgumentsException.java
- Interceptor.java
- IntRange.java
- Lazy.java
- MetaArrayLengthProperty.java
- MetaBeanProperty.java
- MetaClass.java
- MetaClassImpl.java
- MetaClassRegistry.java
- MetaClassRegistryChangeEvent.java
- MetaClassRegistryChangeListener.java
- MetaExpandoProperty.java
- MetaMethod.java
- MetaObjectProtocol.java
- MetaProperty.java
- MissingClassException.java
- MissingFieldException.java
- MissingMethodException.java
- MissingPropertyException.java
- Mixin.java
- MutableMetaClass.java
- Newify.java
- NonEmptySequence.java
- ObjectRange.java
- PackageScope.java
- ParameterArray.java
- PropertyAccessInterceptor.java
- PropertyValue.java
- ProxyMetaClass.java
- Range.java
- ReadOnlyPropertyException.java
- Reference.java
- Script.java
- Sequence.java
- Singleton.java
- SpreadListEvaluatingException.java
- SpreadMap.java
- SpreadMapEvaluatingException.java

- `StringWriterIOException.java`
- `TracingInterceptor.java`
- `Tuple.java`
- `Writable.java`

## **Package: groovy.security**

- `GroovyCodeSourcePermission.java`

## **Package: groovy.time**

- `BaseDuration.java`
- `DatumDependentDuration.java`
- `Duration.java`
- `TimeCategory.java`
- `TimeDatumDependentDuration.java`
- `TimeDuration.java`

## **Package: org.codehaus.groovy.ant**

- `AntProjectPropertiesDelegate.java`
- `CompileTaskSupport.java`
- `FileIterator.java`
- `FileScanner.java`
- `GenerateStubsTask.java`
- `Groovy.java`
- `Groovyc.java`
- `GroovycTask.java`
- `Groovydoc.java`
- `LoggingHelper.java`
- `RootLoaderRef.java`
- `UberCompileTask.java`
- `VerifyClass.java`

## **Package: org.codehaus.groovy.antlr**

- `AntlrASTProcessor.java`
- `AntlrASTProcessSnippets.java`
- `AntlrParserPlugin.java`
- `AntlrParserPluginFactory.java`
- `ASTParserException.java`
- `ASTRuntimeException.java`
- `EnumHelper.java`
- `GroovySourceAST.java`
- `GroovySourceToken.java`

## **Package: org.codehaus.groovy.antlr.parser**

- `GroovyLexer.java`
- `GroovyRecognizer.java`
- `GroovyTokenTypes.java`
- `SourceBuffer.java`
- `SourceInfo.java`

## **Package: org.codehaus.groovy.antlr.treewalker**

- `CompositeVisitor.java`
- `FlatNodeListTraversal.java`

- MindMapPrinter.java
- NodeAsHTMLPrinter.java
- NodeCollector.java
- NodePrinter.java
- PreOrderTraversal.java
- SourceCodeTraversal.java
- SourcePrinter.java
- TraversalHelper.java
- Visitor.java
- VisitorAdapter.java
- UnicodeEscapingReader.java

## **Package: org.codehaus.groovy.ast**

- AnnotatedNode.java
- AnnotationNode.java
- ASTNode.java

## **Package: org.codehaus.groovy.ast.builder**

- AstBuilder.groovy
- AstBuilderTransformation.groovy
- AstSpecificationCompiler.groovy
- AstStringCompiler.groovy
- ClassCodeExpressionTransformer.java
- ClassCodeVisitorSupport.java
- ClassHelper.java
- ClassNode.java
- CodeVisitorSupport.java
- CompileUnit.java
- ConstructorNode.java
- DynamicVariable.java

## **Package: org.codehaus.groovy.ast.expr**

- AnnotationConstantExpression.java
- ArgumentListExpression.java
- ArrayExpression.java
- AttributeExpression.java
- BinaryExpression.java
- BitwiseNegationExpression.java
- BooleanExpression.java
- CastExpression.java
- ClassExpression.java
- ClosureExpression.java
- ClosureListExpression.java
- ConstantExpression.java
- ConstructorCallExpression.java
- DeclarationExpression.java
- ElvisOperatorExpression.java
- EmptyExpression.java
- Expression.java
- ExpressionTransformer.java
- FieldExpression.java
- GStringExpression.java

- ListExpression.java
- MapEntryExpression.java
- MapExpression.java
- MethodCallExpression.java
- MethodPointerExpression.java
- NamedArgumentListExpression.java
- NotExpression.java
- PostfixExpression.java
- PrefixExpression.java
- PropertyExpression.java
- RangeExpression.java
- RegexExpression.java
- SpreadExpression.java
- SpreadMapExpression.java
- StaticMethodCallExpression.java
- TernaryExpression.java
- TupleExpression.java
- UnaryMinusExpression.java
- UnaryPlusExpression.java
- VariableExpression.java
- FieldNode.java
- GenericsType.java
- GroovyClassVisitor.java
- GroovyCodeVisitor.java
- ImportNode.java
- InnerClassNode.java
- InterfaceHelperClassNode.java
- MethodNode.java
- MixinASTTransformation.java
- MixinNode.java
- ModuleNode.java
- PackageNode.java
- Parameter.java
- PropertyNode.java

## **Package: org.codehaus.groovy.ast.stmt**

- AssertStatement.java
- BlockStatement.java
- BreakStatement.java
- CaseStatement.java
- CatchStatement.java
- ContinueStatement.java
- DoWhileStatement.java
- EmptyStatement.java
- ExpressionStatement.java
- ForStatement.java
- IfStatement.java
- ReturnStatement.java
- Statement.java
- SwitchStatement.java
- SynchronizedStatement.java
- ThrowStatement.java
- TryCatchStatement.java

- WhileStatement.java
- Variable.java
- VariableScope.java

## **Package: org.codehaus.groovy.binding**

- AbstractFullBinding.java
- AggregateBinding.java
- BindingProxy.java
- BindingUpdatable.java
- BindPath.java
- ClosureSourceBinding.java
- ClosureTriggerBinding.java
- EventTriggerBinding.java
- FullBinding.java
- MutualPropertyBinding.java
- PropertyBinding.java
- PropertyChangeProxyTargetBinding.java
- PropertyPathFullBinding.java
- SourceBinding.java
- SwingTimerTriggerBinding.java
- TargetBinding.java
- TriggerBinding.java

## **Package: org.codehaus.groovy.classgen**

- AnnotationVisitor.java
- AsmClassGenerator.java
- BytecodeExpression.java
- BytecodeHelper.java
- BytecodeInstruction.java
- BytecodeSequence.java
- ClassCompletionVerifier.java
- ClassGenerator.java
- ClassGeneratorException.java
- CompileStack.java
- DummyClassGenerator.java
- EnumVisitor.java
- ExtendedVerifier.java
- genArrayAccess.groovy
- genArrays.groovy
- genDgmMath.groovy
- GeneratorContext.java
- genMathModification.groovy
- GroovyCompilerVersion.java
- InnerClassVisitor.java
- MethodCaller.java
- MethodCallerMultiAdapter.java
- ReturnAdder.java
- RuntimeIncompleteClassException.java
- Variable.java
- VariableScopeVisitor.java
- Verifier.java
- VerifierCodeVisitor.java

## **Package: org.codehaus.groovy.control**

- CompilationFailedException.java
- CompilationUnit.java
- CompilePhase.java
- CompilerConfiguration.java
- ConfigurationException.java
- ErrorCollector.java
- GenericsVisitor.java
- HasCleanup.java

## **Package: org.codehaus.groovy.control.io**

- AbstractReaderSource.java
- FileReaderSource.java
- InputStreamReaderSource.java
- NullWriter.java
- ReaderSource.java
- StringReaderSource.java
- URLReaderSource.java
- Janitor.java
- LabelVerifier.java

## **Package: org.codehaus.groovy.control.messages**

- ExceptionMessage.java
- LocatedMessage.java
- Message.java
- SimpleMessage.java
- SyntaxErrorMessage.java
- WarningMessage.java
- MultipleCompilationErrorsException.java
- OptimizerVisitor.java
- ParserPlugin.java
- ParserPluginFactory.java
- Phases.java
- ProcessingUnit.java
- ResolveVisitor.java
- SourceUnit.java
- StaticImportVisitor.java
- GroovyBugError.java

## **Package: org.codehaus.groovy.tools.ast**

- TransformTestHelper.groovy
- Compiler.java
- DgmConverter.java
- ErrorReporter.java
- FileSystemCompiler.java
- GrapeMain.groovy
- GroovyClass.java

## **Package: org.codehaus.groovy.tools.gse**

- DependencyTracker.java
- StringSetMap.java

## Package: org.codehaus.groovy.tools.javac

- JavaAwareCompilationUnit.java
- JavaAwareResolveVisitor.java
- JavacCompilerFactory.java
- JavacJavaCompiler.java
- JavaCompiler.java
- JavaCompilerFactory.java
- JavaStubCompilationUnit.java
- JavaStubGenerator.java
- LoaderConfiguration.java
- RootLoader.java

## Package: org.codehaus.groovy.transform

- ASTTransformation.java
- ASTTransformationCollectorCodeVisitor.java
- ASTTransformationVisitor.java
- CategoryASTTransformation.java
- DelegateASTTransformation.java
- GroovyASTTransformation.java
- GroovyASTTransformationClass.java
- ImmutableASTTransformation.java
- LazyASTTransformation.java
- NewifyASTTransformation.java
- PackageScopeASTTransformation.java

## Package: org.codehaus.groovy.transform.powerassert

- AssertionRenderer.java
- AssertionRewriter.java
- AssertionTransformation.java
- AssertionVerifier.java
- PowerAssertionError.java
- SourceText.java
- SourceTextNotAvailableException.java
- StatementReplacingVisitorSupport.java
- TruthExpressionRewriter.java
- Value.java
- ValueRecorder.java
- SingletonASTTransformation.java

## Package: org.codehaus.groovy.util

- AbstractConcurrentDoubleKeyMap.java
- AbstractConcurrentMap.java
- AbstractConcurrentMapBase.java
- ComplexKeyHashMap.java
- DoubleKeyHashMap.java
- FastArray.java
- Finalizable.java
- HashCodeHelper.java
- LazyReference.java
- LockableObject.java
- ManagedConcurrentMap.java
- ManagedDoubleKeyMap.java

- ManagedLinkedList.java
- ManagedReference.java
- Reference.java
- ReferenceBundle.java
- ReferenceManager.java
- ReferenceType.java
- ReleaseInfo.java
- SingleKeyHashMap.java
- TripleKeyHashMap.java

## **Package: org.codehaus.groovy.runtime.typehandling**

- BigDecimalMath.java
- BigIntegerMath.java
- ClassDistance.java
- DefaultTypeTransformation.java
- FloatingPointMath.java
- GroovyCastException.java
- IntegerMath.java
- LongMath.java
- NumberMath.java
- NumberMathModificationInfo.java

## **Package: org.codehaus.groovy.runtime.wrappers**

- BooleanWrapper.java
- ByteWrapper.java
- CharWrapper.java
- DoubleWrapper.java
- FloatWrapper.java
- GroovyObjectWrapper.java
- IntWrapper.java
- LongWrapper.java
- PojoWrapper.java
- ShortWrapper.java
- Wrapper.java
- WritableFile.java
- XmlGroovyMethods.java

## **Package: org.codehaus.groovy.syntax**

- ASTHelper.java
- CSTNode.java
- Numbers.java
- ParserException.java
- ReadException.java
- Reduction.java
- RuntimeParserException.java
- SyntaxException.java
- Token.java
- TokenException.java
- TokenMismatchException.java
- Types.java

## **Package: org.codehaus.groovy.reflection**

- CachedClass.java
- CachedConstructor.java
- CachedField.java
- CachedMethod.java
- ClassInfo.java
- ClassLoaderForClassArtifacts.java
- GeneratedMetaMethod.java
- handlegen.groovy
- MethodHandle.java
- MethodHandleFactory.java
- MixinInMetaClass.java
- ParameterTypes.java
- ReflectionCache.java
- ReflectionUtils.java

## **Package: org.codehaus.groovy.reflection.stdclasses**

- ArrayCachedClass.java
- BigDecimalCachedClass.java
- BigIntegerCachedClass.java
- BooleanCachedClass.java
- ByteCachedClass.java
- CachedClosureClass.java
- CharacterCachedClass.java
- DoubleCachedClass.java
- FloatCachedClass.java
- IntegerCachedClass.java
- LongCachedClass.java
- NumberCachedClass.java
- ObjectCachedClass.java
- ShortCachedClass.java
- StringCachedClass.java
- SunClassLoader.java

## **Package: org.codehaus.groovy.runtime**

- ArrayUtil.java

## **Package: org.codehaus.groovy.runtime.callsite**

- AbstractCallSite.java
- CallSite.java
- CallSiteArray.java
- CallSiteAwareMetaMethod.java
- CallSiteClassLoader.java
- CallSiteGenerator.java
- ClassMetaClassGetPropertySite.java
- ConstructorMetaClassSite.java
- ConstructorMetaMethodSite.java
- ConstructorSite.java
- DummyCallSite.java
- GetEffectivePogoFieldSite.java
- GetEffectivePogoPropertySite.java
- GetEffectivePojoFieldSite.java
- GetEffectivePojoPropertySite.java

- GroovySunClassLoader.java
- MetaClassConstructorSite.java
- MetaClassSite.java
- MetaMethodSite.java
- NullCallSite.java
- PerInstancePojoMetaClassSite.java
- PogoGetPropertySite.java
- PogoInterceptableSite.java
- PogoMetaClassGetPropertySite.java
- PogoMetaClassSite.java
- PogoMetaMethodSite.java
- PojoMetaClassGetPropertySite.java
- PojoMetaClassSite.java
- PojoMetaMethodSite.java
- StaticMetaClassSite.java
- StaticMetaMethodSite.java
- ClassExtender.java
- ConversionHandler.java
- ConvertedClosure.java
- ConvertedMap.java
- CurriedClosure.java
- DefaultCachedMethodKey.java
- DefaultGroovyMethods.java
- DefaultGroovyMethodsSupport.java
- DefaultGroovyStaticMethods.java
- DefaultMethodKey.java

## **Package: org.codehaus.groovy.runtime.dgimpl.arrays**

- ArrayGetAtMetaMethod.java
- ArrayMetaMethod.java
- ArrayPutAtMetaMethod.java
- BooleanArrayGetAtMetaMethod.java
- BooleanArrayPutAtMetaMethod.java
- ByteArrayGetAtMetaMethod.java
- ByteArrayPutAtMetaMethod.java
- CharacterArrayGetAtMetaMethod.java
- CharacterArrayPutAtMetaMethod.java
- DoubleArrayGetAtMetaMethod.java
- DoubleArrayPutAtMetaMethod.java
- FloatArrayGetAtMetaMethod.java
- FloatArrayPutAtMetaMethod.java
- IntegerArrayGetAtMetaMethod.java
- IntegerArrayPutAtMetaMethod.java
- LongArrayGetAtMetaMethod.java
- LongArrayPutAtMetaMethod.java
- ObjectArrayGetAtMetaMethod.java
- ObjectArrayPutAtMetaMethod.java
- ShortArrayGetAtMetaMethod.java
- ShortArrayPutAtMetaMethod.java
- NumberNumberDiv.java
- NumberNumberMetaMethod.java
- NumberNumberMinus.java
- NumberNumberMultiply.java

- NumberNumberPlus.java
- FlushingStreamWriter.java
- GeneratedClosure.java
- GroovyCategorySupport.java
- GStringImpl.java
- HandleMetaClass.java
- InvokerHelper.java
- InvokerInvocationException.java
- IteratorClosureAdapter.java

## **Package: org.codehaus.groovy.runtime.metaclass**

- ClosureMetaClass.java
- ClosureMetaMethod.java
- ClosureStaticMetaMethod.java
- ConcurrentReaderHashMap.java
- MemoryAwareConcurrentReadMap.java
- MetaClassRegistryImpl.java
- MetaMethodIndex.java
- MethodHelper.java
- MethodSelectionException.java
- MissingMethodExceptionNoStack.java
- MissingMethodExecutionFailed.java
- MissingPropertyExceptionNoStack.java
- MixedInMetaClass.java
- MixinInstanceMetaMethod.java
- MixinInstanceMetaProperty.java
- NewInstanceMetaMethod.java
- NewMetaMethod.java
- NewStaticMetaMethod.java
- OwnedMetaClass.java
- ReflectionMetaMethod.java
- ReflectorLoader.java
- TemporaryMethodKey.java
- ThreadManagedMetaBeanProperty.java
- TransformMetaMethod.java
- MetaClassHelper.java
- MethodClosure.java
- MethodKey.java
- MethodRankHelper.java
- NullObject.java
- ReflectionMethodInvoker.java
- Reflector.java
- RegexSupport.java
- ReverseListIterator.java
- ScriptBytecodeAdapter.java
- ScriptReference.java
- ScriptTestAdapter.java
- StackTraceUtils.java
- StringBufferWriter.java
- TimeCategory.java

## **Package: groovy.util**

- AbstractFactory.java

- AllTestSuite.java
- AntBuilder.java
- BuilderSupport.java
- CharsetToolkit.java
- CliBuilder.groovy
- ClosureComparator.java
- ConfigObject.groovy
- ConfigSlurper.groovy
- Eval.java
- Expando.java
- Factory.java
- FactoryBuilderSupport.java
- FileNameByRegexFinder.groovy
- FileNameFinder.groovy
- GroovyCollections.java
- GroovyLog.java
- GroovyScriptEngine.java
- GroovyShellTestCase.groovy
- GroovyTestCase.java
- GroovyTestSuite.java
- IFileNameFinder.java
- IndentPrinter.java
- MapEntry.java
- Node.java
- NodeBuilder.java
- NodeList.java
- NodePrinter.java
- ObjectGraphBuilder.java
- ObservableList.java
- ObservableMap.java
- OrderBy.java
- PermutationGenerator.java
- Proxy.java
- ProxyGenerator.java
- ResourceConnector.java
- ResourceException.java
- ScriptException.java

## **Package: org.codehaus.groovy.vmplugin.v4**

- Java4.java

## **Package: org.codehaus.groovy.vmplugin.v5**

- Java5.java
- JUnit4Utils.java
- PluginDefaultGroovyMethods.java
- TestNgUtils.java
- VMPlugin.java
- VMPluginFactory.java

---

Module: test

## **Package: groovy.mockPackage: groovy.mock.interceptor**

- Demand.groovy
  - LooseExpectation.groovy
  - MockFor.groovy
  - MockInterceptor.groovy
  - MockProxyMetaClass.java
  - StrictExpectation.groovy
  - StubFor.groovy
- 

Module: jmx

## Package: groovy.jmxPackage: groovy.jmx.builder

- JmxAttributeInfoManager.groovy
- JmxBeanExportFactory.groovy
- JmxBeanFactory.groovy
- JmxBeanInfoManager.groovy
- JmxBeansFactory.groovy
- JmxBuilder.groovy
- JmxBuilderException.java
- JmxBuilderModelMBean.java
- JmxBuilderTools.groovy
- JmxClientConnectorFactory.groovy
- JmxEmitterFactory.groovy
- JmxEventEmitter.java
- JmxEventEmitterMBean.java
- JmxEventListener.java
- JmxListenerFactory.groovy
- JmxMetaMapBuilder.groovy
- JmxOperationInfoManager.groovy
- JmxServerConnectorFactory.groovy
- JmxTimerFactory.groovy

## Package: groovy.util

- GroovyMBean.java
- 

Module: swing

## Package: groovy.model

- ClosureModel.java
- DefaultTableColumn.java
- DefaultTableModel.java
- FormModel.java
- NestedValueModel.java
- PropertyModel.java
- ValueHolder.java
- ValueModel.java

## Package: groovy.swingPackage: groovy.swing.binding

- AbstractButtonProperties.java
- AbstractSyntheticBinding.java
- AbstractSyntheticMetaMethods.groovy
- JComboBoxMetaMethods.groovy
- JComboBoxProperties.java

- JComponentProperties.java
- JScrollBarProperties.java
- JSliderProperties.java
- JSpinnerProperties.java
- JTableMetaMethods.groovy
- JTableProperties.java
- JTextComponentProperties.java

## **Package: groovy.swing.factory**

- ActionFactory.groovy
- BeanFactory.groovy
- BevelBorderFactory.groovy
- BindFactory.groovy
- BindGroupFactory.groovy
- BindProxyFactory.groovy
- BoxFactory.groovy
- BoxLayoutFactory.groovy
- ButtonGroupFactory.groovy
- CollectionFactory.groovy
- ComboBoxFactory.groovy
- ComponentFactory.groovy
- CompoundBorderFactory.groovy
- DialogFactory.groovy
- EmptyBorderFactory.groovy
- EtchedBorderFactory.groovy
- FormattedTextFactory.groovy
- FrameFactory.groovy
- GridBagFactory.groovy
- ImageIconFactory.groovy
- InternalFrameFactory.groovy
- LayoutFactory.groovy
- LineBorderFactory.groovy
- ListFactory.groovy
- MapFactory.groovy
- MatteBorderFactory.groovy
- RendererFactory.groovy
- RichActionWidgetFactory.groovy
- RootPaneContainerFactory.groovy
- ScrollPaneFactory.groovy
- SeparatorFactory.groovy
- SplitPaneFactory.groovy
- SwingBorderFactory.groovy
- TabbedPaneFactory.groovy
- TableFactory.groovy
- TableLayoutFactory.groovy
- TableModelFactory.groovy
- TextArgWidgetFactory.groovy
- TitledBorderFactory.groovy
- WidgetFactory.groovy
- WindowFactory.groovy

## **Package: groovy.swing.impl**

- ClosureRenderer.java

- ComponentFacade.java
- ContainerFacade.java
- DefaultAction.java
- Startable.java
- TableLayout.java
- TableLayoutCell.java
- TableLayoutRow.java
- LookAndFeelHelper.groovy
- SwingBuilder.groovy

## Package: org.codehaus.groovy.runtime.metaclass

- SwingGroovyMethods.java
- 

Module: xml

## Package: groovy.util.slurpersupport

- Attribute.java
- Attributes.java
- FilteredAttributes.java
- FilteredNodeChildren.java
- GPathResult.java
- NoChildren.java
- Node.java
- NodeChild.java
- NodeChildren.java
- NodeIterator.java
- ReplacementNode.java
- XmlNodePrinter.java
- XmlParser.java
- XmlSlurper.java

## Package: groovy.xmlPackage: groovy.xml.dom

- DOMCategory.java
- DOMUtil.java
- DOMBuilder.java
- Entity.groovy
- FactorySupport.java
- MarkupBuilder.java
- MarkupBuilderHelper.java
- Namespace.java
- NamespaceBuilder.java
- NamespaceBuilderSupport.java
- QName.java
- SAXBuilder.java
- StaxBuilder.groovy
- StreamingDOMBuilder.groovy
- StreamingMarkupBuilder.groovy

## Package: groovy.xml.streamingmarkupsupport

- AbstractStreamingBuilder.groovy
- BaseMarkupBuilder.java
- Builder.java

- StreamingMarkupWriter.java
- StreamingSAXBuilder.groovy
- XmlUtil.java

## Package: org.codehaus.groovy.tools.xml

- DomToGroovy.java
- 

Module: sql

## Package: groovy.sql

- CallResultSet.java
- DataSet.java
- ExpandedVariable.java
- GroovyResultSet.java
- GroovyResultSetExtension.java
- GroovyResultSetProxy.java
- GroovyRowResult.java
- InOutParameter.java
- InParameter.java
- OutParameter.java
- ResultSetMetaDataWrapper.java
- ResultSetOutParameter.java
- Sql.java
- SqlOrderByVisitor.java
- SqlWhereVisitor.java

## Package: org.codehaus.groovy.runtime.metaclass

- SqlGroovyMethods.java
- 

Module: web

## Package: groovy.servlet

- AbstractHttpServlet.java
  - GroovyServlet.java
  - ServletBinding.java
  - ServletCategory.java
  - TemplateServlet.java
- 

Module: template-engine

## Package: groovy.text

- GStringTemplateEngine.java
  - SimpleTemplateEngine.java
  - Template.java
  - TemplateEngine.java
  - XmlTemplateEngine.java
- 

Module: bean-scripting-framework

## Package: org.codehaus.groovy.bs

- CachingGroovyEngine.java

- GroovyEngine.java
- 

Module: javax.script (jsr-223)

## Package: org.codehaus.groovy.jsr223

- GroovyCompiledScript.java
  - GroovyScriptEngineFactory.java
  - GroovyScriptEngineImpl.java
- 

Module: groovydoc

## Package: org.codehaus.groovy.tools.groovydoc

- ClasspathResourceManager.java
- ExternalGroovyClassDoc.java
- FileOutputTool.java
- FileSystemResourceManager.java
- GroovyDocTemplateEngine.java
- GroovyDocTool.java
- GroovyDocWriter.java
- GroovyRootDocBuilder.java

## Package:

## org.codehaus.groovy.tools.groovydoc.gstringTemplates.to pLevel

- LinkArgument.java
- MockOutputTool.java
- OutputTool.java
- ResourceManager.java
- SimpleGroovyAnnotationRef.java
- SimpleGroovyClassDoc.java
- SimpleGroovyClassDocAssembler.java
- SimpleGroovyConstructorDoc.java
- SimpleGroovyDoc.java
- SimpleGroovyExecutableMemberDoc.java
- SimpleGroovyFieldDoc.java
- SimpleGroovyMemberDoc.java
- SimpleGroovyMethodDoc.java
- SimpleGroovyPackageDoc.java
- SimpleGroovyParameter.java
- SimpleGroovyProgramElementDoc.java
- SimpleGroovyRootDoc.java
- SimpleGroovyTag.java
- SimpleGroovyType.java
- GroovyStarter.java

## Package: org.codehaus.groovy.groovydoc

- GroovyAnnotationRef.java
- GroovyClassDoc.java
- GroovyConstructorDoc.java
- GroovyDoc.java

- GroovyDocErrorReporter.java
  - GroovyExecutableMemberDoc.java
  - GroovyFieldDoc.java
  - GroovyMemberDoc.java
  - GroovyMethodDoc.java
  - GroovyPackageDoc.java
  - GroovyParameter.java
  - GroovyProgramElementDoc.java
  - GroovyRootDoc.java
  - GroovyTag.java
  - GroovyType.java
  - GroovyException.java
  - GroovyExceptionInterface.java
- 

Module: shell

## Package: org.codehaus.groovy.tools.shell

- BufferManager.groovy
- Command.groovy
- CommandAlias.groovy
- CommandRegistry.groovy
- CommandSupport.groovy
- ComplexCommandSupport.groovy
- ExitNotification.groovy
- Groovysh.groovy
- InteractiveShellRunner.groovy
- Interpreter.groovy
- IO.java
- Main.groovy
- Parser.groovy
- Shell.groovy
- ShellRunner.groovy

## Package: org.codehaus.groovy.tools.shell.commands

- AliasCommand.groovy
- ClearCommand.groovy
- DisplayCommand.groovy
- EditCommand.groovy
- ExitCommand.groovy
- HelpCommand.groovy
- HistoryCommand.groovy
- ImportCommand.groovy
- InspectCommand.groovy
- LoadCommand.groovy
- PurgeCommand.groovy
- RecordCommand.groovy
- RegisterCommand.groovy
- SaveCommand.groovy
- SetCommand.groovy
- ShadowCommand.groovy
- ShowCommand.groovy

## Package: org.codehaus.groovy.tools.shell.util

- ClassNameCompletor.groovy
  - HelpFormatter.groovy
  - Logger.java
  - MessageSource.java
  - NoExitSecurityManager.java
  - Preferences.java
  - SimpleCompletor.java
  - XmlCommandRegistrar.groovy
- 

Module: console

## Package: groovy.inspect

- Inspector.java

## Package: groovy.inspect.swingui

- AstBrowser.groovy
- AstBrowserProperties.groovy
- ObjectBrowser.groovy
- ScriptToTreeNodeAdapter.groovy
- TableMap.java
- TableSorter.java

## Package: groovy.ui

- Console.groovy
- ConsoleActions.groovy
- ConsoleApplet.groovy
- ConsoleSupport.java
- ConsoleTextEditor.java
- ConsoleView.groovy
- GroovyMain.java
- GroovySocketServer.java
- HistoryRecord.groovy

## Package: groovy.ui.icons

- InteractiveShell.java
- OutputTransforms.groovy
- SystemOutputInterceptor.java

## Package: groovy.ui.text

- AutoIndentAction.groovy
- FindReplaceUtility.java
- GroovyFilter.java
- StructuredSyntaxDocumentFilter.java
- StructuredSyntaxHandler.java
- StructuredSyntaxResources.java
- TextEditor.java
- TextUndoManager.java

## Package: groovy.ui.view

- BasicContentPane.groovy

- BasicMenuBar.groovy
  - BasicStatusBar.groovy
  - BasicToolBar.groovy
  - Defaults.groovy
  - GTKDefaults.groovy
  - MacOSXDefaults.groovy
  - MacOSXMenuBar.groovy
  - WindowsDefaults.groovy
- 

Module: java2groovy

## Package: org.codehaus.groovy.antlr.java

- Groovifier.java
- Java2GroovyConverter.java
- Java2GroovyMain.java
- JavaLexer.java
- JavaRecognizer.java
- JavaTokenTypes.java
- PreJava2GroovyConverter.java
- LexerFrame.java
- LineColumn.java
- Main.java

## Roadmap

Considering our limited human resources and time constraints, it is hard to give definitive and accurate estimates of the milestones we are going to release.

### Groovy 2.1

Groovy 2.1 is the latest stable version of Groovy.

#### *Feature set*

- **offers full support for the JDK 7 “invoke dynamic”** bytecode instruction and API for improved performance,
- goes beyond conventional static type checking capabilities with a **special annotation to assist with documentation and type safety of Domain-Specific Languages**,
- adds **static type checker extensions**,
- provides **additional compilation customization** options,
- features a **meta-annotation facility** for combining annotations elegantly.

#### *Releases*

- Groovy 2.1 beta: end of December 2012 
- Groovy 2.1 RC: early January 2013 
- Groovy 2.1 final: end of January 2013 

### Groovy 2.2 (Q3 2013)

#### *Feature set for consideration*

- Class-loading-less compilation
  - using ASM when needed, to avoid loading/initializing classes needed during the compilation
- Stub-less joint compilation
  - to avoid various problems with the stubs and the various compilation phases

- Static traits or mixins
  - but dependent on stub-less joint compilation issues

### **Groovy 3.0 (Q1 2014)**

#### ***Feature set for consideration***

- New Meta-Object Protocol dedicated to fully leverage "invoke dynamic"
- Rewrite the Groovy grammar from scratch with Antlr 4
- Retrofitting Groovy closures to accommodate JDK 8 upcoming closures for interoperability

### **Groovy 4.0 (Q1 2015)**

#### ***Feature set for consideration***

- No concrete plans at the moment

### **Other topics we could consider for later Groovy**

- add static "trait" capabilities
- compiler related:
  - investigate the integration of the Eclipse joint compiler to replace the Groovy stub-based joint compiler
  - investigate making the groovyc compiler multithreaded
  - Incremental compiler
- a treturn keyword or dedicated AST transformation for tail calls for closures and methods
- ability to pass expression trees / AST nodes as parameters (see C# 4's own expression tree)
- lexical categories
- a symbol concept, a bit like Ruby's :symbol, or like Java interned strings
- co-routines and/or generators
- pattern matching
- parser combinators
- a native template engine compiling to AST (faster, correct line numbers for error reporting, optimized outputting, etc.)
- ...

### **notes on JDK 7 and JDK 8 future**

A few notes, as of September 2010, on the future of Java/JDK 7, and even 8.

Mark Reinhold's post on a possible roadmap for Java 7/8: [http://blogs.sun.com/mr/entry/rethinking\\_jdk7](http://blogs.sun.com/mr/entry/rethinking_jdk7)

Jo Darcy on the Project Coin proposals: [http://blogs.sun.com/darcy/entry/project\\_coin\\_jdk\\_7\\_plan](http://blogs.sun.com/darcy/entry/project_coin_jdk_7_plan)

And this link provides the best overview of what's to expect for each version: <http://openjdk.java.net/projects/jdk7/features/>

The Project Coin proposals retained and presented at JavaOne: [http://blogs.sun.com/darcy/entry/project\\_coin\\_javaone\\_2010](http://blogs.sun.com/darcy/entry/project_coin_javaone_2010)

Latest notes from Mark Reinhold, after the JCP meeting in Germany, post JavaOne: [http://blogs.sun.com/mr/entry/planning\\_details](http://blogs.sun.com/mr/entry/planning_details)

Some concrete examples of Project Coin usage: <http://aruld.info/rest-of-project-coin-explored-advantage-java-7/>

### **Not Yet Documented**

The following items need documentation.

When documenting, please create a page for each and link to it.

- Generics
- [Closures](#)
  - Closure's variable resolving strategy
- Joint Groovy/Java compiler, and its options and impact on the Groovyc Ant task
- Explain the new Foo(a:1, b:2) shortcut notation calls setA() and setB() after new Foo() is created and initialized, and that this is not an atomic thread-safe operation
- Type handling
  - 'as' keyword with its different meanings
  - conversion vs coercion vs autoboxing
  - closure and map coercion to interfaces and classes
  - general typing approach (explicit static typing vs duck typing)
  - static methods called from instances (see GROOVY-1706)
  - Conversion rules, in all their complexity
- [Swing Builder](#)
- more builder documentation - [Builders](#)
- Scoping
  - handling of "return, break, and continue"
  - open blocks vs closed blocks
  - Name scoping rules
  - scoping rules for Closures
- method dispatch algorithm
- Method calls and property references, in all their complexity
- 'special' variables like ~~'owner' in Closures~~, 'out' in Groovlets etc.
- Class member naming conventions, and other JVM interfaces

Also, when you are working on a page that is unfinished, label that page with "TODO" and it will appear in this list:

[SwingBuilder.button](#)

---

[Not Yet Documented](#)

---

[FactoryBuilderSupport](#)

---

[Gldapwrap - Usage](#)

---

[Feature Wish List](#)

---

### **Gldapwrap - Usage**

 This package is not actively maintained, but it will be re-released once it has been reworked as time permits.

### ***Gldapwrap - Usage***

To use Gldapwrap, you need to make sure the Gldapwrap jar is in your classpath as well as the Spring LDAP jars (provided in the download). An easy way to do this is to put the jars in `~/ .groovy/lib/`.

First thing you need to do is define a schema class. A schema class is just an ordinary POGO.

```
class MyLdapEntry
{
  List objectclass
  String distinguishName
}
```

That's all you need to do to define a schema.

Next is to create a `GldapwrapTemplate` which is basically the details of the LDAP server you want to connect to.

```
import gldapwrap.GldapwrapTemplate

def template = new GldapwrapTemplate(
  url: "ldap://example.com",
  base: "dc=example,dc=com",
  userDn: "cn=admin,dc=example,dc=com",
  password: "secret"
)
```

Next you need to inject the template into your schema class.

```
import
gldapwrap.GldapwrapInjector(MyLdapEntry,
template)
```

Now you can do searches.

```
List entries = MyLdapEntry.find() // Find
all entries at base

entries = MyLdapEntry.find(
  filter: "(objectclass=person)",
  searchScope:
  javax.naming.directory.SearchControls.SUBTRE
  E_SCOPE
) // Find all people in the whole directory

entries = MyLdapEntry.find(
  filter: "(objectclass=person)",
  searchScope:
  javax.naming.directory.SearchControls.SUBTRE
  E_SCOPE
  base: "ou=People"
  countLimit: 50
) // Find the first 50 people in the
directory under the people OU
```

See [Searching](#) for more detail about searching.

## Articles

Here are a bunch of articles and blog posts on all things groovy

### Beginners Tutorials

- The K&R Series by Simon P. Chappell
  1. [Groovin' with Kernighan and Ritchie](#)
  2. [Groovin' with K&R 2](#)
  3. [Groovin' and Regroovin'](#)
  4. [Groovin' with K&R 4](#)
- Getting Groovy series by James Williams  
[Originally a series of three articles. It has been merged into one document.]  
[Getting Groovy Without the Bad Clothes](#)

## James Strachan talks about Groovy

- A [collection](#) of references to articles and emails by James Strachan on the topic of Groovy.

## Guillaume Laforge interviews on Groovy and Grails

- [Vanward / Stelligent interview](#)
- [IndicThreads interview](#)

## General

- [Templates in Groovy](#) by JavaBeat.
- [Closures in Groovy](#) by JavaBeat.
- [Web Development in Groovy using Groovlets](#) by JavaBeat.
- [Introduction to Groovy - Scripting Language](#) by JavaBeat.
- [Using Groovy to Send Emails](#) by Paul King.
- Groovy: XML Without the Bloat by Pan Pantziarka: [Part 1](#) and [Part 2](#)
- Russel Winder gave a talk "[Builders: How MOPs Make Life Easy](#)" at ACCU 2007 which focused on the way Groovy does things.
- [Implementing Domain-Specific Languages with Groovy](#) - tutorial given by [Guillaume Laforge](#) and John Wilson at QCon 2007
- BEA Dev2Dev features an [introduction to Groovy and Grails](#)
- Scott Hickey gave a [presentation](#) about Groovy at the Omaha Java User Group, July 2006
- Russel Winder had an article introducing Groovy published in *{CVU}* **18**(3):3-7. *{CVU}* is the journal of the [ACCU](#). For more details on the article click [here](#).
- Scott Davis gave a recent [presentation](#) about Groovy even mentioning a few words about [Grails](#).
- [Groovying With the JVM](#) was a presentation given by [Russel Winder](#) at the [ACCU](#) 2006 conference.
- [It's a Groovy Day!](#) by Eric Armstrong (inspired by Rod Cope's presentation at JavaOne 2005)
- [JavaPolis 2004 presentation](#) or [video](#) from James and Dion along with a [snap of James rambling](#) 😊
- JSR 241 - Nov 2004 London Conference presentations
  - [Keynote\(mp3 - 34Mb\)](#) by James Strachan
  - [History of Groovy](#) by Jeremy Rayner
  - [User Feedback \(mp3 - 15Mb\)](#) by Guillaume LaForge
- Ian Darwin wrote [this article](#) for O'Reilly
- John Wilson gave [this presentation](#) on XML processing in Groovy at XMLOpen 2004 in Cambridge
- Ian Darwin gave [this presentation at the Toronto JUG in November] <http://www.darwinsys.com/groovy/jugslides-20041102.pdf>
- James Strachan and Rod Cope gave [this presentation at JavaOne 2004](#) or [as PDF](#)
- Alexander Schmid gave [this presentation](#) at the JA00 in Cannes
- Rod Cope gave [this presentation](#) at the Denver JUG
- Laurent Weichberger gave [this presentation](#) at JSPRING in the Netherlands
- Mike Spille wrote a great [review of Groovy](#)
- Ted Leung did a great [presentation at SeaJUG](#)
- Gerald Bauer did a presentation at the [Austria JUG](#) and another one [in Vancouver](#)
- Mark Volkmann has written the excellent [Groovy - Scripting in Java](#)
- An old presentation James Strachan gave at CodehausOne August 2003 is available as a [PPT](#)
- Marc Hedlund has written several very good introductory articles about getting stuff done with Groovy (especially with the SwingBuilder). The index of all his articles can be found at [O'Reilly's website](#).
- Articles from the [Practically Groovy](#) series by Andrew Glover
  - [Smooth operators](#)
  - (25 Oct 2005) [Of MOPs and mini-languages](#)

- (20 Sep 2005) [Functional programming with curried closures](#)
- (23 Aug 2005) [Groovy's growth spurt](#)
- (19 Jul 2005) [Stir some Groovy into your Java apps](#)
- [Mark it up with Groovy Builders](#)
- (12 Apr 2005) [Go server side up, with Groovy](#)
- (15 Mar 2005) [MVC programming with Groovy templates](#)
- (15 Feb 2005) [JDBC programming with Groovy](#)
- (11 Jan 2005) [Ant scripting with Groovy](#)
- (14 Dec 2004) [Unit test your Java code faster with Groovy](#)
- (09 Nov 2004) Craig Castelaz guides you through [Groovy closures](#) on java.net
- If you're [getting to know Groovy](#) John Zukowski will bring you up to speed with Groovy (syntax prior to the JSR syntax)
- Matthias Luebken wrote about [Implementing OSGi-Services in Groovy](#).

## References

- Jeremy Rayner has created a [Groovy reference card](#) with [latex source](#)

## French articles

- [Guillaume Laforge](#) gave an [introductory presentation](#) of Groovy at the Parisian [OSS-Get Together](#) event
- Guillaume speaks about the advantage of using dynamic languages to increase the semantic density to
- [Introduction au langage de script Groovy](#) on the JDN site
- [Introduction au langage de script Groovy](#) on developpez.com
- [Intégrer JXTA dans une application Web avec JSF et Groovy](#) par Bertrand Goetzmann
- [Invoquer un service JBI \(Java Business Integration\) écrit en Groovy avec AJAX](#) par Bertrand Goetzmann

## German articles

- June 2009: Joachim Baumann presented the new functionality in Groovy 1.6 in a talk at the [Entwicklertag Karlsruhe](#). The slides can be downloaded [here](#)
- October 2008: Lars Blumberg, Christoph Hartmann & Arvid Heise did a seminar about **Groovy Meta Programming** at the Hasso-Plattner-Institute. [Groovy Meta Programming Paper \(German\)](#) and [Groovy Meta Programming Slides \(English\)](#)
- May 2007: Joachim Baumann presented Groovy in a talk at the [Entwicklertag Karlsruhe](#). The slides can be downloaded [here](#)
- Dierk's Groovy series in JavaMagazin.
  - 8.2006 **Groovy für Java-Entwickler: Dynamische Programmierung auf der Java-Plattform** [Dynamischer Nachwuchs](#)
  - 9.2006 **Groovy für Java-Entwickler: Ausdruckskraft durch starke Syntax** [Klassen- und Objektnotation, Referenzierungsmöglichkeiten, Operatoren, Kontrollstrukturen und Meta-Objekt-Protokoll](#)
  - 10.2006 **Groovy-Datentypen** [First class citizens: Zahlen, Strings, Reguläre Ausdrücke, Listen, Maps, Ranges und Closures](#)
  - 11.2006 **Ausgewählte Groovy-Beispiele** [Groovy everywhere](#)
  - 12.2006 **Grails** [Groovy für Java-Entwickler](#)
  - 27.12.2006 [Interview: Groovy - das Beste aus der Java- und der Scripting-Welt vereinen](#)
- Joachim Baumann gave a [presentation](#) about Groovy in German at the "Symposium: Trends in der Informations- und Kommunikationstechnik" in Stuttgart, September 2006.
- An [article](#) by Alexander Schmid
- [Sigs Datacom article](#)
- Dierk's Groovy presentation at [JAX 2006](#) is attached as [Groovy at JAX pub.zip](#).

- Dierk's Groovy usage patterns article in [iX Magazin 7/06](#)

## Korean articles

- [Groovy Language Study](#)
- [Groovy I, Dynamic Agile Scripting Language](#) (Korean Language) by Pilho Kim
- [Groovy II, Groovlet and GSP](#) (Korean Language) by Pilho Kim
- [Groovy III, Compare Groovy to Other Languages](#) (Korean Language) by Pilho Kim

## Russian articles

- [What is Groovy](#) - A brief introduction into Groovy language features in Russian
- [Groovy + JSON + Prototype in Action](#) - Example of using Groovy in a 'real' project
- [Groovy: Easy extracting XML from database \(Russian edition\)](#)
- [The "Exception #7" conference Groovy presentation materials](#), by [Vadim Voituk](#)
- ... and [more russian Groovy articles](#)

## Japanese articles

- coverage of our JavaOne talk [June 2004](#)

## Spanish articles

- [Introduccion a Groovy I](#) - Introduction to Groovy part I by Andres Almiray at [Groovy.org.es](#)
- [Introduccion a Groovy II](#) - Introduction to Groovy part II by Andres Almiray at [Groovy.org.es](#)

## Success Stories

This page contains Success Stories from groovy users. A Success Story should contain information about the company, the project, the number of people involved, and eventually a link the homepage of the project or company. It would be nice to include statements about how groovy impacted on the company.

It would also be nice to include companies "know or suspected" to use groovy, providing evidences of this (articles, statements, etc).

### ***We use Groovy!***

Company:  
Project:  
Description:  
Link:;

### ***Seems they use Groovy...***

Company:  
Project:  
Description:  
Link:

## Version Scheme

### The Groovy Version Scheme

#### Since Groovy 2.0.0:

Since Groovy 2.0.0 we follow the scheme as described in <http://semver.org/>. This means the next minor version

after 2.0.0 is 2.1.0, the first bugfix version after 2.0 is 2.0.1 and the next major version will be 3.0.0.

### **Before Groovy 2.0.0:**

Before Groovy 2.0.0 we followed a version scheme where we have X.Y.Z, where X.Y is the major version, and Z the minor version. Bugfix versions were not really done, you had to upgrade to the next minor version for that. Since Groovy 1.0 we incremented only the Y for a new major version. The increment of X we wanted to leave for a very big breaking change, like a new MOP. The last major version in these scheme is 1.8(.0), 1.8.1 is the first minor and bugfix version. The major versions in the past using this scheme are: 1.8, 1.7, 1.6, 1.5, 1.0. Each of them having around 10 minor/bugfix versions.

### **Official Major Version:**

The official major version is the current major version that should/can be used by the developers if they are not bound to a specific major version.

### **Maintenance Release Branch:**

Here we indicate a former major version's bugfix release.

### **How long is a major version maintained?**

That depends on the users. Let's say we have X in maintenance and Y is the official major version, then if a new major version Z is released, Y goes into maintenance. Usually we make one or two more bugfix releases for X and then we discontinue it, unless there are strong requests to have certain things fixed for users that can absolutely not upgrade.

## **Documentation**

### **[Getting Started with Groovy](#)**

Installation and quickstart instructions, tutorials, feature overview.

### **[Using Groovy](#)**

Bean scripting, compile-time metaprogramming, AST transformations, dynamic features, IDE support, XML processing, GUI programming, Ant integration, more.

### **[Cookbook Examples](#)**

Practical examples that focus on common applications and tasks.

### **[Advanced Usage](#)**

Design patterns, polyglot programming, Ant troubleshooting, security, compiling, refactoring, more.

### **[Testing with Groovy](#)**

Groovy mocks, model-based testing, unit testing, testing Web applications and Web services, integration with other frameworks, test coverage, more.

### **[Developing with Groovy](#)**

Building Groovy from source, setting up the environment, continuous integration, release process, more.

### **Feature Highlights**

- [GroovyMarkup](#)
- [GPath](#)

- [Groovy Beans](#)
- [Using Ant from Groovy](#)
- [Regular Expressions](#)
- [Groovy Templates](#)
- [Dynamic Groovy](#)
- [Groovy and JMX](#)
- [Spring Factories and Groovy](#)

## Reference

- [Groovy API](#)
- [Groovy JDK](#)
- [DZone quick reference card](#)
- [Groovy wiki collection \(20120226\)](#)

## Getting Started Guide

- [Download](#)
  - [InvokeDynamic support](#)
- [Feature Overview](#)
  - [Groovlets](#)
  - [Groovy Beans](#)
  - [GroovyMarkup](#)
  - [Groovy Templates](#)
- [Quick Start](#)
  - [Installing Groovy](#)
    - [Installing Groovy and Grails on the Eee PC](#)
  - [Running](#)
- [Beginners Tutorial](#)
  - [Tutorial 1 - Getting started](#)
  - [Tutorial 2 - Code as data, or closures](#)
  - [Tutorial 3 - Classes and Objects](#)
  - [Tutorial 4 - Regular expressions basics](#)
  - [Tutorial 5 - Capturing regex groups](#)
  - [Tutorial 6 - Groovy SQL](#)
- [Differences to Other Languages](#)
  - [Differences from Java](#)
  - [Differences from Python](#)
  - [Differences from Ruby](#)
- [Groovy style and language feature guidelines for Java developers](#)
- [For those new to both Java and Groovy](#)
  - [JN0025-Starting](#)
  - [JN0515-Integers](#)
  - [JN0525-Decimals](#)
  - [JN0535-Floats](#)
  - [JN0545-Dates](#)
  - [JN1015-Collections](#)
  - [JN1025-Arrays](#)
  - [JN1035-Maps](#)
  - [JN1515-Characters](#)
  - [JN1525-Strings](#)
  - [JN1535-Patterns](#)
  - [JN2015-Files](#)
  - [JN2025-Streams](#)

- [JN2515-Closures](#)
- [JN2525-Classes](#)
- [JN2535-Control](#)
- [JN3015-Types](#)
- [JN3025-Inheritance](#)
- [JN3035-Exceptions](#)
- [JN3515-Interception](#)
- [JN3525-MetaClasses](#)
- [JN3535-Reflection](#)
- [Groovy for the Office](#)
- [Groovy Quick Start Project](#)

## Download

In this download area, you will be able to download the distribution (binary and source), the Windows installer (for some of the versions) and the documentation for Groovy.

For a quick and effortless start on Mac OSX, Linux or Cygwin, you can use [GVM](#) (the Groovy enVironment Manager) to download and configure any Groovy version of your choice. Basic [instructions](#) can be found below.

## Groovy 2.1

Groovy 2 is the latest official major [version of Groovy](#).

Learn more about Groovy 2.1 in the [release notes](#).

### *Groovy 2.1.5*

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

### *Groovy 2.1.4*

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

### *Groovy 2.1.3*

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

### **Groovy 2.1.2**

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

## **Groovy 2.0**

Groovy 2.0 is the [minor version](#) before Groovy 2.1.

Learn more about Groovy 2.0 in the [release notes](#).

### **Groovy 2.0.8**

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

### **Groovy 1.8 (Maintenance Release)**

Groovy 1.8 is the latest maintenance release branch of the Groovy 1 major version.

Learn more about the Groovy 1.8 in the [release notes](#).

### **Groovy 1.8.9**

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

You can have a look at the [JIRA release notes](#).

## **Snapshots**

For those who want to test the very latest versions of Groovy and live on the bleeding edge, you can use our [snapshot builds](#). As soon as a build succeeds on our continuous integration server a snapshot is deployed to Codehaus' snapshot repository.

## **Older Releases**

All maintenance of the Groovy 1.5.X, 1.6.X and 1.7.X series has now stopped. These are the last released versions.

# Groovy 1.7.11

Download zip: [Binary Release](#) | [Source Release](#)

Download documentation: [JavaDoc and zipped online documentation](#)

You can have a look at the [JIRA release notes](#).

## *Groovy 1.6.9*

Download zip: [Binary Release](#) (JDK 1.4 version) | [Source Release](#)

Download documentation: [JavaDoc and zipped online documentation](#)

You can have a look at the [JIRA release notes](#).

## *Groovy 1.5.8*

Download zip: [Binary Release](#) | [Source Release](#)

Download Windows-Installer: [Binary Release](#)

Download unofficial Fedora/RHEL/CentOS package: [Binary Release](#) | [Source Release](#)

Download documentation: [JavaDoc and zipped online documentation](#)

The release notes are [Release Notes](#).

## Legacy Groovy 1.0

If you still need to use the old Groovy 1.0, here are the links you may be interested in:

Download zip: [Binary Release](#) | [Source Release](#)

Download unofficial Ubuntu/Debian package: [Binary Release](#)

Download Windows-Installer: [Binary Release](#)

Download tar/gz: [Binary Release](#) | [Source Release](#)

Download Javadoc: [Javadoc zip](#)

[Once you've downloaded the distribution, please read the installation instructions.](#)

Other versions can be found in the [distributions archive](#).

## Maven Repositories

If you wish to embed Groovy in your application, you may just prefer to point to your favourite maven repositories or the [codehaus maven repository](#). You should consult the individual pom files for the exact details but here is a summary of the main supported artifactIds and groupIds:

## Groovy 2.0.X and newer

Gradle	Maven	Explanation
--------	-------	-------------

'org.codehaus.groovy:groovy:x.y.z'	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy</artifactId> <version>x.y.z</version>	Just the core of groovy without the modules (see below). Treats Antlr, ASM, etc. as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars.
'org.codehaus.groovy:groovy-\$module:x.y.z'	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy-\$module</artifactId> <version>x.y.z</version>	"\$module" stands for the different optional groovy modules "ant", "bsf", "console", "docgenerator", "groovydoc", "groovysh", "jmx", "json", "jsr223", "servlet", "sql", "swing", "test", "testng" and "xml". Example: <artifactId>groovy-sql</artifactId>
'org.codehaus.groovy:groovy-all:x.y.z'	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy-all</artifactId> <version>x.y.z</version>	The core plus all the modules. Also includes jarjar versions of Antlr, ASM, Commons-CLI and Retrotranslator runtime. Allows you or your other dependencies (e.g. Hibernate) to use other versions of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.

To use the [InvokeDynamic](#) version of the jars just append ':indy' for Gradle or <classifier>indy</classifier> for Maven.

### Groovy 1.6.X - 1.8.X

Gradle	Maven	Explanation
'org.codehaus.groovy:groovy:x.y.z'	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy</artifactId> <version>x.y.z</version>	Treats Antlr, ASM, etc. as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.

'org.codehaus.groovy:groovy-all:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-all&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Includes jarjar versions of Antlr, ASM, Commons-CLI and Retrotranslator runtime. Allows you or your other dependencies (e.g. hibernate) to use other versions of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
--	--	--

## Groovy 1.1.X - 1.5.X

Gradle	Maven	Explanation
'org.codehaus.groovy:groovy:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Treats Antlr and ASM as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'org.codehaus.groovy:groovy-all:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-all&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Includes jarjar versions of Antlr, ASM and Commons-CLI. Allows you or your other dependencies (e.g. hibernate) to use other versions of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'org.codehaus.groovy:groovy-all-minimal:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-all-minimal&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Includes jarjar versions of Antlr and ASM. Does not include Commons-CLI or any optional dependencies. Not suitable by itself if you want to use any tools which do command-line processing, e.g. groovyc, GroovyShell, ...

## Groovy 1.0

Available in the [Maven 1](#) and [Maven 2](#) repositories.

Gradle	Maven	Explanation
--------	-------	-------------

'groovy:groovy:x.y.z'	<groupId>groovy</groupId> <artifactId>groovy</artifactId> <version>1.0</version>	Treats Antlr and ASM as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars. Dependencies mandated for all optional parts of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'groovy:groovy-all:x.y.z'	<groupId>groovy</groupId> <artifactId>groovy-all</artifactId> <version>1.0</version>	Includes jarjar versions of Antlr and ASM. Allows you or your other dependencies (e.g. hibernate) to use other versions of these jars. Dependencies mandated for all optional parts of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'groovy:groovy-all-minimal:x.y.z'	<groupId>groovy</groupId> <artifactId>groovy-all-minimal</artifactId> <version>1.0</version>	Includes jarjar versions of Antlr and ASM. Does not include any optional dependencies.

For historical purposes, the releases candidates for 1.0 are also available by using the appropriate version instead of 1.0 in the above version tag.

### Snapshot Releases

In addition to the stable and milestone releases you can find intermediate SNAPSHOT releases at the [codehaus snapshot maven repository](#).

### GVM (the Groovy enVironment Manager)

This tool makes installing Groovy on any Bash platform (Mac OSX, Linux, Cygwin, Solaris or FreeBSD) very easy.

Simply open a new terminal and enter:

```
$ curl -s get.gvmtool.net | bash
```

Follow the instructions on-screen to complete installation.

Open a new terminal or type the command:

```
$ source "$HOME/.gvm/bin/gvm-init.sh"
```

Then install the latest stable Groovy:

```
$ gvm install groovy
```

After installation is complete and you've made it your default version, test it with:

```
$ groovy -version
```

That's all there is to it!

## Other ways to get Groovy

If you're on MacOS and have [Homebrew](#) installed, you can run "brew install groovy" to install Groovy.

If you're on MacOS and have [MacPorts](#) installed, you can run "sudo port install groovy" to install the latest Groovy release.

If you're on Windows, you can also use the [NSIS Windows installer](#).

You may download other distributions of Groovy from [this site](#).

If you prefer to live on the bleeding edge, you can also grab the [source code from SVN](#).

If you are an IDE user, you can just grab the latest [IDE plugin](#) and follow the plugin installation instructions.

## InvokeDynamic support

### Foreword

Since Groovy 2.0, we added support for the JVM [invokedynamic](#) instruction. This instruction is supported since Java 7 and is a new bytecode instruction in the JVM that allows easier implementation of dynamic languages. This instruction will also be used internally, by the JVM, for the upcoming lambda support in Java 8.

This means that unlike APIs, AST transformations or syntactic sugar, this feature is **not visible** to the developer or the end user. It is a compilation and runtime feature only. This means that given two programs written in Groovy, you have the choice to compile it with or without invokedynamic support. Whatever you choose, it comes with pros and cons:

- classes compiled with invokedynamic can only be used on JDK 1.7+
  - without invokedynamic, Groovy classes are still compatible with JDK 1.5+
- call site caching, as implemented in "normal" Groovy is replaced with invokedynamic since Groovy 2.1
- it is possible to mix classes compiled with and without invokedynamic in the same project, as long as you run JDK 1.7+
- depending on the JVM (even different minor versions of the JVM), you can target close to Java performance for dynamic Groovy with invokedynamic support activated

### The distributions

#### *Two jars*

The Groovy distribution comes with **two** jars:

- groovy-x.y.z.jar : compatible with JDK 1.5+, makes use of call site caching
- groovy-x-y-z-indy.jar : compatible with JDK 1.7+ only, has invokedynamic support bundled, old call site caching still possible

The first jar is Groovy compiled without invokedynamic support, while the second one has invokedynamic support bundled. As Groovy core and the groovy modules are sometimes written in Groovy, we currently have no choice but issuing two distinct versions of Groovy. This means that if you pick the "normal" jar, the groovy classes of groovy itself are compiled with call site caching (1.5+), while if you use the "indy" jar, the groovy classes of groovy itself are compiled using invokedynamic. This means that the invokedynamic version of Groovy doesn't make use of the old call site caching mechanism.

Both jars contain a fully working groovy implementation. They are mutually exclusive (don't put both on classpath).

## Command-line and indy

If you download the distribution and that you use the command line, it's always the "normal" version of Groovy which is picked up in classpath. This means that whatever command you use (*groovy*, *groovyc*, *groovysh* or *groovyConsole*), invokedynamic support is not available out of the box. To use the invokedynamic version, you have to switch the jars manually. The distribution makes use of the jars in the *lib* directory, while the indy jars are available in the *indy* directory. You have three things to do:

- remove or rename the groovy-\*.jar files in the lib directory
- replace them with the indy version from the indy directory
- remove the -indy classifier from jar names

## Here's a bash script that would do it all at once:

```
$ for f in `ls lib/groovy*.jar | cut -d/
-f2`;do k=`basename $f .jar`; mv lib/$k.jar
lib/$k.jar.old; cp indy/$k-indy.jar
lib/$k.jar ; done
```

### The compilation flag

Independently of the jar version that you use (and after having exchanged the jars as described), invokedynamic support requires a specific compilation flag (*indy*). If you want to compile your classes with invokedynamic support, this flag must be set at compile time. The following tables show you what happens with user compiled classes and groovy core classes depending on the jar you use and the compilation flag:

indy flag	off	on
normal jar	call site caching	N/A
indy jar	call site caching	invokedynamic

#### *user compiled classes*

indy flag	off	on
normal jar	call site caching	N/A
indy jar	invokedynamic	invokedynamic

#### *core groovy classes*

So even if you use the indy jar, if you don't use the invokedynamic flag at compile time, then the compiled classes will use the "old" format, meaning they will use the JDK1.5+ classes without invokedynamic.

## Feature Overview

- [Groovlets](#)
- [Groovy Beans](#)
- [GroovyMarkup](#)
- [Groovy Templates](#)

### Groovlets

You can write normal Java servlets in Groovy (i.e. Groovlets).

There is also a [GroovyServlet](#)

This feature will automatically compile your .groovy source files, turn them into bytecode, load the Class and cache it until you change the source file.

Here's a simple example to show you the kind of thing you can do from a Groovlet.

Notice the use of implicit variables to access the session, output & request. Also notice that this is more like a *script* as it doesn't have a class wrapper.

```
if (!session) {
    session = request.getSession(true);
}

if (!session.counter) {
    session.counter = 1
}

println """
<html>
    <head>
        <title>Groovy Servlet</title>
    </head>
    <body>
Hello, ${request.remoteHost}:
${session.counter}! ${new Date()}
    </body>
</html>
"""
    session.counter = session.counter + 1
```

Or, do the same thing using MarkupBuilder:

```

if (!session) {
    session = request.getSession(true)
}

if (!session.counter) {
    session.counter = 1
}

html.html { // html is implicitly bound
to new MarkupBuilder(out)
    head {
        title("Groovy Servlet")
    }
    body {
        p("Hello, ${request.remoteHost}:
${session.counter}! ${new Date()}")
    }
}
session.counter = session.counter + 1

```

### *Implicit variables*

The following variables are ready for use in Groovlets:

variable name	bound to	note
request	ServletRequest	-
response	ServletResponse	-
context	ServletContext	unlike Struts
application	ServletContext	unlike Struts

session	getSession(false)	can be null! see ★ A
params		a Map object
headers		a Map object
out	response.getWriter()	see ★ B
sout	response.getOutputStream()	see ★ B
html	new MarkupBuilder(out)	see ★ B
json	new StreamingJsonBuilder(out)	see ★ B

★ A The session variable is only set, if there was already a session object. See the 'if (session == null)' checks in the examples above.

★ B These variables cannot be re-assigned inside a Groovlet. They are bound on first access, allowing to e.g. calling methods on the 'response' object before using 'out'.

#### **Setting up groovylets**

Put the following in your web.xml:

```

<servlet>
<servlet-name>Groovy</servlet-name>
<servlet-class>groovy.servlet.GroovyServlet<
/servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>Groovy</servlet-name>
<url-pattern>*.groovy</url-pattern>
</servlet-mapping>

```

Then all the groovy jar files into WEB-INF/lib. You should only need to put the **groovy.jar**, the **antlr.jar** and the **asm.jar**. Or copy the **groovy-all-xyz.jar** into WEB-INF/lib - this *almost* all jar contains the antlr and asm jars. Now put the .groovy files in, say, the root directory (i.e. where you would put your html files). The groovy servlet takes care of compiling the .groovy files.

So for example using tomcat you could edit tomcat/conf/server.xml like so:

```
<Context path="/groovy"  
docBase="c:/groovy-servlet"/>
```

Then access it with <http://localhost:8080/groovy/hello.groovy>

## Groovy Beans

GroovyBeans are JavaBeans but using a much simpler syntax.  
Here's an example:

```
class Customer {  
    // properties  
    Integer id  
    String name  
    Date dob  
  
    // sample code  
    static void main(args) {  
        def customer = new Customer(id:1,  
name:"Gromit", dob:new Date())  
        println("Hello ${customer.name}")  
    }  
}
```

**Hello Gromit**

Notice how the properties look just like public fields. You can also set named properties in a bean constructor in Groovy. In Groovy, fields and properties have been merged so that they act and look the same. So, the Groovy code above is equivalent to the following Java code:

```
import java.util.Date;
```

```
public class Customer {
    // properties
    private Integer id;
    private String name;
    private Date dob;

    public Integer getId() {
        return this.id;
    }

    public String getName() {
        return this.name;
    }

    public Date getDob() {
        return this.dob;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setDob(Date dob) {
        this.dob = dob;
    }

    // sample code
```

```
public static void main(String[] args) {  
    Customer customer = new Customer();  
    customer.setId(1);  
    customer.setName("Gromit");  
    customer.setDob(new Date());  
  
    System.out.println("Hello " +
```

```
customer.getName() );  
    }  
}
```

### **Property and field rules**

When Groovy is compiled to bytecode, the following rules are used.

- If the name is declared with an access modifier (public, private or protected) then a field is generated.
- A name declared with no access modifier generates a private field with public getter and setter (i.e. a property).
- If a property is declared final the private field is created final and no setter is generated.
- You can declare a property and also declare your own getter or setter.
- You can declare a property and a field of the same name, the property will use that field then.
- If you want a private or protected property you have to provide your own getter and setter which must be declared private or protected.
- If you access a property from within the class the property is defined in at compile time with implicit or explicit this (for example this.foo, or simply foo), Groovy will access the field directly instead of going through the getter and setter.
- If you access a property that does not exist using the explicit or implicit foo, then Groovy will access the property through the meta class, which may fail at runtime.

So, for example, you could create a read only property or a public read-only property with a protected setter like this:

```
class Foo {  
    // read only property  
    final String name = "John"  
  
    // read only property with public getter  
and protected setter  
    Integer amount  
    protected void setAmount(Integer amount)  
{ this.amount = amount }  
  
    // dynamically typed property  
    def cheese  
}
```

Note that properties need *some* kind of identifier: e.g. a variable type ("String") or untyped using the "def" keyword.

Why a field with public access modifier do not have getter and setter generated? If we'd generate getter / setter all the time, it means Groovy would not let you not define getters / setters, which can be problematic when you really don't want to geters / setters to be exposed.

### ***Closures and listeners***

Though Groovy doesn't support anonymous inner classes, it is possible to define action listeners inline through the means of closures. So instead of writing in Java:

```
Processor deviceProc = ...  
deviceProc.addControllerListener(new  
ControllerListener() {  
    public void  
controllerUpdate(ControllerEvent ce) {  
        ...  
    }  
}
```

You can do that in Groovy with a closure:

```
// Add a closure for a particular method on  
the listener interface  
deviceProc.controllerUpdate = { ce ->  
println "I was just called with event $ce" }
```

Notice how the closure is for a *method* on the listener interface (`controllerUpdate`), and *not for the interface itself* (`ControllerListener`). This technique means that Groovy's listener closures are used like a `ListenerAdapter` where only one method of interest is overridden. Beware: mistakenly misspelling the method name to override or using the interface name instead can be tricky to catch, because Groovy's parser may see this as a property assignment rather than a closure for an event listener.

This mechanism is heavily used in the Swing builder to define event listeners for various components and listeners. The JavaBeans introspector is used to make event listener methods available as properties which can be set with a closure.

The Java Beans introspector (`java.beans.Introspector`) which will look for a `BeanInfo` for your bean or create one using its own naming conventions. (See the Java Beans spec for details of the naming conventions it uses if you don't provide your own `BeanInfo` class). We're not performing any naming conventions ourselves - the standard Java Bean introspector does that for us.

Basically the BeanInfo is retrieved for a bean and its [EventSetDescriptors](#) are exposed as properties (assuming there is no clash with regular beans). It's actually the [EventSetDescriptor.getListenerMethods\(\)](#) which is exposed as a writable property which can be assigned to a closure.

## GroovyMarkup

**Note:** the following examples are snippets, not ready-to-run examples.

Groovy has native support for various markup languages from XML, HTML, SAX, W3C DOM, Ant tasks, Swing user interfaces and so forth.

This is all accomplished via the following syntax...

```
def someBuilder = new NodeBuilder()

someBuilder.people(kind: 'folks',
groovy:true) {
    person(x:123, name: 'James',
cheese: 'edam') {
        project(name: 'groovy')
        project(name: 'geronimo')
    }
    person(x:234, name: 'bob',
cheese: 'cheddar') {
        project(name: 'groovy')
        project(name: 'drools')
    }
}
```

Whichever kind of builder object is used, the syntax is the same. What the above means is that the someBuilder object has a method called 'people' invoked with 2 parameters...

- a Map of arguments ['kind':'folks', 'groovy':true]
- a Closure object which when invoked will call 2 methods on the builder called 'person', each taking 2 parameters, a map of values and a closure...

So we can easily represent any arbitrary nested markup with ease using a simple concise syntax. No pointy brackets! 😊

What's more is this is native Groovy syntax; so you can mix and match this markup syntax with any other Groovy features (iteration, branching, method calls, variables, expressions etc). e.g.

```

// lets create a form with a label & text
field for each property of a bean
def swing = new SwingBuilder()
def widget = swing.frame(title:'My Frame',
defaultCloseOperation:javax.swing.WindowCons
tants.EXIT_ON_CLOSE) {
    panel() {
        for (entry in someBean) {
            label(text:entry.key)
            textField(text:entry.value)
        }
        button(text:'OK', actionPerformed:{
println("I've been clicked with event
${it}") })
    }
}
widget.show()

```

### *Trees, DOMs, beans and event processing*

The really neat thing about GroovyMarkup is that its just a syntax which maps down to method calls. So it can easily support the building of any arbitrary object structure - so it can build any DOMish model, a bean structure, JMX MBeans, PicoComponents, Swing front ends, Ant tasks etc. What's more since its just normal method invocations it can naturally map to SAX event processing too.

Out of the box Groovy comes with a few different markup builders you can use :

- NodeBuilder - creates a tree of Node instances which can be easily navigated in Groovy using an XPath-like syntax
- DOMBuilder - creates a W3C DOM document from the markup its given
- SAXBuilder - fires SAX events into a given SAX ContentHandler
- MarkupBuilder - outputs XML / HTML markup to some PrintWriter for things like implementing servlets or code generation
- AntBuilder - fires off Ant tasks using familiar markup for processing build tasks
- SwingBuilder - creates rich Swing user interfaces using a simple markup

### *Examples*

Here's a simple example which shows how you could iterate through some SQL result set and output a dynamic

XML document containing the results in a custom format using GroovyMarkup

```
// lets output some XML builder (could be
SAX / DOM / TrAX / text)
def xml = new NodeBuilder()
xml.customers() {
    loc = 'London'
    sql.eachRow("select * from customer
where location = ${loc}") {

        // lets process each row by emitting
some markup
        xml.customer(id:it.id,
type:'Customer', foo:someVariable)) {
            role(it.person_role)
            name(it.customer_name)
            location(id:it.location_id,
name:it.location_name)
        }
    }
}
```

The interesting thing about the above is that the XML technology used at the other end could be push-event based (SAX) or pull-event based (StAX) or a DOM-ish API (W3C, dom4j, JDOM, EXML, XOM) or some JAXB-ish thing (XMLBeans, Castor) or just beans or just good old text files.

e.g. a pull parser could literally pull the data out of the database - or the data could be pushed into data some structure or piped straight to a file using IO or async NIO.

The use of GroovyMarkup means developers can hide the XML plumbing and focus on tackling the real problems we're trying to solve.

To see more examples of using GroovyMarkup try looking at our unit test cases

- [XML unit tests](#)
- [Ant unit tests](#)
- [Swing demos](#)

There is more detail on markup here [Make a builder](#).

## Groovy Templates

### *Introduction*

Groovy supports multiple ways to generate text dynamically including [GStrings](#), `printf` if you are using Java 5, and [MarkupBuilder](#) just to name a few. In addition to these, there is a dedicated template framework which is well-suited to applications where the text to be generated follows the form of a static template.

### *Template framework*

The template framework in Groovy consists of a `TemplateEngine` abstract base class that engines must implement and a `Template` interface that the resulting templates they generate must implement.

Included with Groovy are several template engines:

- `SimpleTemplateEngine` - for basic templates
- `GStringTemplateEngine` - stores the template as writable closures (useful for streaming scenarios)
- `XmlTemplateEngine` - works well when the template and output are valid XML

### *SimpleTemplateEngine*

Shown here is the `SimpleTemplateEngine` that allows you to use JSP-like scriptlets (see example below), `script`, and EL expressions in your template in order to generate parameterized text. Here is an example of using the system:

```

import groovy.text.SimpleTemplateEngine

def text = 'Dear "$firstname $lastname",\nSo
nice to meet you in <% print city %>.\nSee
you in ${month},\n${signed}'

def binding = ["firstname":"Sam",
"lastname":"Pullara", "city":"San
Francisco", "month":"December",
"signed":"Groovy-Dev"]

def engine = new SimpleTemplateEngine()
template =
engine.createTemplate(text).make(binding)

def result = 'Dear "Sam Pullara",\nSo nice
to meet you in San Francisco.\nSee you in
December,\nGroovy-Dev'

assert result == template.toString()

```

While it is generally not deemed good practice to mix processing logic in your template (or view), sometimes very simple logic can be useful. E.g. in the example above, we could change this:

```
$firstname
```

to this (assuming we have set up a static import for `capitalize` inside the template):

```
${capitalize(firstname)}
```

or this:

```
<% print city %>
```

to this:

```
<% print city == "New York" ? "The Big  
Apple" : city %>
```

**Advanced Usage Note**

If you happen to be embedding your template directly in your script (as we did above) you have to be careful about backslash escaping. Because the template string itself will be parsed by Groovy before it is passed to the templating framework, you have to escape any backslashes inside GString expressions or scriptlet 'code' that are entered as part of a Groovy program. E.g. if we wanted quotes around *The Big Apple* above, we would use:

```
<% print city == "New York" ? "\\\"The Big  
Apple\\\"" : city %>
```

Similarly, if we wanted a newline, we would use:

```
\\n
```

in any GString expression or scriptlet 'code' that appears inside a Groovy script. A normal "\n" is fine within the static template text itself or if the entire template itself is in an external template file. Similarly, to represent an actual backslash in your text you would need

in an external file or

in any GString expression or scriptlet 'code'. (Note: the necessity to have this extra slash may go away in a future

version of Groovy if we can find an easy way to support such a change.)

### *GStringTemplateEngine*

As an example of using the `GStringTemplateEngine`, here is the example above done again (with a few changes to show some other options). First we will store the template in a file this time:

#### **test.template**

```
Dear "$firstname $lastname",
So nice to meet you in <% out << (city ==
"New York" ? "\"The Big Apple\"" : city) %>.
See you in ${month},
${signed}
```

Note that we used `out` instead of `print` to support the streaming nature of `GStringTemplateEngine`. Because we have the template in a separate file, there is no need to escape the backslashes. Here is how we call it:

```
def f = new File('test.template')
engine = new GStringTemplateEngine()
template =
engine.createTemplate(f).make(binding)
println template.toString()
```

and here is the output:

```
Dear "Sam Pullara",
So nice to meet you in "The Big Apple".
See you in December,
Groovy-Dev
```

You can also plug in other templating solutions, e.g. [GFreeMarker](#), [Velocity](#), [StringTemplate](#), [Canvas](#) and others.

If you wish to combine templating with Ant processing, consider [Gpp](#).

*Using TemplateServlet to serve single JSP-like HTML files*

**⚠ Mind the gap! Ehm, meaning the difference between Groovlets and Templates.**

The [TemplateServlet](#) just works the opposite as the [Groovlets\(GroovyServlet\)](#) does. Here, your source is HTML (or any other, fancy template files) and the template framework will generate a Groovy script on-the-fly. This script could be saved to a `.groovy` file and served by the GroovyServlet (and the GroovyScriptEngine), but after generation, the template is evaluated and responded to the client.

Here is a simple example `helloworld.html` file which is not validating and does not have a head element. But it demonstrates, how to let Groovy compile and serve your HTML files to web clients. The tag syntax close to JSP and should be easy to read:

```
<html>
  <body>
    <% 3.times { %>
      Hello World!
    <% } %>
    <br>
    <% if (session != null) { %>
      My session id is ${session.id}
    <% } else println "No session created."
  %>
  </body>
</html>
```

The first Groovy block - a for loop - spans the `HelloWorld!` text. Guess what happens? And the second Groovy statement prints the servlet's session id - if there is a session available. The variable `session` is one of some default bound keys. More details reveals the documentation of [ServletBinding](#).

Here is some sample code using <http://www.eclipse.org/jetty/s> servlet container. With Jetty, dependencies added through Grape, create a tiny web server with the following. To test it, add your above `helloworld.html` file into your current directory and browse <http://localhost:1234/helloworld.html>

```

@Grapes ( [

@Grab(group='org.eclipse.jetty.aggregate',
module='jetty-server',
version='8.1.7.v20120910'),

@Grab(group='org.eclipse.jetty.aggregate',
module='jetty-servlet',
version='8.1.7.v20120910'),
    @Grab(group='javax.servlet',
module='javax.servlet-api',
version='3.0.1')] )

import org.eclipse.jetty.server.Server
import org.eclipse.jetty.servlet.*
import groovy.servlet.*

def server = new Server(1234)
def context = new
ServletContextHandler(server, "/",
ServletContextHandler.SESSIONS)
context.resourceBase = "."
context.addServlet(TemplateServlet,
"* .html")
server.start()

```

Here is a similiar `web.xml` example.

```
<web-app>
```

```
<servlet>
  <servlet-name>Groovlet</servlet-name>

<servlet-class>groovy.servlet.GroovyServlet<
/servlet-class>
  </servlet>
<servlet>

<servlet-name>Template</servlet-name>

<servlet-class>groovy.servlet.TemplateServle
t</servlet-class>
  </servlet>

  <servlet-mapping>

<servlet-name>Groovlet</servlet-name>
  <url-pattern>*.groovy</url-pattern>
</servlet-mapping>
<servlet-mapping>

<servlet-name>Template</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>
<servlet-mapping>

<servlet-name>Template</servlet-name>
  <url-pattern>*.gsp</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

#### Further reading

[Article on templating with Groovy templates](#)

[Article on templating with Groovlets and TemplateServlets](#)

[Blog about combining Groovy and FreeMarker](#)

## Quick Start

### Before beginning...

Before playing with the examples you'll find below, you should first look at:

- [Installing Groovy](#)
- [Running](#)

Some optional more advanced topics you may also wish to peruse:

- [Command Line](#) : Groovy can be launched in shell script mode
- [Compiling Groovy](#) : Groovy can be launched as any Java program
- [Embedding Groovy](#) : embedding Groovy in Java code using built-in capabilities
- [JSR 223 Scripting with Groovy](#) : embedding Groovy in Java code using JSR 223
- [Bean Scripting Framework](#) : embedding Groovy in Java code using the BSF

### Your First Groovy

```
//hello.groovy
println "hello, world"
for (arg in this.args ) {
    println "Argument:" + arg;
}
// this is a comment
/* a block comment, commenting out an
alternative to above:
this.args.each{ arg -> println "hello,
${arg}"}
*/
```

To run it from command line

```
groovy hello.groovy MyName yourName HisName
```

## Overview

Groovy classes compile down to Java bytecode and so there's a 1-1 mapping between a Groovy class and a Java class.

Indeed each Groovy class can be used inside normal Java code - since it is a Java class too.

Probably the easiest way to get groovy is to try working with collections. In Groovy List (`java.util.List`) and Map (`java.util.Map`) are both first class objects in the syntax. So to create a List of objects you can do the following...

```
def list = [1, 2, 'hello', new  
java.util.Date()]  
assert list.size() == 4  
assert list.get(2) == 'hello'  
assert list[2] == 'hello'
```

Notice that everything is an object (or that auto-boxing takes place when working with numbers). To create maps...

```
def map = ['name': 'James',  
'location': 'London']  
assert map.size() == 2  
assert map.get('name') == 'James'  
assert map['name'] == 'James'
```

Iterating over collections is easy...

```
def list = [1, 2, 3]  
for (i in list) { println i }
```

Once you have some collections you can then use some of the new collection helper methods or try working with closures...

## Working with closures

Closures are similar to Java's inner classes, except they are a single method which is invocable, with arbitrary parameters. A closure can have as many parameters as you wish...

```
def closure = { param -> println("hello
${param}") }
closure.call("world!")

closure = { greeting, name ->
println(greeting + name) }
closure.call("hello ", "world!")
```

If no parameter(s) is(are) specified before -> symbol then a default named parameter, called 'it' can be used. e.g.

```
def closure = { println "hello " + it }
closure.call("world!")
```

Using closures allows us to process collections (arrays, maps, strings, files, SQL connections and so forth) in a clean way. e.g

```
[1, 2, 3].each ({ item -> print "${item}-"
})
["k1":"v1", "k2":"v2"].each {key, value ->
println key + "=" + value}
```

**Note:** If a given closure is the last parameter of a method, its definition can reside outside of the parentheses. Thus the following code is valid:

```

def fun(int i, Closure c) {
  c.call(i)
}

// put Closure out of ()
[1, 2, 3].each() { item -> print "${item}-"
} // 1-2-3-
fun(123) { i -> println i } // 123

// omit ()
[1, 2, 3].each ({ item -> print "${item}-"
}) // 1-2-3-

// omit enclosing ()
[1, 2, 3].each { item -> print "${item}-" }
// 1-2-3-

// normal
[1, 2, 3].each(({ item -> print "${item}-"
})) // 1-2-3-

// using the fun function to do the same
thing
[1,2,3].each {fun(it,{item -> print
"${item}-"})} // 1-2-3-

def closure = { i -> println i}

//[1, 2, 3].each() closure // error. closure
has been previously defined

```

Here are a number of helper methods available on collections & strings...

### *each*

iterate via a closure

```
[1, 2, 3].each { item -> print "${item}-" }
```

### *collect*

collect the return value of calling a closure on each item in a collection

```
def value = [1, 2, 3].collect { it * 2 }  
assert value == [2, 4, 6]
```

### *find*

finds first item matching closure predicate

```
def value = [1, 2, 3].find { it > 1 }  
assert value == 2
```

### *findAll*

finds all items matching closure predicate

```
def value = [1, 2, 3].findAll { it > 1 }  
assert value == [2, 3]
```

### *inject*

allows you to pass a value into the first iteration and then pass the result of that iteration into the next iteration and so on. This is ideal for counting and other forms of processing

```
def value = [1, 2, 3].inject('counting: ') {  
  str, item -> str + item }  
assert value == "counting: 123"  
  
value = [1, 2, 3].inject(0) { count, item ->  
  count + item }  
assert value == 6
```

In addition there's 2 new methods for doing boolean logic on some collection...

#### *every*

returns true if all items match the closure predicate

```
def value = [1, 2, 3].every { it < 5 }  
assert value  
  
value = [1, 2, 3].every { item -> item < 3 }  
assert ! value
```

#### *any*

returns true if any item match the closure predicate

```
def value = [1, 2, 3].any { it > 2 }  
assert value  
  
value = [1, 2, 3].any { item -> item > 3 }  
assert value == false
```

Other helper methods include:

#### *max / min*

returns the max/min values of the collection - for Comparable objects

```
value = [9, 4, 2, 10, 5].max()
assert value == 10
value = [9, 4, 2, 10, 5].min()
assert value == 2
value = ['x', 'y', 'a', 'z'].min()
assert value == 'a'
```

*join*

concatenates the values of the collection together with a string value

```
def value = [1, 2, 3].join('-')
assert value == '1-2-3'
```

## Installing Groovy

These instructions describe how to install a binary distribution of **Groovy**.

- first, [Download](#) a binary distribution of Groovy and unpack it into some file on your local file system
- set your `GROOVY_HOME` environment variable to the directory you unpacked the distribution
- add `GROOVY_HOME/bin` to your `PATH` environment variable
- set your `JAVA_HOME` environment variable to point to your JDK. On OS X this is `/Library/Java/Home`, on other unixes its often `/usr/java` etc. If you've already installed tools like Ant or Maven you've probably already done this step.

You should now have Groovy installed properly. You can test this by typing the following in a command shell:

```
groovysh
```

Which should create an interactive groovy shell where you can type Groovy statements. Or to run the [Swing interactive console](#) type:

```
groovyConsole
```

To run a specific Groovy script type:

# groovy SomeScript.groovy

## Alternative Installation on Mac OS X

Set up [MacPorts](#)

Open Terminal

```
sudo port install groovy
```

## Installing Groovy and Grails on the Eee PC

Have an [ASUS Eee PC](#)?

Worried that framework bloat might cramp your development practices on such a small device? Why not run Groovy and Grails on it! Shown here running Linux, several command shells, a Firefox browser, the Groovy Console, a database, a web container and Grails (which itself includes Hibernate and Spring) all in 512M of memory. And of course it has no (traditional) hard disk, so the whole footprint of Groovy, Grails and Java 6 is just a few hundred meg of the available flash memory.



### Installing Java

- Given the relatively humble processor in the Eee PC, you probably want to use Java 6. Install it as per these [instructions](#). This is a flash memory vs speed trade-off.

### Installing Groovy

- Then [download](#) and install the Groovy linux distribution. Ctrl-Alt-T will start up a terminal, then type:

```
sudo dpkg -i groovy-1.5.1.deb
```

### Installing Grails

- You might also want to grab [Grails](#) and install that too by unzipping it or installing the deb from [here](#).

### Going further

- Google and you will find instructions for installing Eclipse and other applications too.

## Running

Groovy scripts are a number of statements and class declarations in a text file. Groovy scripts can be used similarly to other scripting languages. There are various ways of running Groovy scripts

### Using the interactive console

Groovy has a Swing interactive console that allows you to type in commands and execute them rather like using an SQL query tool. History is available and such like so you can move forwards and backwards through commands etc.

If you [install](#) a binary distribution of Groovy then you can run the Groovy Swing console by typing this on the command line.

```
groovyConsole
```

For a command line interactive shell type

```
groovysh
```

To see how to add things to the classpath see below.

#### *Running Groovy scripts from your IDE*

There is a helper class called [GroovyShell](#) which has a main(String[]) method for running any Groovy script. You can run any groovy script as follows

```
java groovy.lang.GroovyShell  
foo/MyScript.groovy [arguments]
```

You can then run the above Groovy main() in your IDE to run or debug any Groovy script.

#### *Running Groovy scripts from the command line*

There are shell scripts called 'groovy' or 'groovy.bat' depending on your platform which is part of the Groovy runtime. Once the runtime is [installed](#) you can just run groovy like any other script...

```
groovy foo/MyScript.groovy [arguments]
```

If you are using Groovy built from CVS Head (after Beta-5, see below if you want to upgrade), apart from Groovy scripts, you may also now run different kind of classes from the command-line.

- Classes with a main method of course,
- Classes extending GroovyTestCase are run with JUnit's test runner,
- Classes implementing the Runnable interface are instantiated either with a constructor with String[] as argument, or with a no-args constructor, then their run() method is called.

To work from the latest and greatest Groovy see [Building Groovy from Source](#). Once built you'll then have a full

binary distribution made for you in `groovy/target/install`. You can then add `groovy/target/install/bin` to your path and you can then run groovy scripts easily from the command line.

To see how to add things to the classpath see below.

### ***Creating Unix scripts with Groovy***

You can write unix scripts with Groovy and execute them directly on the command line as if they were normal unix shell scripts. Providing you have installed the Groovy binary distribution (see above) and 'groovy' is on your PATH then the following should work.

The following is a sample script, which you should copy and save as `helloWorld.groovy`.

```
#!/usr/bin/env groovy
println("Hello world")
for (a in this.args) {
    println("Argument: " + a)
}
```

Then to run the script from the command line, just make sure the script is executable then you can call it.

```
chmod +x helloWorld
./helloWorld
```

Note that you will not be able to provided any arguments to groovy when running with the shebang(#!) on Linux because the arguments are treated as part of the name of the command to run. Mac OS X will interpret additional arguments correctly.

### ***Adding things to the classpath***

When running command line scripts or interactive shells you might want to add things to your classpath such as JDBC drivers or JMS implementations etc. To do this, you have a few choices:

- Add things to your CLASSPATH environment variable
- Pass `-classpath` (or `-cp`) into the command you used to create the shell or run the script
- It's also possible to create a `~/.groovy/lib` directory and add whatever jars you need in there.
- If the jars you need are in a Maven or Ivy repository, you can "grab" them with [Grape](#).

### ***Increasing Groovy's JVM Heap Size***

To increase the amount of memory allocated to your groovy scripts, set your JAVA\_OPTS environment variable. `JAVA_OPTS="-Xmx..."`

## ***Avoiding PermGen out of memory***

Groovy creates classes dynamically, but the default Java VM does not GC the PermGen. If you are using Java 6 or later, add `-XX:+CMSClassUnloadingEnabled -XX:+UseConcMarkSweepGC`. `UseConcMarkSweepGC` is needed to enable `CMSClassUnloadingEnabled`.

## **Beginners Tutorial**

Welcome on board the Groovy flight. Before proceeding through the content of this tutorial, please make sure to fasten your seat belt, before we take off to higher levels of grooviness...

This page is intended to get you started with Groovy, following a trail of a few tutorial labs on various topics mainly oriented towards typical use of scripting languages for data crunching or text manipulation.

Graham Miller, a Groovy aficionado, has been teaching a class of business on data crunching. And he was kind enough to contribute back to the Groovy project this great set of educational material to help you learn Groovy, using some nice examples to massage, summarize and analyze data - a task for which Groovy is a quite good fit.

The topics covered are about Groovy basics, text parsing, regular expressions, and SQL:

- [Getting started](#)
- [Code as data](#)
- [Classes and Objects](#)
- [Regular Expressions](#)
- [Capturing groups](#)
- [Groovy SQL](#)

### **If you are a Java developer**

- you might want to check on the [Differences from Java](#)
- also there are a few [Things to remember](#)

## **Tutorial 1 - Getting started**

### **Getting Started**

#### ***Setting up your Java environment***

Groovy requires Java, so you need to have a version available (while groovy 1.6 supported JDK 1.4 or greater, for groovy 1.7 onwards, minimum JDK 1.5 is needed). Here are the steps if you don't already have Java installed:

- Get the suitable Java distribution from the <http://java.sun.com> website.
- Run the installer.
- Set the JAVA\_HOME environment variables. On Windows, follow these steps:
  - Open the System control panel
  - Click the Advanced tab
  - Click the Environment Variables button
  - Add a new System variable with the name JAVA\_HOME and the value of the directory Java was installed in (mine is C:\Program Files\Java\jdk1.5.0\_04)
  - Optionally add %JAVA\_HOME%\bin to your system path  
(Note: as an alternative to setting a system environment variable, you can create yourself a '.bat' or '.cmd' file which sets the variable. You then need to run that batch file in any console window in which you wish to run Java and double clicking on '.bat' or '.cmd' files containing Java invocation instructions won't work. If you are unsure about what this means, follow the earlier instructions.)

#### ***Setting up your Groovy environment***

Download the Groovy installer or binaries from the [downloads](#) page and follow the installation instructions. (There is currently an issue where you cannot have spaces in the path where Groovy is installed under windows. So, instead of accepting the default installation path of "c:\Program Files\Groovy" you will want to change the path to something like "c:\Groovy")

OR

- Get a copy of the Groovy distribution from the [website](#), and copy it to some place on your hard drive.
- Unzip the groovy archive to some logical place on your hard drive, I have mine in C:\dev\groovy-2.0.5
- Set the GROOVY\_HOME environment variables. On Windows, follow these steps:
  - Optionally add a new System variable with the name GROOVY\_HOME and the value of the directory groovy was installed in (mine is C:\dev\groovy-2.0.5)
  - Start a command prompt, and type "set" and hit return to see that your environment variables were set correctly.
- Optionally add %GROOVY\_HOME%\bin to your system path
- Try opening groovyConsole.bat by double clicking on the icon in the bin directory of the Groovy distribution. If it doesn't work, open a command prompt, and change to the bin directory and run it from there to see what the error message is. If it is complaining about not finding parts of Groovy, then setting GROOVY\_HOME may help,

#### **Setting up optional jar files**

You may wish to obtain optional jar files, either corresponding to Groovy modules (see module documentation for details) or corresponding to other Java classes you wish to make use of from Groovy. Some possibilities are listed below:

Name	From	Description
jtids- <i>version</i> .jar	<a href="http://jtids.sourceforge.net">http://jtids.sourceforge.net</a>	Database driver for SQL Server and/or Sybase
hsqldb- <i>version</i> .jar	<a href="http://www.hsqldb.org/">http://www.hsqldb.org/</a>	Database driver for HSQLDB, a 100% Java database

The recommended way for making Groovy be aware of your additional jar files is to place them in a predefined location. Your Groovy install should include a file called `groovy-starter.conf`. Within that file, make sure a line such as

```
load ${user.home}/.groovy/lib/*
```

is not commented out. The `user.home` system property is set by your operating system. (Mine is C:\Document and Settings\paul.

Now simply place your jar files into the `.groovy/lib` directory. (Note: as an alternative, you can set up a CLASSPATH variable and make sure it mentions all of your additional jar files, otherwise Groovy works fine with an empty or no CLASSPATH variable.)

#### **Hello, World**

In the top part of the window of the groovyConsole, type the following

```
println "Hello, World!"
```

And then type <CTRL-R>.

Notice that the text gets printed out in the OS console window (the black one behind the groovyConsole window) and the bottom part of the groovyConsole says:

```
groovy> println "Hello, World!"  
null
```

The line starting with "groovy>" is just the text of what the console processed. The "null" is what the expression "evaluated to". Turns out the expression to print out a message doesn't have any "value" so the groovyConsole printed "null".

Next try something with an actual value. Replace the text in the console with:

```
123+45*67
```

or your favorite arithmetic expression, and then type <CTRL-R> (I'm going to stop telling you to hit <CTRL-R>, I think you get the idea). Now the "value" printed at the bottom of the groovyConsole has more meaning.

### ***Variables***

You can assign values to variables for later use. Try the following:

```
x = 1
println x

x = new java.util.Date()
println x

x = -3.1499392
println x

x = false
println x

x = "Hi"
println x
```

### *Lists and Maps*

The Groovy language has built-in support for two important data types, lists and maps (Lists can be operated as arrays in Java language). Lists are used to store ordered collections of data. For example an integer list of your favorite integers might look like this:

```
myList = [1776, -1, 33, 99, 0, 928734928763]
```

You can access a given item in the list with square bracket notation (indexes start at 0):

```
println myList[0]
```

Should result in this output:

```
1776
```

You can get the length of the list with the "size" method:

```
println myList.size()
```

Should print out:

```
6
```

But generally you shouldn't need the length, because unlike Java, the preferred method to loop over all the elements in a list is to use the "each" method, which is described below in the "Code as Data" section.

Another native data structure is called a map. A map is used to store "associative arrays" or "dictionaries". That is unordered collections of heterogeneous, named data. For example, let's say we wanted to store names with IQ scores we might have:

```
scores = [ "Brett":100, "Pete":"Did not  
finish", "Andrew":86.87934 ]
```

Note that each of the values stored in the map is of a different type. Brett's is an integer, Pete's is a string, and Andrew's is a floating point number. We can access the values in a map in two main ways:

```
println scores["Pete"]  
println scores.Pete
```

Should produce the output:

```
Did not finish  
Did not finish
```

To add data to a map, the syntax is similar to adding values to an list. For example, if Pete re-took the IQ test and got a 3, we might:

```
scores["Pete"] = 3
```

Then later when we get the value back out, it will be 3.

```
println scores["Pete"]
```

should print out 3.

Also as an aside, you can create an empty map or an empty list with the following:

```
emptyMap = [:]  
emptyList = []
```

To make sure the lists are empty, you can run the following lines:

```
println emptyMap.size()  
println emptyList.size()
```

Should print a size of 0 for the List and the Map.

### ***Conditional Execution***

One of the most important features of any programming language is the ability to execute different code under different conditions. The simplest way to do this is to use the "if" construct. For example:

```
amPM =
Calendar.getInstance().get(Calendar.AM_PM)
if (amPM == Calendar.AM)
{
    println("Good morning")
} else {
    println("Good evening")
}
```

Don't worry too much about the first line, it's just some code to determine whether it is currently before noon or after. The rest of the code executes as follows: first it evaluates the expression in the parentheses, then depending on whether the result is "true" or "false" it executes the first or the second code block. See the section below on boolean expressions.

Note that the "else" block is not required, but the "then" block is:

```
amPM =
Calendar.getInstance().get(Calendar.AM_PM)
if (amPM == Calendar.AM)
{
    println("Have another cup of coffee.")
}
```

### ***Boolean Expressions***

There is a special data type in most programming languages that is used to represent truth values, "true" and "false". The simplest boolean expressions are simply those words. Boolean values can be stored in variables, just like any other data type:

```
myBooleanVariable = true
```

A more complex boolean expression uses one of the boolean operators:

```
* ==
* !=
* >
* >=
* <
* <=
```

Most of those are probably pretty intuitive. The equality operator is "==" to distinguish from the assignment operator "=". The opposite of equality is the "!=" operator, that is "not equal". So some examples:

```
titanicBoxOffice = 1234600000
titanicDirector = "James Cameron"

trueLiesBoxOffice = 219000000
trueLiesDirector = "James Cameron"

returnOfTheKingBoxOffice = 752200000
returnOfTheKingDirector = "Peter Jackson"

theTwoTowersBoxOffice = 581200000
theTwoTowersDirector = "Peter Jackson"

titanicBoxOffice > returnOfTheKingBoxOffice
// evaluates to true
titanicBoxOffice >= returnOfTheKingBoxOffice
// evaluates to true
titanicBoxOffice >= titanicBoxOffice
// evaluates to true
titanicBoxOffice > titanicBoxOffice
```

```
// evaluates to false
titanicBoxOffice + trueLiesBoxOffice <
returnOfTheKingBoxOffice +
theTwoTowersBoxOffice // evaluates to false

titanicDirector > returnOfTheKingDirector
// evaluates to false, because "J" is before
"p"
titanicDirector < returnOfTheKingDirector
// evaluates to true
titanicDirector >= "James Cameron"
```

```
// evaluates to true
titanicDirector == "James Cameron"
// evaluates to true
```

Boolean expressions are especially useful when used in conjunction with the "if" construct. For example:

```
if (titanicBoxOffice + trueLiesBoxOffice >
returnOfTheKingBoxOffice +
theTwoTowersBoxOffice)
{
    println(titanicDirector + " is a better
director than " + returnOfTheKingDirector)
}
```

An especially useful test is to test whether a variable or expression is null (has no value). For example let's say we want to see whether a given key is in a map:

```
suvMap = ["Acura MDX": "\$36,700", "Ford
Explorer": "\$26,845"]
if (suvMap["Hummer H3"] != null)
{
    println("A Hummer H3 will set you back
"+suvMap["Hummer H3"]);
}
```

Generally null is used to indicate the lack of a value in some location.

### Debugging and Troubleshooting Tips

- Print out the class of a variable that you're interested in with `myVar.getClass()`. Then look up the documentation for that class.
- If you're having trouble with a complex expression, pare it down to a simpler expression and evaluate that. Then build up to your more complex expression.

- Try restarting the groovyConsole (this will clear out all the variables so you can start over.
- Look for the topic you're interested in in the Groovy [User Guide](#)

### If you are a Java developer

- you might want to check on the [Differences from Java](#)
- also there a few [Things to remember](#)

## Tutorial 2 - Code as data, or closures

### Closures

One of the things that makes Groovy different than most compiled languages is that you can create functions that are first class objects. That is you can define a chunk of code and then pass it around as if it were a string or an integer. Check out the following code:

```
square = { it * it }
```

The curly braces around the expression "it \* it" tells the Groovy compiler to treat this expression as code. In the software world, this is called a "closure". In this case, the designator "it" refers to whatever value is given to the function. Then this compiled function is assigned to the variable "square" much like those above. So now we can do something like this:

```
square(9)
```

and get the value 81.

This is not very interesting until we find that we can pass this function "square" around as a value. There are some built in functions that take a function like this as an argument. One example is the "collect" method on arrays. Try this:

```
[ 1, 2, 3, 4 ].collect(square)
```

This expression says, create an array with the values 1,2,3 and 4, then call the "collect" method, passing in the closure we defined above. The collect method runs through each item in the array, calls the closure on the item, then puts the result in a new array, resulting in:

```
[ 1, 4, 9, 16 ]
```

For more methods you can call with closures as arguments, see the [Groovy GDK documentation](#).

By default closures take 1 parameter called "it", you can also create closures with named parameters. For example

the method `Map.each()` can take a closure with two variables, to which it binds the key and associated value:

```
printMapClosure = { key, value -> println  
key + "=" + value }  
[ "Yue" : "Wu", "Mark" : "Williams", "sudha"  
: "Kumari" ].each(printMapClosure)
```

Produces:

```
Yue=Wu  
Mark=Williams  
Sudha=Kumari
```

#### *More Closure Examples*

Here are a few more closure examples. This first one shows a couple of things. First, the closure is interacting with a variable outside itself. That is, the closure's purpose is to put together the parts of a stock order held in the array `orderParts`, by adding (appending) it to the variable `fullString`. The variable `fullString` is not in the closure. The second thing that is different about this example is that the closure is "anonymous", meaning that it is not given a name, and is defined in the place where the `each` method is called.

```
fullString = ""  
orderParts = ["BUY", 200, "Hot Dogs", "1"]  
orderParts.each {  
    fullString += it + " "  
}  
  
println fullString
```

You can probably guess what this prints out.

The next example is another anonymous closure, this time, summing up the values stored in a map.

```
myMap = [ "China": 1 , "India" : 2, "USA" :  
3]  
  
result = 0  
myMap.keySet().each( { result+= myMap[it] }  
)  
println result
```

### Dealing with Files

Reading data from files is relatively simple. First create a text file, and call it myfile.txt. It doesn't matter what's in it, just type some random text into it and save it on your C: drive in the \temp directory. Then type the following code in the groovyConsole:

```
myFileDirectory = "C:\\temp\\"  
myFileName = "myfile.txt"  
myFile = new File(myFileDirectory +  
myFileName)  
  
printFileLine = { println "File line: " + it  
}  
  
myFile.eachLine( printFileLine )
```

This should print out every line in the file prefixed with "File line: ". The first two lines of the code simply declare variables to specify where the file is located. The variable names don't have any special significance, and as you can see, all we do is combine them when we use them. Note that because the backslash character has special meaning in groovy, you have to use two of them to tell it that you "really" mean a backslash.

The next line that starts "myFile =" creates a new File object. An object is simply a collection of related methods and data. For example, a file object might have data describing its location, in this case "C:\temp\myfile.txt", and maybe a method to delete the file if it exists. In this case the only method we are going to use is the eachLine method, which we call in the last line of code. The line before that is a simple closure definition, that you have seen several times by this point.

### Dealing with strings

Strings in Groovy have all the same functionality of Java strings. That is, a Groovy string is just a Java string with a few extra things added to it. Because of that, we can refer to the [Java documentation for the String class](#) to find out some of the interesting things we can do with it. For example, look in the section entitled "Method Summary" at the description for the "split" method. This method does something very useful, which is to split a string based on a regular expression. We will talk more about regular expressions later, but for now the only thing you have to know is that the simplest regular expression is a single character. So let's say that we want to split up the components of the date "2005-07-04", so that we can add one to the year to get the date of next fourth of July. We might:

```
stringDate = "2005-07-04"
dateArray = stringDate.split("-") // split()
uses regex's, so you need to escape chars
such as a "." -> "\\."
year = dateArray[0].toInteger()
year = year + 1
newDate = year + "-" + dateArray[1] + "-" +
dateArray[2]
```

This code brings together a bunch of things we have talked about before. There are two new things, first is the use of the split method on a String. Second is the call of toInteger() on a String. This call to toInteger simply tells Groovy that you want to treat that data as a number rather than a String. See what happens if you run the same code without ".toInteger()" at the end of the third line.

Another thing you might notice is that toInteger is not listed in the Java documentation for string. That is because it is one of the extra features that Groovy has added to Strings. You can also take a look at the documentation for the [Groovy extensions to Java objects](#).

### Tutorial 3 - Classes and Objects

#### Classes and Objects

- Objects are collections of related code and data
- Everything in Java and Groovy can be considered an object
- A class is a higher level description of an object.
  - For example a 10-Q is a specification developed by the SEC and can be thought of as a "Class". A quarterly report issued by **IBM** for Q2 2005 can be thought of as an object of the class 10-Q.
- Documentation for java classes can be found [here](#)
- Documentation for Groovy extensions to Java classes can be found [here](#)

### Tutorial 4 - Regular expressions basics

#### Regular Expressions

Regular expressions are the Swiss Army knife of text processing. They provide the programmer the ability to match and extract **patterns** from strings. The simplest example of a regular expression is a string of letters and numbers. And the simplest expression involving a regular expression uses the `==~` operator. So for example to match Dan

Quayle's spelling of 'potato':

```
"potatoe" ==~ /potatoe/
```

If you put that in the groovyConsole and run it, it will evaluate to true. There are a couple of things to notice. First is the `==~` operator, which is similar to the `==` operator, but matches patterns instead of computing exact equality. Second is that the regular expression is enclosed in `/`'s. This tells groovy (and also anyone else reading your code) that this is a regular expression and not just a string.

But let's say that we also wanted to match the correct spelling, we could add a `'?'` after the `'e'` to say that the `e` is optional. The following will still evaluate to true.

```
"potatoe" ==~ /potatoe?/
```

And the correct spelling will also match:

```
"potato" ==~ /potatoe?/
```

But anything else will not match:

```
"motato" ==~ /potatoe?/
```

So this is how you define a simple boolean expression involving a regular expression. But let's get a little bit more tricky. Let's define a method that tests a regular expression. So for example, let's write some code to match Pete Wisniewski's last name:

```

def checkSpelling(spellingAttempt,
spellingRegularExpression)
{
    if (spellingAttempt ==~
spellingRegularExpression)
    {
        println("Congratulations, you
spelled it correctly.")
    } else {
        println("Sorry, try again.")
    }
}

theRegularExpression = /Wisniewski/
checkSpelling("Wisniewski",
theRegularExpression)
checkSpelling("Wisnewski",
theRegularExpression)

```

There are a couple of new things we have done here. First is that we have defined a function (actually a method, but I'll use the two words interchangeably). A function is a collection of code similar to a closure. Functions always have names, whereas closures can be "anonymous". Once we define this function we can use it over and over later.

In this function the **if** statement in bold tests to see if the parameter `spellingAttempt` matches the regular expression given to the function by using the `==~` operator.

Now let's get a little bit more tricky. Let's say we also want to match the string if the name does not have the 'w' in the middle, we might:

```
theRegularExpression = /Wisniew?ski/  
checkSpelling("Wisniewski",  
theRegularExpression)  
checkSpelling("Wisnieski",  
theRegularExpression)  
checkSpelling("Wisniewewski",  
theRegularExpression)
```

The single `?` that was added to the `spellingRegularExpression` says that the item directly before it (the character `w`) is optional. Try running this code with different spellings in the variable `spellingAttempt` to prove to yourself that the only two spellings accepted are now "Wisniewski" and "Wisnieski". (Note that you'll have to leave the definition of `checkSpelling` at the top of your `groovyConsole`)

The `**` is one of the characters that have special meaning in the world of regular expressions. You should probably assume that any punctuation has special meaning.

Now let's also make it accept the spelling if "ie" in the middle is transposed. Consider the following:

```
theRegularExpression = /Wisn(ie|ei)w?ski/  
checkSpelling("Wisniewski",  
theRegularExpression)  
checkSpelling("Wisnieski",  
theRegularExpression)  
checkSpelling("Wisniewewski",  
theRegularExpression)
```

Once again, play around with the spelling. There should be only four spellings that work, "Wisniewski", "Wisneiwski", "Wisnieski" and "Wisneiski". The bar character `|` says that either the thing to the left or the thing to the right is acceptable, in this case "ie" or "ei". The parentheses are simply there to mark the beginning and end of the interesting section.

One last interesting feature is the ability to specify a group of characters all of which are ok. This is done using square brackets `[ ]*`. Try the following regular expressions with various misspellings of Pete's last name:

```

theRegularExpression = /Wis[abcd]niewski/ //
requires one of 'a', 'b', 'c' or 'd'
theRegularExpression = /Wis[abcd]?niewski/
// will allow one of 'a', 'b', 'c' or 'd',
but not required (like above)
theRegularExpression = /Wis[a-zA-Z]niewski/
// requires one of any upper- or lower-case
letter
theRegularExpression = /Wis[^abcd]niewski/
// requires one of any character that is
''not'' 'a', 'b', 'c' or 'd'

```

The last one warrants some explanation. If the first character in the square brackets is a `^` then it means anything but the characters specified in the brackets.

### *The operators*

So now that you have a sense for how regular expressions work, here are the operators that you will find helpful, and what they do:

#### Regular Expression Operators

<code>a?</code>	matches 0 or 1 occurrence of <code>*a*</code>	'a' or empty string
<code>a*</code>	matches 0 or more occurrences of <code>*a*</code>	empty string or 'a', 'aa', 'aaa', etc
<code>a+</code>	matches 1 or more occurrences of <code>*a*</code>	'a', 'aa', 'aaa', etc
<code>alb</code>	match <code>*a*</code> or <code>*b*</code>	'a' or 'b'
<code>.</code>	match any single character	'a', 'q', 'l', '_', '+', etc
<code>[woeirjds]</code>	match any of the named characters	'w', 'o', 'e', 'i', 'r', 'j', 's', 'd'
<code>[1-9]</code>	match any of the characters in the range	'1', '2', '3', '4', '5', '6', '7', '8', '9'

<code>[^13579]</code>	match any characters not named	even digits, or any other character
<code>(ie)</code>	group an expression (for use with other operators)	'ie'
<code>^a</code>	match an <code>*a*</code> at the beginning of a line	'a'
<code>a\$</code>	match an <code>*a*</code> at the end of a line	'a'

There are a couple of other things you should know. If you want to use one of the operators above to mean the actual character, like you want to match a question mark, you need to put a `\` in front of it. For example:

```
// evaluates to true, and will for anything ending in a question mark (that doesn't have a question mark in it)
"How tall is Angelina Jolie?" =~ /[^\?]+\?/
```

This is your first really ugly regular expression. (The frequent use of these in PERL is one of the reasons it is considered a "write only" language). By the way, google knows how tall [she is](#). The only way to understand expressions like this is to pick it apart:

<code>/</code>	<code>[^?]</code>	<code>+</code>	<code>?</code>	<code>/</code>
begin expression	any character other than '?'	more than one of those	a question mark	end expression

So the use of the `\` in front of the `?` makes it refer to an actual question mark.

## Tutorial 5 - Capturing regex groups

### Capture groups

One of the most useful features of Groovy is the ability to use regular expressions to "capture" data out of a regular expression. Let's say for example we wanted to extract the location data of Liverpool, England from the following data:

```
locationData = "Liverpool, England: 53d 25m 0s N 3d 0m 0s"
```

We could use the `split()` function of string and then go through and strip out the comma between Liverpool and

England, and all the special location characters. Or we could do it all in one step with a regular expression. The syntax for doing this is a little bit strange. First, we have to define a regular expression, putting anything we are interested in in parentheses.

```
myRegularExpression = /([a-zA-Z]+),  
([a-zA-Z]+): ([0-9]+). ([0-9]+). ([0-9]+).  
([A-Z]) ([0-9]+). ([0-9]+). ([0-9]+)./
```

Next, we have to define a "matcher" which is done using the =~ operator:

```
matcher = ( locationData =~  
myRegularExpression )
```

The variable `matcher` contains a `java.util.regex.Matcher` as enhanced by groovy. You can access your data just as you would in Java from a `Matcher` object. A groovier way to get your data is to use the `matcher` as if it were an array--a two dimensional array, to be exact. A two dimensional array is simply an array of arrays. In this case the first "dimension" of the array corresponds to each match of the regular expression to the string. With this example, the regular expression only matches once, so there is only one element in the first dimension of the two-dimensional array. So consider the following code:

```
matcher[0]
```

That expression should evaluate to:

```
["Liverpool, England: 53d 25m 0s N 3d 0m  
0s", "Liverpool", "England", "53", "25",  
"0", "N", "3", "0", "0"]
```

And then we use the second dimension of the array to access the capture groups that we're interested in:

```

if (matcher.matches()) {
  println(matcher.getCount()+ " occurrence of
the regular expression was found in the
string.");
  println(matcher[0][1] + " is in the " +
matcher[0][6] + " hemisphere. (According to:
" + matcher[0][0] + ")")
}

```

Notice that the extra benefit that we get from using regular expressions is that we can see if the data is well-formed. That is if `locationData` contained the string "Could not find location data for Lima, Peru", the if statement would not execute.

### Non-matching Groups

Sometimes it is desirable to group an expression without marking it as a capture group. You can do this by enclosing the expression in parentheses with `?:` as the first two characters. For example if we wanted to reformat the names of some people, ignoring middle names if any, we might:

```

names = [
  "Graham James Edward Miller",
  "Andrew Gregory Macintyre",
  "No MiddleName"
]

printClosure = {
  matcher = (it =~ /(.*?)(?: .*)* (.*)/); //
notice the non-matching group in the middle
  if (matcher.matches())
    println(matcher[0][2]+", "+matcher[0][1]);
}
names.each(printClosure);

```

Should output:

```
Miller, Graham  
Macintyre, Andrew  
MiddleName, No
```

That way, we always know that the last name is the second matcher group.

### *Replacement*

One of the simpler but more useful things you can do with regular expressions is to replace the matching part of a string. You do that using the `replaceFirst()` and `replaceAll()` functions on `java.util.regex.Matcher` (this is the type of object you get when you do something like `myMatcher = ("a" += /b/);`).

So let's say we want to replace all occurrences of Harry Potter's name so that we can resell J.K. Rowlings books as Tanya Grotter novels (yes, someone tried this, Google it if you don't believe me).

```
excerpt = "At school, Harry had no one.  
Everybody knew that Dudley's gang hated that  
odd Harry Potter "+  
        "in his baggy old clothes and  
broken glasses, and nobody liked to disagree  
with Dudley's gang."  
matcher = (excerpt =~ /Harry Potter/);  
excerpt = matcher.replaceAll("Tanya  
Grotter");  
  
matcher = (excerpt =~ /Harry/);  
excerpt = matcher.replaceAll("Tanya");  
println("Publish it! "+excerpt);
```

In this case, we do it in two steps, one for Harry Potter's full name, one for just his first name.

### *Reluctant Operators*

The operators `?`, `+`, and `*` are by default "greedy". That is, they attempt to match as much of the input as possible. Sometimes this is not what we want. Consider the following list of fifth century popes:

```

popesArray = [
    "Pope Anastasius I 399-401",
    "Pope Innocent I 401-417",
    "Pope Zosimus 417-418",
    "Pope Boniface I 418-422",
    "Pope Celestine I 422-432",
    "Pope Sixtus III 432-440",
    "Pope Leo I the Great 440-461",
    "Pope Hilarius 461-468",
    "Pope Simplicius 468-483",
    "Pope Felix III 483-492",
    "Pope Gelasius I 492-496",
    "Pope Anastasius II 496-498",
    "Pope Symmachus 498-514"
]

```

A first attempt at a regular expression to parse out the name (without the sequence number or modifier) and years of each pope might be as follows:

```

/Pope (.*)(?: .*)? ([0-9]+)-([0-9]+)/

```

Which splits up as:

/	Pope	(.*)	(?: .*)?	([0-9]+)	-	([0-9]+)	/
begin expression	Pope	capture some characters	non-capture group: space and some characters	capture a number	-	capture a number	end expression

We hope that then the first capture group would just be the name of the pope in each example, but as it turns out, it captures too much of the input. For example the first pope breaks up as follows:

/	Pope	(.*)	(?:.*)?	([0-9]+)	-	([0-9]+)	/
begin expression	Pope	Anastasius I		399	-	401	end expression

Clearly the first capture group is capturing too much of the input. We only want it to capture Anastasius, and the modifiers should be captured by the second capture group. Another way to put this is that the first capture group should capture as little of the input as possible to still allow a match. In this case it would be everything until the next space. Java regular expressions allow us to do this using "reluctant" versions of the \*, + and ? operators. In order to make one of these operators reluctant, simply add a ? after it (to make \*?, +? and ??). So our new regular expression would be:

**`/Pope (.*)? (?:.*)? ([0-9]+)-([0-9]+)/`**

So now let's look at our new regular expression with the most difficult of the inputs, the one before Pope Hilarius (a real jokester), breaks up as follows:

/	Pope	(.*)?	(?:.*)?	([0-9]+)	-	([0-9]+)	/
begin expression	Pope	Leo	I the Great	440	-	461	end expression

Which is what we want.

So to test this out, we would use the code:

```

popesArray = [
    "Pope Anastasius I 399-401",
    "Pope Innocent I 401-417",
    "Pope Zosimus 417-418",
    "Pope Boniface I 418-422",
    "Pope Celestine I 422-432",
    "Pope Sixtus III 432-440",
    "Pope Leo I the Great 440-461",
    "Pope Hilarius 461-468",
    "Pope Simplicius 468-483",
    "Pope Felix III 483-492",
    "Pope Gelasius I 492-496",
    "Pope Anastasius II 496-498",
    "Pope Symmachus 498-514"
]

myClosure = {
    myMatcher = (it =~ /Pope (.*)?(?: .*)?
([0-9]+)-([0-9]+)/);
    if (myMatcher.matches())
        println(myMatcher[0][1]+":
" +myMatcher[0][2]+" to "+myMatcher[0][3]);
}
popesArray.each(myClosure);

```

Try this code with the original regular expression as well to see the broken output.

## Tutorial 6 - Groovy SQL

### Groovy SQL

This section borrows some content from this [GroovySQL article](#), by Andrew Glover. If some of the references to JDBC don't make sense, don't worry. There is one new language construct that is used below, which is the inclusion

of variables in string definitions. For example try the following:

```
piEstimate = 3
println("Pi is about ${piEstimate}")
println("Pi is closer to ${22/7}")
```

As you can see, in a string literal, Groovy interprets anything inside `${}` as a groovy expression.

This feature is used extensively below.

### *Performing a simple query*

Your first Groovy SQL code consists of three lines.

```
import groovy.sql.Sql
sql = Sql.newInstance(
    'jdbc:jtds:sqlserver://serverName/dbName-CLASS;domain=domainName', 'username',
    'password',
    'net.sourceforge.jtds.jdbc.Driver' )
sql.eachRow( 'select * from tableName' ) {
    println "$it.id -- ${it.firstName} --" }
```

The first line is a Java import. It simply tells Groovy the full name of the `Sql` object. The second line creates a new connection to the SQL database, and stores the connection in the variable `sql`.

This code is written for a `JTDS` connection to a `MS SQL Server` database. You will need to adjust all the parameters to `newInstance` to connect to your database, especially `username` and `password`.

Finally the third line calls the `eachRow` method of `sql`, passing in two arguments, the first being the query string, the second being a closure to print out some values.

Notice that in the closure the fields of "it" are accessed in two different ways. The first is as a simple field reference, accessing the `id` field of it. The second is the included Groovy expression mentioned above.

So the output from a row might look like:

```
001 -- Lane --
```

## Retrieving a single value from DB

If all you need is a value of one or a few columns of a single row in the DB, you could do this

```
row = sql.firstRow('select columnA, columnB
from tableName')
println "Row: columnA = ${row.columnA} and
columnB = ${row.columnB}"
```

## Doing more complex queries

The previous examples are fairly simple, but GroovySql is just as solid when it comes to more complex data manipulation queries such as insert, update, and delete queries. For these, you wouldn't necessarily want to use closures, so Groovy's Sql object provides the execute and executeUpdate methods instead. These methods are reminiscent of the normal JDBC statement class, which has an execute and an executeUpdate method as well.

Here you see a simple insert that uses variable substitution again with the `${}` syntax. This code simply inserts a new row into the people table.

```
firstName = 'yue'
lastName = "O'shea"
sql.execute("insert into people (firstName,
lastName) values (${firstName},
${lastName})")
```

Because the sql statement is expressed in a GString, the values `firstName` and `lastName` are provided as parameters and so the quote mark in `lastName` will not be seen as part of the statement.

Another way to do the same thing is to use prepared statements as follows:

```
firstName = 'yue'
lastName = 'wu'
sql.execute('insert into people (firstName,
lastName) values (?,?)', [firstName,
lastName])
```

The data that you want to insert is replaced with `"?"` in the insert statement, and then the values are passed in as an

array of data items. Updates are much the same in that they utilize the executeUpdate method. Notice, too, that in Listing 8 the executeUpdate method takes a list of values that will be matched to the corresponding ? elements in the query.

```
comment = 'Lazy bum'  
sql.executeUpdate('update people set comment  
= ? where id=002', [comment])
```

Deletes are essentially the same as inserts, except, of course, that the query's syntax is different.

```
sql.execute('delete from word where word_id  
= ?' , [5])
```

#### *Other Tips*

If you are content with using your resulting database columns in your business logic, it's nice and easy to just return a collection of GroovyRowResult objects which you can use directly:

```
def getPersons() {  
  def persons = []  
  sql.eachRow('Select * from Person') {  
    persons << it.toRowResult()  
  }  
  persons  
}
```

If you prefer to use a defined type instead of a GroovyRowResult, as long as your type has all the fields returned from your query you can just do:

```
Person p = new Person( it.toRowResult() )
```

#### **Differences to Other Languages**

- [Differences from Java](#)
- [Differences from Python](#)
- [Differences from Ruby](#)

## Differences from Java

Groovy tries to be as natural as possible for Java developers. We've tried to follow the principle of least surprise when designing Groovy, particularly for developers learning Groovy who've come from a Java background.

Here we list all the major differences between Java and Groovy.

### *Default imports*

All these packages and classes are imported by default, i.e. you do not have to use an explicit `import` statement to use them:

- `java.io.*`
- `java.lang.*`
- `java.math.BigDecimal`
- `java.math.BigInteger`
- `java.net.*`
- `java.util.*`
- `groovy.lang.*`
- `groovy.util.*`

### *Common gotchas*

Here we list the common things you might trip over if you're a Java developer starting to use Groovy.

- `==` means equals on all types. In Java there's a wierd part of the syntax where `==` means equality for primitive types and `==` means identity for objects. Since we're using autoboxing this would be very confusing for Java developers (since `x == 5` would be mostly false if `x` was 5 😊). So for simplicity `==` means `equals()` in Groovy. If you really need the identity, you can use the method `"is"` like `foo.is(bar)`. This does not work on null, but you can still use `==` here: `foo==null`.
- `in` is a keyword. So don't use it as a variable name.
- When declaring array you can't write

```
int[] a = {1,2,3};
```

you need to write

```
int[] a = [1,2,3]
```

- If you are used to writing a for loop that looks like

```
for (int i=0; i < len; i++) {...}
```

in groovy you can use that too, but you can use only one count variable. Alternatives to this are

```
for (i in 0..len-1) {...}
```

or

```
for (i in 0..<len) {...}
```

or

```
len.times {...}
```

### **Things to be aware of**

- Semicolons are optional. Use them if you like (though you must use them to put several statements on one line).
- The `return` keyword is optional.
- You can use the `this` keyword inside static methods (which refers to this class).
- Methods and classes are public by default.
- `Protected` in Groovy has the same meaning as `protected` in Java, i.e. you can have friends in the same package and derived classes can also see protected members.
- Inner (non-static nested) classes are not supported. In most cases you can use [closures](#) instead, e.g. to [implement interfaces](#). Static nested classes are supported.
- The `throws` clause in a method signature is not checked by the Groovy compiler, because there is no difference between checked and unchecked exceptions.
- You will not get compile errors like you would in Java for using undefined members or passing arguments of the wrong type. See [Runtime vs Compile time](#), [Static vs Dynamic](#).

### **Uncommon Gotchas**

Java programmers are used to semicolons terminating statements and not having [closures](#). Also there are instance initializers in class definitions. So you might see something like:

```
class Trial {  
    private final Thing thing = new Thing ( )  
;  
    { thing.doSomething ( ) ; }  
}
```

Many Groovy programmers eschew the use of semicolons as distracting and redundant (though others use them all the time - it's a matter of coding style). A situation that leads to difficulties is writing the above in Groovy as:

```
class Trial {
    private final thing = new Thing ( )
    { thing.doSomething ( ) }
}
```

This will throw a `MissingMethodException`!

The issue here is that in this situation the newline is not a statement terminator so the following block is treated as a [closure](#), passed as an argument to the `Thing` constructor. Bizarre to many, but true. If you want to use instance initializers in this sort of way, it is effectively mandatory to have a semicolon:

```
class Trial {
    private final thing = new Thing ( ) ;
    { thing.doSomething ( ) }
}
```

This way the block following the initialized definition is clearly an instance initializer.

Another document lists some [pitfalls](#) you should be aware of and give some advice on best practices to avoid those pitfalls.

#### ***New features added to Groovy not available in Java***

- [Closures](#)
- native [syntax](#) for lists and maps
- [GroovyMarkup](#) and GPath support
- native support for [regular expressions](#)
- polymorphic [iteration](#) and powerful [switch statement](#)
- dynamic and static typing is supported - so you can omit the type declarations on methods, fields and variables
- you can embed expressions inside [strings](#)
- lots of new helper methods added to the [JDK](#)
- simpler syntax for writing [beans](#) for both properties and adding event listeners
- [safe navigation](#) using the `?.` operator, e.g. "variable?.field" and "variable?.method()" - no more nested ifs to check for null clogging up your code

## **Differences from Python**

### **General**

<b>Python</b>	<b>Groovy</b>
---------------	---------------

<pre>repr(x)</pre>	<pre>x.inspect(), x.dump()</pre>
<pre>x.y if x else None</pre>	<pre>x?.y</pre>
<pre>"%(foo)s" % locals()</pre>	<pre>"\${foo}"</pre>

### Lists

Python	Groovy
<pre>not x</pre>	<pre>!x x.empty</pre>
<pre>len(x)</pre>	<pre>x.size()</pre>
<pre>for item, idx in enumerate(x): ...</pre>	<pre>x.eachWithIndex { item, idx -&gt; ... }</pre>

### Maps

Python	Groovy
<pre>{}</pre>	<pre>[:] // an empty map</pre>

<p>Depends:  d if used like: for k in d:  list(d) if list needed  d[iter].keys() explicitly</p>	<pre>d.keySet()</pre>
<pre>d.[iter]values()</pre>	<pre>d.values()</pre>
<pre>[k+1 for k in d]</pre>	<pre>d.collect { k, v -&gt; k+1 }</pre>
<pre>d = dict(zip(k, v))</pre>	<pre>k = 1..3 v = 'a'..'c' d = [:]; k.eachWithIndex { it, i -&gt; d[it] = v[i] } println d // [1:"a", 2:"b", 3:"c"]</pre>

#### Ranges/Slices

Python	Groovy
<pre>range(3)</pre>	<pre>0..&lt;3</pre>
<pre>range(1, 3+1)</pre>	<pre>1..3</pre>
<pre>range(0, 10, 2)</pre>	<p>not represented as a data type but you can use</p> <pre>0.step(10, 2) {...}</pre>

```
"abcdef"[3:]
```

```
"abcdef"[3..-1]
```

#### Object access

Python	Groovy
<pre>m = 'strip'; getattr('!', m)()</pre>	<pre>m = 'trim'; '!'."\$m"()</pre>
<pre>args = ('a', 2); 'abcabc'.find(*args)</pre>	<pre>args = ['a', 2]; 'abcabc'.indexOf(*args)</pre>

#### Differences from Ruby

The core abstract programming model of Ruby and Groovy are very similar: everything is an object, there is a MOP in control of all activity, and closures are the core structuring tool after classes. Ruby uses the Ruby library, Groovy uses the Java library with some additions of its own. This is the biggest difference but it is a huge difference.

Syntactically, things like:

```
File.open( 'blah' ) { | file | puts(  
file.read ) }
```

becomes:

```
println ( new File ( 'blah' ).text )
```

which doesn't show that the Groovy closures syntax is:

```
{ file -> doSomething ( file ) }
```

which is slightly different from Ruby, but does show that sometimes Groovy has a different approach to certain things compared to Ruby. So in moving from Ruby to Groovy, there are gotchas.

#### Groovy style and language feature guidelines for Java developers

A Java developer embarking on a Groovy adventure will always have Java in mind, and will progressively learn Groovy, one feature at a time, becoming more productive and writing more idiomatic Groovy code. This document's purpose is to guide such a developer along the way, teaching some common Groovy syntax style, new operators, and new features like closures, etc. This guide is not complete and only serves as a quick intro and a base for further guideline sections should you feel like contributing to the document and enhancing it.

### ***No semicolons***

When coming from a C / C++ / C# / Java background, we're so used to semicolons, that we put them everywhere. Even worse, Groovy supports 99% of Java's syntax, and sometimes, it's so easy to paste some Java code into your Groovy programs, that you end up with tons of semicolons everywhere. But... semicolons are optional in Groovy, you can omit them, and it's more idiomatic to remove them.

### ***Return keyword optional***

In Groovy, the last expression evaluated in the body of a method can be returned without necessitating the 'return' keyword. specially for short methods and for closures, it's nicer to omit it for brevity:

```
String toString() { return "a server" }  
String toString() { "a server" }
```

But sometimes, this doesn't look too good when you're using a variable, and see it visually twice on two rows:

```
def props() {  
    def m1 = [a: 1, b: 2]  
    m2 = m1.findAll { k, v -> v % 2 == 0 }  
    m2.c = 3  
    m2  
}
```

In such case, either putting a newline before the last expression, or explicitly using 'return' may yield better readability.

I, for myself, sometimes use the return keyword, sometimes not, it's often a matter of taste. But often, inside of closure, we omit it more often than not, for example. So even if the keyword is optional, this is by no means mandatory to not use it if you think it halts the readability of your code.

A word of caution, however. When using methods which are defined with the 'def' keyword instead of a specific concrete type, you may be surprised to see the last expression being returned sometimes. So usually prefer using a specific return type like void or a type. In our example above, imagine we forgot to put m2 as last statement to be returned, the last expression would be m2.c = 3, which would return... 3, and not the map you expect.

Statements like if/else, try/catch can thus return a value as well, as there's a "last expression" evaluated in those statements:

```
def foo(n) {
    if(n == 1) {
        "Roshan"
    } else {
        "Dawrani"
    }
}

assert foo(1) == "Roshan"
assert foo(2) == "Dawrani"
```

#### *Def and type*

As we're talking about def and types, I often see developers using both 'def' and a type. But 'def' is redundant here. So make a choice, either use 'def' or a type.

So don't write:

```
def String name = "Guillaume"
```

But:

```
String name = "Guillaume"
```

When using def in Groovy, the actual type holder is Object (so you can assign any object to variables defined with def, and return any kind of object if a method is declared returning def).

When defining a method with untyped parameters, you can use 'def' but it's not needed, so we tend to omit them. So instead of:

```
void doSomething(def param1, def param2) { }
```

Prefer:

```
void doSomething(param1, param2) { }
```

But as we mention in the last section of the document, it's usually better to type your method parameters, so as to help with documenting your code, and also help IDEs for code-completion, or for leveraging the static type checking or static compilation capabilities of Groovy.

Another place where 'def' is redundant and should be avoided is when defining constructors:

```
class MyClass {  
    def MyClass() {}  
}
```

Instead, just remove the 'def':

```
class MyClass {  
    MyClass() {}  
}
```

#### *Public by default*

By default, Groovy considers classes and methods 'public'. So you don't have to use the 'public' modifier everywhere something is public. Only if it's not public, you should put a visibility modifier.

So instead of:

```
public class Server {  
    public String toString() { return "a  
server" }  
}
```

Prefer the more concise:

```
class Server {  
    String toString() { "a server" }  
}
```

You may wonder about the 'package-scope' visibility, and the fact Groovy allows one to omit 'public' means that this scope is not supported by default, but there's actually a special Groovy annotation which allows you to use that visibility:

```
class Server {  
    @PackageScope Cluster cluster  
}
```

#### *Omitting parentheses*

Groovy allows you to omit the parentheses for top-level expressions, like with the println command:

```
println "Hello"  
method a, b
```

vs:

```
println("Hello")  
method(a, b)
```

When a closure is the last parameter of a method call, like when using Groovy's 'each' iteration mechanism, you can put the closure outside the closing parens, and even omit the parentheses:

```
list.each( { println it } )  
list.each(){ println it }  
list.each { println it }
```

Always prefer the third form, which is more natural, as an empty pair of parentheses is just useless syntactical noise!

There are some cases where Groovy doesn't allow you to remove parentheses. As I said, top-level expressions can omit them, but for nested method calls or on the right-hand side of an assignment, you can't omit them there.

```
def foo(n) { n }

println foo 1 // won't work
def m = foo 1
```

### *Classes as first-class citizens*

The .class suffix is not needed in Groovy, a bit like in Java's instanceof.

For example:

```
connection.doPost(BASE_URI + "/modify.hqu",
params, ResourcesResponse.class)
```

Using GStrings we're going to cover below, and using first class citizens:

```
connection.doPost("${BASE_URI}/modify.hqu",
params, ResourcesResponse)
```

### *Getters and Setters*

In Groovy, a getters and setters form what we call a "property", and offers a shortcut notation for accessing and setting such properties. So instead of the Java-way of calling getters / setters, you can use a field-like access notation:

```
resourceGroup.getResourcePrototype().getName  
( ) == SERVER_TYPE_NAME  
resourceGroup.resourcePrototype.name ==  
SERVER_TYPE_NAME  
  
resourcePrototype.setName("something")  
resourcePrototype.name = "something"
```

When writing your beans in Groovy, often called POGOs (Plain Old Groovy Objects), you don't have to create the field and getter / setter yourself, but let the Groovy compiler do it for you.

So instead of:

```
class Person {  
    private String name  
    String getName() { return name }  
    void setName(String name) { this.name =  
name }  
}
```

You can simply write:

```
class Person {  
    String name  
}
```

As you can see, a free standing 'field' without modifier visibility actually makes the Groovy compiler to generate a private field and a getter and setter for you.

When using such POGOs from Java, the getter and setter are indeed there, and can be used as usual, of course.

Although the compiler creates the usual getter/setter logic, if you wish to do anything additional or different in those getters/setters, you're free to still provide them, and the compiler will use your logic, instead of the default generated one.

### *Initializing beans with named parameters and the default constructor*

With a bean like:

```
class Server {  
    String name  
    Cluster cluster  
}
```

Instead of setting each setter in subsequent statements as follows:

```
def server = new Server()  
server.name = "Obelix"  
server.cluster = aCluster
```

You can use named parameters with the default constructor (first the constructor is called, then the setters are called in turn):

```
def server = new Server(name: "Obelix",  
cluster: aCluster)
```

### *Using with() for repeated operations on the same bean*

Named-parameters with the default constructor is interesting when creating new instances, but what if you are updating an instance that was given to you, do you have to repeat the 'server' prefix again and again? No, thanks to the with() method that Groovy adds on all objects of any kind:

```
server.name = application.name  
server.status = status  
server.sessionCount = 3  
server.start()  
server.stop()
```

vs:

```
server.with {
    name = application.name
    status = status
    sessionCount = 3
    start()
    stop()
}
```

### ***Equals and ==***

Java's == is actually Groovy's is() method, and Groovy's == is a clever equals()!

To compare the references of objects, instead of ==, you should use a.is(b).

But to do the usual equals() comparison, you should prefer Groovy's ==, as it also takes care of avoiding NullPointerException, independently of whether the left or right is null or not.

Instead of:

```
status != null &&
status.equals(ControlConstants.STATUS_COMPLETED)
```

Do:

```
status == ControlConstants.STATUS_COMPLETED
```

### ***GStrings (interpolation, multiline)***

We often use string and variable concatenation in Java, with many opening / closing of double quotes, plus signs, and \n characters for newlines. With interpolated strings (called GStrings), such strings look better and are less painful to type:

```
throw new Exception("Unable to convert
resource: " + resource)
```

vs:

```
throw new Exception("Unable to convert  
resource: ${resource}")
```

Inside the curly braces, you can put any kind of expression, not just variables. For simple variables, or variable.property, you can even drop the curly braces:

```
throw new Exception("Unable to convert  
resource: $resource")
```

You can even lazily evaluate those expressions using a closure notation with `${-> resource }`. When the GString will be coerced to a String, it'll evaluate the closure and get the `toString()` representation of the return value. Example:

```
int i = 3  
  
def s1 = "i's value is: ${i}"  
def s2 = "i's value is: ${-> i}"  
  
i++  
  
assert s1 == "i's value is: 3" // eagerly  
evaluated, takes the value on creation  
assert s2 == "i's value is: 4" // lazily  
evaluated, takes the new value into account
```

When strings and their concatenated expression are long in Java:

```
throw new PluginException("Failed to execute
command list-applications:" +
    " The group with name " +
    parameterMap.groupname[0] +
    " is not compatible group of type " +
    SERVER_TYPE_NAME)
```

You can use the `\` continuation character (this is not a multiline string):

```
throw new PluginException("Failed to execute
command list-applications: \
The group with name
${parameterMap.groupname[0]} \
is not compatible group of type
${SERVER_TYPE_NAME}")
```

Or using multiline strings with triple quotes:

```
throw new PluginException("""Failed to
execute command list-applications:
    The group with name
    ${parameterMap.groupname[0]}
    is not compatible group of type
    ${SERVER_TYPE_NAME})""")
```

You can also strip the indentation appearing on the left side of the multiline strings by calling `.stripIndent()` on that string.

Also note the difference between single quotes and double quotes in Groovy: single quotes always create Java Strings, without interpolation of variables, whereas double quotes either create Java Strings or GStrings when interpolated variables are present.

For multiline strings, you can triple the quotes: ie. triple double quotes for GStrings and triple single quotes for mere

Strings.

If you need to write regular expression patterns, you should use the "slashy" string notation:

```
assert "foooo/baaaaar" ==~ /fo+\//ba+r/
```

The advantage of the "slashy" notation is that you don't need to double escape backslashes, making working with regex a bit simpler.

Last but not least, prefer using single quoted strings when you need string constants, and use double quoted strings when you are explicitly relying on string interpolation.

#### *Native syntax for data structures*

Groovy provides native syntax constructs for data structures like lists, maps, regex, or ranges of values. Make sure to leverage them in your Groovy programs.

Here are some examples of those native constructs:

```
def list = [1, 4, 6, 9]  
  
// by default, keys are Strings, no need to  
quote them  
// you can wrap keys with () like  
[(variableStateAcronym): stateName] to  
insert a variable or object as a key.  
def map = [CA: 'California', MI: 'Michigan']  
  
def range = 10..20  
def pattern = ~/fo*/  
  
// equivalent to add()  
list << 5  
  
// call contains()  
assert 4 in list  
assert 5 in list
```

```
assert 15 in range

// subscript notation
assert list[1] == 4

// add a new key value pair
map << [WA: 'Washington']
// subscript notation
assert map['CA'] == 'California'
// property notation
assert map.WA == 'Washington'
```

```
// matches() strings against patterns  
assert 'foo' =~ pattern
```

### ***The Groovy Development Kit***

Continuing on the data structures, when you need to iterate over collections, Groovy provides various additional methods, decorating Java's core data structures, like `each{}`, `find{}`, `findAll{}`, `every{}`, `collect{}`, `inject{}`. These methods add a functional flavor to the programming language and help working with complex algorithms more easily. Lots of new methods are applied to various types, through decoration, thanks to the dynamic nature of the language. You can find lots of very useful methods on String, Files, Streams, Collections, and much more:

<http://groovy.codehaus.org/groovy-jdk/>

### ***The power of switch***

Groovy's switch is much more powerful than in C-ish languages which usually only accept primitives and assimilated. Groovy's switch accepts pretty much any kind of type.

```
def x = 1.23
def result = ""
switch (x) {
    case "foo": result = "found foo"
    // lets fall through
    case "bar": result += "bar"
    case [4, 5, 6, 'inList']:
        result = "list"
        break
    case 12..30:
        result = "range"
        break
    case Integer:
        result = "integer"
        break
    case Number:
        result = "number"
        break
    default: result = "default"
}
assert result == "number"
```

And more generally, types with an `isCase()` method can also decide whether a value corresponds with a case

### *Import aliasing*

In Java, when using two classes of the same name but from different packages, like `java.util.List` and `java.awt.List`, you can import one class, but have to use a fully-qualified name for the other.

Also sometimes, in your code, when using often a long class name, and the code becomes a bit more verbose.

To improve such situations, Groovy features import aliasing:

```
import java.util.List as juList
import java.awt.List as aList

import java.awt.WindowConstants as WC
```

You can also import methods statically:

```
import static pkg.SomeClass.foo
foo()
```

### ***Groovy Truth***

All objects can be 'coerced' to a boolean value: everything that's null, void or empty evaluates to false, and if not, evaluates to true.

So instead of writing `if (name != null && name.length > 0) {}`, you can just do `if (name) {}`. Same thing for collections, etc.

Thus, you can use some shortcuts in things like `while()`, `if()`, the ternary operator, the Elvis operator (see below), etc.

It's even possible to customize the Groovy Truth, by adding an `boolean asBoolean()` method to your classes!

### ***Safe graph navigation***

Groovy supports a variant of the `.` operator to safely navigate an object graph.

In Java, when you're interested in a node deep in the graph and need to check for null, you often end up writing complex `if`, or nested `if` statements like this:

```

if (order != null) {
    if (order.getCustomer() != null) {
        if (order.getCustomer().getAddress()
!= null) {

System.out.println(order.getCustomer().getAd
dress());
        }
    }
}

```

With ?. safe dereference operator, you can simplify such code with:

```

println order?.customer?.address

```

Nulls are checked throughout the call chain and no NullPointerException will be thrown if any element is null, and the resulting value will be null if something's null.

#### *Assert*

To check your parameters, your return values, and more, you can use the 'assert' statement. Contrary to Java's assert, asserts don't need to be activated to be working, so asserts are always checked.

```

def check(String name) {
    // name non-null and non-empty according
to Groovy Truth
    assert name
    // safe navigation + Groovy Truth to
check
    assert name?.size() > 3
}

```

You'll also notice the nice output that Groovy's "Power Assert" statement provides, with a graph view of the various values of each sub-expressions being asserted.

#### *Elvis operator for default values*

The Elvis operator is a special ternary operator shortcut which is handy to use for default values.

We often have to write code like:

```
def result = name != null ? name : "Unknown"
```

Thanks to Groovy Truth, the null check can be simplified to just 'name'.

And to go even further, since you return 'name' anyway, instead of repeating name twice in this ternary expression, we can somehow remove what's in between the question mark and colon, by using the Elvis operator, so that the above becomes:

```
def result = name ?: "Unknown"
```

#### *Catch any exception*

If you don't really care of the exception which are thrown inside your try block, you can simply catch any of them and simply omit the type of the caught exception. So instead of catching the throwables like in:

```
try {  
    // ...  
} catch (Throwable t) {  
    // something bad happens  
}
```

Then catch anything ('any' or 'all', or whatever makes you think it's anything):

```
try {  
    // ...  
} catch (any) {  
    // something bad happens  
}
```

## ***Optional typing advice***

I'll finish on some words on when and how to use optional typing. Groovy lets you decide whether you use explicit strong typing, or when you use 'def'.

I've got a rather simple rule of thumb: whenever the code you're writing is going to be used by others as a public API, you should always favor the use of strong typing, it helps making the contract stronger, avoids possible passed arguments type mistakes, gives better documentation, and also helps the IDE with code completion whenever the code is for your use only, like private methods, or when the IDE can easily infer the type, then you're more free to decide when to type or not

## **For those new to both Java and Groovy**

Java developers benefit from using Groovy, but so can you who **don't already know Java**. If you want to access the power of the Java Virtual Machine and Development Kit libraries when programming, but don't want to learn the Java Language, you can **use Groovy instead**. Or maybe you do want to learn Java, but do so the easy way: you can **learn Groovy first**. You'll be productive sooner, and can go on to learn more about Java at your own pace.

Much of the documentation on this website at Codehaus is for those who already know Java. These pages are for you who don't, so you can **learn enough of the Groovy basics** to easily use the other documentation on this website. They introduce **Groovy's core classes and syntax** together. All code examples have been tested using Groovy 1.0 or later inside a script. It's aimed at **you who have already programmed before**, just not in Java, maybe in PHP, Perl, or Visual Basic. Do note that although this documentation is correct and detailed, it's still a little raw because it's still being written.

[Getting Started](#) - enough background to dive into the tutorials that follow

### **1. Numeric Processing**

[Integer Math](#) - choose from many types of integers

[Decimal Math](#) - for high-precision decimal math

[Floating Point Math](#) - for high-speed decimal math

[Dates and Times](#) - enabling complex date manipulations

### **2. Collections**

[Lists and Sets](#) - group various items into a collection

[Arrays](#) - fixed-size arrays for faster collections

[Maps](#) - assign collected values to keys

### **3. Text Processing**

[Characters](#) - access the full power of Unicode

[Strings](#) - easily handle strings of characters

[String Pattern Matching](#) - find patterns within strings

### **4. Input and Output**

[Files](#) - manipulate the file system easily

[Streams, Readers, and Writers](#) - access data as a flow of information

### **5. Control Structures**

[Blocks, Closures, and Functions](#) - compose programs from many building blocks

[Expandos, Classes, and Categories](#) - encapsulate program complexity

[Program Control](#) - various ways to structure program logic

## 6. Data Typing

[Static Typing and Interfaces](#) - put compile-time restrictions in programs

[Inheritance](#) - use classes and methods for many purposes

[Exceptions](#) - handle exception and error conditions simply

## 7. Meta-Programming

[Interceptors](#) - intercept method calls

[MetaClasses](#) - add and modify behavior of objects

[Class Reflection](#) - examine and manipulate objects dynamically - **IN PROGRESS**

## Other Topics Coming

Packages

Multi-Threading

Networking

Internationalization

Annotations

Enums

Builders

Class Loading

Permissions

To continue learning Groovy, you can now go on to:

[Java](#), the engine behind Groovy's power and performance

[Swing](#), the graphical interface for Java, made easy with Groovy's own [SwingBuilder](#)

[Eclipse](#), the free IDE with a [Groovy plugin](#) to make managing your code easy

[useful Groovy modules](#), such as [Gant](#), which extend the Groovy system

[Grails](#), bringing the power of Groovy to website development and deployment

## JN0025-Starting

The Groovy Programming Language runs on top of the Java Runtime Environment, which itself runs on almost any computer system, such as Windows, Linux, and Macintosh. If you don't have Groovy installed, see these pages:

[Installing Groovy](#)

[Running Groovy](#)

If you don't have the Java Runtime Environment:

[Installing Java](#)

These tutorials for those new to both Java and Groovy are in a sequence that builds on knowledge already presented. This tutorial therefore starts with the basics. Throughout, we use code examples rather than lengthy explanations to present the features of Groovy, so you might miss things if you just skim. We don't (yet) explain what you would use the features for, but rely on your previous programming background for this.

The code snippets in these tutorials use comments to explain things:

```
//comment like this to end of line, ignoring
*/ and /* and ' and "
/*or comment like this, ignoring // and '
and " until: */
/*or comment over
many lines, /*with no nesting*/
```

Groovy code can contain strings:

```
'A string can be within single quotes on one
line... '
'''...or within triple single quotes
over many lines, ignoring // and */ and /*
comment delimiters,...'''
"...or within double quotes..."
""""...or within triple double quotes
over many lines. """"
```

Each line here does the same:

```

println 'hello, world' //the function
'println' prints a string then newline
print 'hello, world\n' //'print' doesn't
print newline, but we can embed
                                //newlines ('\n' on
Unix/Linux, '\r\n' on Windows)
println 'hello' + ', ' + 'world' // + joins
strings together
print 'hello, '; println 'world'
                                //use semi-colons to
join two statements on one line
println( 'hello, world' )
                                //can put command parameter in
parens, sometimes we might have to
def a= 'world'; println 'hello, ' + a
                                //'def' to define a
variable and give it a value
print 'hello, world'; println()
//empty parens must be used for no-arg
functions; here, prints a blank line
def b= 'hello', c= 'world'; println "$b,
${c}"
                                // $ in print string
captures variable's value

```

We can also assign integers and decimals to variables:

```
def g = 7, groovy = 10.2
    //we can separate more than one defined
variable by a comma
print g + ', ' + groovy + '\n' //prints: 7,
10.2
assert g + ', ' + groovy == '7, 10.2' //we
can use assert statement and ==

//operator to understand examples
```

We can use operators like + - \* / and parentheses ( ) with numbers, following usual math grouping rules:

```
assert 4 * ( 2 + 3 ) - 6 == 14 //integers
only
assert 2.5 + 7 == 9.5
assert 7 / 4 == 1.75 //decimal number or
division converts expression to decimal
```

We can use the operators == > < >= <= != with numbers, the values true and false, the operators ! (not), && (and), and || (or), all with parentheses, to produce boolean expressions:

```
assert 2 > 3 == false
assert 7 <= 9
assert 7 != 2
assert true
assert ! false
assert 2 > 3 || 7 <= 9
assert ( 2 > 3 || 4 < 5 ) && 6 != 7
```

Variables are versatile:

```
def a
  assert a == null
  //variables defined but not given a value
  have special value null
def b = 1
  assert b == 1
  b = 2
  assert b == 2 //variables can be re-assigned
  to
  b = 'cat'
  assert b == 'cat' //they can be re-assigned
  different types/classes of data
  b = null
  assert b == null //they can be unassigned
```

All names in Groovy, including variable names, can contain any alphabetic character or the underscore, and contain any digit not in first position:

```

def abc= 4
def a23c= 4
def ab_c= 4
def _abc= 4

def ABC= 4
assert abc == ABC //although their values
are the same...
assert ! abc.is( ABC ) //...the variables
'abc' and 'ABC' are different,
                        //the names being
case-sensitive

/*these each produce compile errors when
uncommented...
def abc //already defined
def a%c= 4 //not a valid name because it
contains a symbol other than _
def 2bc= 4 //may not contain a digit in
first position
*/

```

All data in Groovy is built from "classes" and instances of them. Class names by convention begin with an uppercase character:

```
assert Byte.MAX_VALUE == 127
    //a class can have attached variables,
called 'fields'
assert Byte.parseByte('34') == 34
    //a class can have attached functions,
called 'methods'
def b= new Byte('34')
    //we can create an 'instance' of a class
using the 'new' keyword
assert b.intValue() == 34
    //each instance can also have attached
fields and methods
```

We can inspect the class of any entity, such as numbers and strings, using the class field:

```
assert 4.class == Integer //the common types
have both a short name...
assert 4.class == java.lang.Integer //...and
a long name
assert 4.5.class == BigDecimal
assert 'hello, world'.class == String
def a= 7
assert a.class == Integer
```

There are many predefined classes in Groovy, but only the most common ones are always visible to Groovy code. Most need to be qualified with a "package" name, eg, 'java.text.DecimalFormat', or the package must be imported beforehand:

```
import java.text.*
assert new DecimalFormat( '#,#00.0#'
).format( 5.6789 ) == '05.68'
```

Or:

```
assert new java.text.DecimalFormat(
'#,#00.0#' ).format( 5.6789 ) == '05.68'
```

If a line can be interpreted as a valid statement, it will be:

```
def i=
1 //because 'def i=' isn't a valid
statement,
  //the '1' is appended to the previous line

//a compile error when uncommented: 'def j'
is valid, so is interpreted as
//a statement. Then the invalid '= 1' causes
the error...
/*
def j
= 1
*/

def k \
= 1 //a backslash ensures a line is never
interpreted as a standalone statement
```

Sometimes code in a script doesn't compile: we comment it out in our examples. Other code compiles but generates a "checked exception" which we can catch and handle:

```
try{
    'moo'.toLong() //this will generate an
exception
    assert false
        //this code should never be reached,
so will always fail if executed
}catch(e){ assert e instanceof
NumberFormatException }
    //we can check the exception type using
'instanceof'
```

We can use square brackets [ ] to represent both ordered lists and key mappings:

```
def list= [1, 2, 3]
list= [] //empty list
list= [1, 'b', false, 4.5 ] //mixed types of
values OK
assert list[0] == 1 && list[1] == 'b' && !
list[2] && list[3] == 4.5
    //we can refer to items individually by
index

def map= [1:'a', 2:'b', 3:'c'] //map
indicated with colon :
map= [:] //empty map
map= ['a': 1, 'b': 'c', 'groovy': 78.9, 12:
true] //mixed types of values
assert map['a'] == 1 && map['b'] == 'c' &&
```

```
map['groovy'] == 78.9 && map[12]
  //we can refer to values individually by
key
```

```
'each' tells the code following it to
execute for each item in a list or map:
//for every item in list, assign to 'it' and
execute the following code...
```

```
[ 2, -17, +987, 0 ].each{
  println it
}
```

```
//we can specify a different name for the
argument other than the default...
```

```
[ 2, -17, +987, 0 ].each{ n ->
  println n
}
```

```
//we can specify two or more arguments, as
with this map...
```

```
[ 1: 3, 2: 6, 3: 9, 4: 12 ].each{ k, v->
  assert k * 3 == v
}
```

We can specify a list as a 'range', ie, by only the first and last items:

```
( 3..7 ).each{ println it } //prints numbers
3, 4, 5, 6, and 7
( 3..<7 ).each{ println it } //prints
numbers 3, 4, 5, and 6 //excludes 7
```

We can convert data of one type to another using the 'as' keyword:

```
assert ('100' as Integer) == 100
```

Sometimes, we need to use a more efficient type of list known as an array, where the type of each element must be the same. Arrays can't be represented directly in the syntax, but we can convert a list to one easily:

```
def x = ['97', '98', '99'] as Integer[]
//convert each item in list to an Integer
assert x[0] == 97 && x[1] == 98 && x[2] ==
99 //access each element individually
```

We can choose between two execution options using the if-else-statement:

```
def a= 2
if( a < 5 ){
  println "a, being $a, is less than 5."
}else{
  assert false //this line should never
execute
}
```

We can execute some code a certain number of times:

```
def i=0
10.times{ println i++ } //increment i by 1
after printing it

//another less declarative style of
looping...
while( i > 0 ){
  println i-- //decrement i by after
printing it
}
```

We can enclose code in parentheses and execute it later. The enclosed code is called a "closable block" or "closure":

```

def c= { def a= 5, b= 9; a * b }
assert c() == 45

[ { def a= 'ab'; a + 'bc' },
  { 'abbc' } ,
].each{ assert it() == 'abbc' }

```

We can spawn new threads from our main thread:

```

def i=0, j=0
def f= new File('TheOutput.txt') //create or
overwrite this file
Thread.start{
  while(true){
    i++
    if(i%1000 == 0) f<< 'S' //spawned thread
  }
}
while(true){
  j++
  if(j%1000 == 0) f<< 'M' //main thread
}

```

After, say, 5 seconds, abort the program then look at the file. On many computers, it'll show a roughly equal distribution of 'S' and 'M', but there'll be some irregularities showing that thread scheduling isn't perfectly timed.

The tutorials following are grouped into functional areas, beginning with numeric processing, and build up to the advanced features of Groovy.

### JN0515-Integers

Groovy numbers are either decimals or integers. The 3 main types of integers are Integer, Long, and BigInteger. BigInteger has no size limit, while Integer and Long do. We can enquire their minimum and maximum values:



```
assert 0x7FFFFFFF.class == Integer
assert (-0x7FFFFFFF).class == Integer //we
must negate using the minus sign
assert 0x80000000.class == Long
assert (-0x80000000).class == Integer
assert (-0x80000001).class == Long
```

We can force an integer (including hexadecimals and octals) to have a specific type by giving a suffix (I for Integer, L for Long, G for BigInteger), either uppercase or lowercase:

```
assert 42i.class == Integer //lowercase i
more readable
assert 123L.class == Long //uppercase L more
readable
assert 456g.class == BigInteger
assert 0xFFi.class == Integer
```

### Fixed-Size Integers

The fixed-size integers, Integer and Long, each have size limits but are more efficient in calculations.

There are also the less common Byte and Short types of integer, which act like the Integer type in math operations.

```
assert Short.MAX_VALUE == 32767
assert Short.MIN_VALUE == -32768
assert Byte.MAX_VALUE == 127
assert Byte.MIN_VALUE == -128
def a= new Byte('34'), b= new Byte('2')
assert (a+b).class == Integer
```

We can enquire the bit-size of each type of fixed-size integer:

```
assert Integer.SIZE == 32  
assert Long.SIZE == 64  
assert Short.SIZE == 16  
assert Byte.SIZE == 8
```

The class `Integer` can often be written `int`. The classes `Long`, `Short`, and `Byte` can each also often be written uncapitalized, ie, `long`, `short`, and `byte`. We can enquire these alternative (aka "primitive type") names:

```
assert Integer.TYPE == int  
assert Long.TYPE == long  
assert Short.TYPE == short  
assert Byte.TYPE == byte
```

The fixed-size integer classes can be converted to one another:

```
assert 45L as Integer == 45i
assert 45L as int == 45i //example of using
'int' for Integer
assert 45L.toInteger() == 45i //alternative
syntax
assert 23L.intValue() == 23i //another
alternative syntax

assert 45i as Long == 45L
assert 45i as long == 45L
assert 23i.toLong() == 23L
assert 45i.longValue() == 45L

//if converted number too large for target,
only lowest order bits returned...
assert 256i as Byte == 0
assert 200i as byte == -56 //...and this may
result in a negative number
```

We can create new fixed-sized integers from strings:

```
assert '42'.toInteger() == 42i
assert '56'.toLong() == 56L

try{ 'moo'.toLong(); assert false }
catch(e){ assert e instanceof
NumberFormatException }

assert new Integer( '45' ) == 45i
assert new Byte( '45' ) == 45 as byte

try{ new Integer( 'oink' ); assert false }
catch(e){ assert e instanceof
NumberFormatException }
```

To convert from a fixed-size integer to a string in various bases:

```
//second character is the base/radix...
assert Integer.toString( 29, 16 ) == '1d'

//Long version behaves just like Integer
version...
assert Long.toString( 29L, 16 ) == '1d'

//if number is negative, so is first
character of returned string...
assert Integer.toString( -29, 16 ) == '-1d'

//only time result begins with zero is if it
is zero...
assert Integer.toString(0) == '0'

assert Integer.toString( 29, 16
).toUpperCase() == '1D'

//second argument defaults to 10...
assert Integer.toString( 29 ) == '29'

//Short version only accepts one parameter,
only allowing base 10...
assert Short.toString( 29 as short ) == '29'
```

If the base/radix isn't between `Character.MIN_RADIX` and `Character.MAX_RADIX`, base 10 is used instead:

```
assert Integer.toString( 999,  
Character.MIN_RADIX - 1 ) ==  
    Integer.toString( 999, 10 )  
  
assert Integer.toString( 999,  
Character.MAX_RADIX + 1 ) ==  
    Integer.toString( 999, 10 )  
  
assert Character.MAX_RADIX == 36  
    //the symbols letters  
0123456789abcdefghijklmnopqrstuvwxyz are  
used
```

The common bases have similar methods which always return an unsigned integer:

```

assert Integer.toHexString(29) == '1d'
//return unsigned base-16 integer
assert Integer.toHexString(0) == '0'
assert Integer.toHexString(-17) ==
'ffffffef'
assert Long.toHexString(-17L) ==
'ffffffffffffffef'

//same as toString(,16) when number
positive...
assert Integer.toHexString(29) ==
Integer.toString(29,16)

//...but different when number negative
assert Integer.toHexString(-17) !=
Integer.toString(-17,16)

assert Integer.toOctalString(29) == '35'
assert Integer.toOctalString(0) == '0'
assert Integer.toOctalString(-17) ==
'3777777757'
assert Integer.toBinaryString(29) == '11101'

```

We can convert a string representation to an integer, using a specified base/radix:

```
assert Integer.parseInt("0", 10) == 0
assert Integer.parseInt("473", 10) == 473
assert Long.parseLong("473", 10) == 473L
//Long type has similarly-acting method
assert Integer.parseInt("473") == 473 //base
10 is the default base/radix
assert Integer.parseInt("-0", 10) == 0
assert Integer.parseInt("-FF", 16) == -255
assert Integer.parseInt("1100110", 2) == 102
assert Integer.parseInt("2147483647", 10) ==
2147483647
assert Integer.parseInt("-2147483648", 10)
== -2147483648
assert Integer.parseInt("Kona", 27) ==
411787
assert Long.parseLong("Hazelnut", 36) ==
1356099454469L
assert Short.parseShort("-FF", 16) == -255
```

A NumberFormatException may be thrown:

```

[ { Integer.parseInt("2147483648", 10) },
//number too large
  { Integer.parseInt("99", 8) }, //digit 9
not octal
  { Integer.parseInt("Kona", 10) }, //digits
not decimal
  { Integer.parseInt("1111",
Character.MIN_RADIX - 1 ) }, //radix too
small
  { Integer.parseInt("1111",
Character.MAX_RADIX + 1 ) }, //radix too
large
  { Integer.parseInt( '@#$%' ) }, //invalid
number
  { Integer.parseInt( '' ) }, //invalid
number
].each{ c->
  try{ c(); assert false }
  catch(e){assert e instanceof
NumberFormatException}
}

```

An alternative method name is:

```

assert Integer.valueOf( '12af', 16 ) ==
0x12af
    //same as: Integer.parseInt( '12af', 16
)

assert Long.valueOf( '123' ) == 123
    //same as: Long.parseInt( '123' )

assert Short.valueOf( 027 as short ) == 027

```

We can convert a string to a fixed-size integer, similar to `parseInt()` etc, but with the radix instead indicated inside the string:

```

assert Integer.decode('0xff') == 0xFF
assert Integer.decode('#FF') == 0xFF
assert Long.decode('#FF') == 0xFFL //long,
short, and byte also can be decoded
assert Short.decode('#FF') == 0xFF as short
assert Byte.decode('#F') == 0xF as byte
assert Integer.decode('-077') == -077
assert Integer.decode('2345') == 2345

try{ Integer.decode('7 @8'); assert false }
catch(e){ assert e instanceof
NumberFormatException }

```

We can return an integer representing the sign:

```
assert Integer.signum(45i) == 1
assert Integer.signum(0i) == 0
assert Integer.signum(-43i) == -1
assert Long.signum(-43L) == -1
```

We can compare fixed-size integers with each other:

```
assert 45i.compareTo( 47L ) < 0
assert (45 as byte).compareTo( 43 as short )
> 0
assert 45.compareTo( 45 ) == 0
```

#### Calculations with Fixed-Size Integers

We can perform addition, subtraction, multiplication, exponents, modulus, and negations on Integers and Longs, using both an operator syntax and a method syntax:

```
assert 34 + 33 == 67      && 34.plus( 33 ) ==
67
assert 34L - 21L == 13L && 34L.minus( 21L )
== 13L
assert 3 * 31 == 93      && 3.multiply( 31 )
== 93
assert 23 % 3 == 2      && 23.mod( 3 ) == 2
assert 3**2 == 9        && 3.power( 2 ) == 9
```

Not all calculations have a special operator symbol:

```
assert 22.intdiv(5) == 4
assert (-22).intdiv(5) == -4
assert (-34).abs() == 34
assert (-34L).abs() == 34L
```

We can increment and decrement variables, using operators, either before and after evaluation:

```
def a= 7
assert a++ == 7 && a == 8 && a-- == 8 && a
== 7 &&
      ++a == 8 && a == 8 && --a == 7 && a
== 7

def b = 7, c = 7 //These operators use
methods next() and previous()
assert ( ++b ) == ( c = c.next() )
assert b == c
assert ( --b ) == ( c = c.previous() )
assert b == c
assert ( b++ ) == { def z = c; c = c.next();
z }()
assert b == c

def b= Integer.MAX_VALUE
assert ++b == Integer.MIN_VALUE && --b ==
Integer.MAX_VALUE
```

Rules of parentheses and precedence apply to these operators. The operators have the same precedence irrespective of what type of values they operate on.

```
assert 3*(4+5) != 3*4+5 //parenthesized
expressions always have highest precedence
```

```
assert -3**2 == -(3**2) //power has next
highest precedence
```

```
assert ( 2*3**2 == 2*(3**2) ) && ( 2*3**2 !=
(2*3)**2 )
```

```
assert -3+2 != -(3+2) //unary operators have
next highest precedence
```

```
assert ~234 == -(~234) //unary operators
group right-to-left
```

```
//multiplication and modulo have next
highest precedence
```

```
assert 3*4%5 == (3*4)%5 //multiplication and
modulo have equal precedence
```

```
assert 3%4*5 == (3%4)*5
```

```
//addition and subtraction have equal
precedence, lower than mult/etc
```

```
assert 4+5-6 == 3
```

```
assert 5+3*4 == 5+(3*4)
```

Integers often convert their types during math operations. For + - \*, a Long with an Integer converts the Integer to a Long:

```
assert (23i+45L).class == Long
```

Because the fixed-sized integers have fixed width, they might overflow their boundaries during math operations, so

we need to be aware of the range of values we'll use a fixed-size integer for:

```
//each 256 is an int, so final product also  
an int, and calc overflowed...
```

```
assert 256*256*256*256 == 0
```

```
//we can fix this problem by using a long at  
the beginning of the calculation...
```

```
assert 256L*256*256*256 == 4294967296L
```

We can compare fixed-size integers using < <= > >= operators, of lower precedence than addition/etc:

```
assert 14 > 7                && 14.compareTo(7) >  
0
```

```
assert 14 >= 8               && 14.compareTo(8) >=  
0
```

```
assert -4 < 3                 && (-4).compareTo(3)  
< 0
```

```
assert -14 <= -9             &&  
(-14).compareTo(-9) <= 0
```

The operators == != <=> are of lower precedence than the other comparison operators:

```

def a = 4, b = 4, c = 5
assert a == b && a.equals(b)
assert a != c && ! a.equals(c)
assert (4 <=> 7) == -1 && 4.compareTo(7) ==
-1
assert (4 <=> 4) == 0 && 4.compareTo(4) ==
0
assert (4 <=> 2) == 1 && 4.compareTo(2) ==
1

```

#### Bit-Manipulation on Fixed-Sized Integers

We can examine and manipulate the individual bits on the fixed-sized integers.

To return an int or long with a single 1-bit in the position of the highest-order 1-bit in the argument:

```

assert Integer.highestOneBit( 45 ) == 32
assert Integer.highestOneBit( 27 ) == 16
assert Integer.highestOneBit( 0 ) == 0
assert Integer.highestOneBit( -1 ) ==
-128*256*256*256
assert Long.highestOneBit( -1L ) ==
-128*256*256*256 * 256*256*256*256

assert Integer.lowestOneBit( 45i ) == 1
//match lowest order 1-bit in argument
assert Integer.lowestOneBit( 46i ) == 2
assert Integer.lowestOneBit( 48i ) == 16

```

To return the number of zero bits preceding the highest-order 1-bit:

```

[ 0:32, 1:31, 2:30, 4:29 ].each{ k, v->
  assert Integer.numberOfLeadingZeros( k )
  == v
  //returns the number of zero-bits
preceding the highest-order 1-bit
  assert Long.numberOfLeadingZeros( k as
long ) == v + 32
}

[ 0:32, 45:0, 46:1, 48:4 ].each{ k, v->
  assert Integer.numberOfTrailingZeros( k )
  == v
  //returns the number of 0-bits following
the lowest-order 1-bit
}

//returns the number of 1-bits in the binary
representation...
assert Integer.bitCount( 7 ) == 3
assert Integer.bitCount( -1 ) == 32

```

We can perform a bitwise complement of the bits in a fixed-size integer using the ~ operator:

```

def x= 0x33333333i
assert ~x == -x - 1
  //how bitwise complement and negation
are related under 2's-complement

```

We can shift the bits of a fixed-size integer to the left or right. This is of lower precedence than addition/etc, but higher than the comparison operators.

```

//shift 4 bits to the left...
assert 0xB4F<<4 == 0xB4F0      &&
0xB4F.leftShift( 4 ) == 0xB4F0

//shift 4 bits to the right, dropping off
digits...
assert 0xD23C>>4 == 0xD23      &&
0xD23C.rightShift( 4 ) == 0xD23

//sign-extension performed when
right-shifting...
assert -0xFFF>>4 == -0x100     &&
(-0xFFF).rightShift( 4 ) == -0x100

//...unless triple >>> used
assert -0xFFF>>>4 == 0xFFFFF00 &&
(-0xFFF).rightShiftUnsigned(4) == 0xFFFFF00
[ 0xABC, -0x98765 ].each{
    it << 8 == it >> -8
}

```

We can rotate the bits in an integer or long:

```

assert Integer.rotateLeft( 0x456789AB, 4 )
== 0x56789AB4
    //we use multiples of 4 only to show
what's happening easier

assert Integer.rotateLeft( 0x456789AB, 12 )
==
    Integer.rotateRight( 0x456789AB,
Integer.SIZE - 12 )
    //rotating left and right are inverse
operations

assert Integer.rotateLeft( 0x456789AB, 32 )
== 0x456789AB //no change here

assert Long.rotateRight( 0x0123456789ABCDEF,
40 ) == 0x6789ABCDEF012345

```

We can perform bitwise 'and', 'or', and 'xor' operations on fixed-size integers. This is of lower precedence than the comparison operators.

```

assert (0x33 & 0x11) == 0x11 &&
0x33.and(0x11) == 0x11
assert (0x33 | 0x11) == 0x33 &&
0x33.or(0x11) == 0x33
assert (0x33 ^ 0x11) == 0x22 &&
0x33.xor(0x11) == 0x22

```

We can reverse the bits or bytes of the binary representation of an int or long:

```

assert Integer.toString( 123456, 2 ) ==
'11110001001000000'

assert Integer.toString( Integer.reverse(
123456 ), 2 ) ==
'10010001111000000000000000' //reverse
bits

assert Integer.reverseBytes( 0x157ACE42 ) ==
0x42CE7A15 //also works for bytes

```

#### Boolean, Conditional, and Assignment Operators with Fixed-Sized Integers

The boolean, conditional, and assignment operators are of even lower precedence than the bitwise operators.

When using an integer with boolean operators `!`, `&&`, and `||`, `0` evaluates to false, while every other integer evaluates to true:

```

assert ! 0; assert 1; assert 2; assert -1;
assert -2
assert ( ! 1 && 0 ) != ( ! (1 && 0) )
// the unary ! has the same, high,
precedence as the other unary operators
assert ( 1 || 0 && 0 ) != ( (1 || 0) && 0 )
// && has higher precedence than ||

```

The boolean operators `&&` and `||` only have their operands evaluated until the final result is known. This affects operands with side effects, such as increment or decrement operators:

```

def x = 0
0 && x++
assert x == 0
//x++ wasn't performed because falsity of (0
&& x++) was known when 0 evaluated

1 || x++
assert x == 0
//x++ wasn't performed because truth of (1
|| x++) was known when 1 evaluated

```

We can use the conditional operator `?:`, of lower precedence than the boolean operators, to choose between two values:

```

def x = 1? 7: -5
assert x == 7

```

We can put the assignment operator `=` within expressions, but must surround it with parentheses because its precedence is lower than the conditional:

```

def x, y = (x = 3) && 1
assert (x == 3) && y

def i = 2, j = (i=3) * i
//in the multiplication, lefthand (i=3)
evaluated before righthand i
assert j == 9

```

Of equal precedence as the plain assignment operator `=` are the quick assignment `*= += -= %= **= <<= >>= >>>=` &= ^= |= operators:

```
def a = 7
a += 2 //short for a = a + 2
assert a == 9
a += (a = 3) //expands to a = a + (a = 3)
before any part is evaluated
assert a == 12
```

### BigIntegers

The BigInteger has arbitrary precision, growing as large as necessary to accommodate the results of an operation.

We can explicitly convert fixed-sized integers to a BigInteger, and vice versa:

```
assert 45i as BigInteger == 45g
assert 45L.toBigInteger() == 45g
assert 45g as Integer == 45i
assert 45g.intValue() == 45i //alternate
syntax
assert 45g as Long == 45L
assert 45g.longValue() == 45L

assert 256g as Byte == 0
    //if converted number too large for
target, only lowest order bits returned
assert 200g as byte == -56 //...and this may
result in a negative number
```

A method and some fields that give a little more efficiency:

```
assert BigInteger.valueOf( 45L ) == 45g
    //works for longs only (not for ints,
shorts, or bytes)
assert BigInteger.ZERO == 0g
assert BigInteger.ONE == 1g
assert BigInteger.TEN == 10g
```

We can construct a BigInteger using an array of bytes:

```

assert new BigInteger( [1,2,3] as byte[] )
== 1g*256*256 + 2*256 + 3
    //big-endian 2's complement
representation
try{new BigInteger( [] as byte[] ); assert
0}
catch(e){assert e instanceof
NumberFormatException} //empty array not
allowed

assert new BigInteger( -1, [1,2] as byte[] )
== -258g
    //we pass in sign as a separate argument

assert new BigInteger( 1, [1,2] as byte[] )
== 258g

assert new BigInteger( 0, [0,0] as byte[] )
== 0g

assert new BigInteger( 1, [] as byte[] ) ==
0 //empty array allowable

try{ new BigInteger( 2, [1,2,3] as byte[] );
assert 0 }
catch(e){ assert e instanceof
NumberFormatException}
    //sign value must be -1, 0, or 1

```

We can convert a BigInteger back to an array of bytes:

```

def ba= (1g*256*256 + 2*256 +
3).toByteArray()
    //big-endian 2's complement
representation
assert ba.size() == 3 && ba[ 0 ] == 1 && ba[
1 ] == 2 && ba[ 2 ] == 3

def bb= 255g.toByteArray()
assert bb.size() == 2 && bb[ 0 ] == 0 && bb[
1 ] == -1
    //always includes at least one sign bit

def bc= -(2g*256 + 3).toByteArray()
assert bc.size() == 2 && bc[ 0 ] == -3 &&
bc[ 1 ] == -3

```

We can pass in a string in a certain base/radix:

```

assert '27'.toBigInteger() == 27g
assert new BigInteger("27", 10) == 27g
assert new BigInteger("27") == 27g //default
radix is 10
assert new BigInteger("110", 2) == 6g
assert new BigInteger("-1F", 16) == -31g
[ { new BigInteger(" 27", 10) }, //no
whitespace allowed in string
  { new BigInteger("Z", Character.MAX_RADIX
+ 1 ) }, //radix out of range
  { new BigInteger("0", Character.MIN_RADIX
- 1 ) }, //radix out of range
].each{
  try{ it(); assert 0 }catch(e){ assert e
instanceof NumberFormatException }
}

```

We can convert the BigInteger back to a string:

```

assert 6g.toString(2) == '110'
assert (-31g).toString(16) == '-1f'
assert 27g.toString() == '27' //default
radix is 10
assert 27g.toString( Character.MAX_RADIX + 1
) == '27'
//radix is 10 if radix argument invalid

```

We can construct a randomly-generated BigInteger:

```
assert new BigInteger( 20, new Random()
).toString( 2 ).size() == 20
    //20 is max bit length, must be >= 0
assert new BigInteger( 20, new Random() ) >=
0
```

#### Arithmetic with BigIntegers

We can perform the usual arithmetic operations + - \* using either methods or operations:

```
assert 34g.plus( 33g ) == 34g + 33g
assert 34g.add( 33g ) == 34g + 33g
//alternative name for plus
assert 34g.minus( 21g ) == 34g - 21g
assert 34g.subtract( 21g ) == 34g - 21g
//alternative name for minus
assert 3g.multiply( 31g ) == 3g * 31g
assert 7g.negate() == -7g //unary
operation/method
assert (-7g).negate() == 7g
```

For + - \*, a BigInteger causes any fixed-width integers in the calculation to be converted to a BigInteger:

```
assert (45L + 123g).class == BigInteger
assert (23 - 123g).class == BigInteger
assert ( 3g * 31 ).class == BigInteger
assert ( 3 * 31g ).class == BigInteger
assert 3g.multiply( 31 ).class == BigInteger
assert 3.multiply( 31g ).class == BigInteger
```

We can introduce a BigInteger into an expression with Integers or Longs if overflow may occur. But make sure the BigInteger is introduced before an intermediate value that may overflow, for example, the first-used value in a calculation:

```
assert 256L*256*256*256 * 256*256*256*256 ==
0
    //the first 256 is a Long, so each
intermediate and final product also Long,
    //and calc overflowed

assert 256g*256*256*256 * 256*256*256*256 ==
18446744073709551616
    //no overflow here because BigInteger
introduced in first value
```

We can also increment and decrement BigIntegers:

```
def a= 7g
assert a++ == 7g && a == 8g && a-- == 8g &&
a == 7g &&
    ++a == 8g && a == 8g && --a == 7g &&
a == 7g
```

We can find out the quotient and remainder:

```

assert 7g.divide( 4g ) == 1g
assert 7g.remainder( 4g ) == 3g
def a= 7g.divideAndRemainder( 4g )
assert a[0] == 1g //quotient, same result as
divide()
assert a[1] == 3g //remainder, same result
as remainder()

assert 7g.divide( -4g ) == -1g
assert 7g.remainder( -4g ) == 3g
assert (-7g).divide( 4g ) == -1g
assert (-7g).remainder( 4g ) == -3g
    //division of a negative yields a
negative (or zero) remainder
assert (-7g).divide( -4g ) == 1g
assert (-7g).remainder( -4g ) == -3g

```

Other methods for arithmetic:

```

assert 22g.intdiv(5g) == 4g
assert (-22g).intdiv(5g) == -4g

assert 7g.abs() == 7g //absolute value
assert (-7g).abs() == 7g

assert 28g.gcd(35g) == 7g
    //greatest common divisor of absolute
value of each number

assert (-28g).gcd(35g) == 7g

```

```

assert 28g.gcd(-35g) == 7g
assert (-28g).gcd(-35g) == 7g
assert 0g.gcd(9g) == 9g
assert 0g.gcd(0g) == 0g

assert 4g**3 == 64g //raising to the power
assert (4g**3).class == Integer
    //raising to the power converts a
    BigInteger to an integer

assert 4g.power(3) == 64g //using method
assert 4g.pow(3) == 64g
    //pow() is different to, and sometimes
    slower than, power()
assert (-4g).power(3) == -64g
assert 4g.power(0) == 1g //exponent must be
integer >=0

assert 7g % 4g == 3g && 7g.mod( 4g ) == 3g
    //modulo arithmetic, using operator or
    method
assert 8g % 4g == 0g
assert -7g % 4g == 1g
    //result of mod is between 0 and
    (modulus - 1) inclusive
try{ 7g % -4g; assert 0 }catch(e){ assert e
instanceof ArithmeticException }
    //mod value must be positive

assert 4g.modPow( 3g, 9g ) == 1
    //calculates as ((4**3) mod 9), result

```

```

always zero or positive
assert 4g.modPow( -2g, 9g ) == 4
    //negative exponents allowed, but mod
value must be positive

assert 4g.modInverse( 3g ) == 1 //calculates
as ((4**(-1) mod 3)
    //mod must be positive, and value must
have a multiplicative inverse mod m
    //(ie, be relatively prime to m)

assert 7g.max(5g) == 7g //maximum and
minimum
assert 4g.min(5g) == 4g
def a=5g, b=5g, c=a.min(b)
assert [a,b].any{ c.is(it) }
    //either a or b may be returned if
they're both equal

assert (-45g <=> -43g) && (
(-45g).compareTo( -43g ) == -1 )
    //comparing two BigIntegers
assert 14g >= 8g          &&
14g.compareTo(8g) >= 0

assert 45g.signum() == 1 //return sign as

```

```
-1,0, or 1  
assert 0g.signum() == 0  
assert (-43g).signum() == -1
```

We can construct a randomly generated positive BigInteger with a specified bit length (at least 2 bits), that is probably prime to a specific certainty. The probability the BigInteger is prime is  $>(1 - (1/2)^{\text{certainty}})$ . If the certainty  $\leq 0$ , true always returned. The execution time is proportional to the value of this parameter. We must pass in a new Random object:

```

100.times{
  def primes= [17g, 19g, 23g, 29g, 31g]
    //bitlength is 5, so primes from 16 to
31 incl
  assert new BigInteger( 5, 50, new Random()
) in primes
    //5 is bit-length, 50 is certainty
(must be integer)
}

def pp= BigInteger.probablePrime( 20, new
Random() )
  //if we don't want to specify certainty
  //20 is bit-length; there's <1.0e-30
chance the number isn't prime

def pn= pp.nextProbablePrime()
  //this is probably next prime, but
definitely no primes skipped over
( (pp+1)..<pn ).each{
  assert ! it.isProbablePrime(50)
    //we can test for primality to
specific certainty (here, 50).
    //True if probably prime, false if
definitely composite
}
assert 10g.nextProbablePrime() == 11
assert 0g.nextProbablePrime() == 2

```

All operations behave as if `BigIntegers` were represented in two's-complement notation.

Bit operations operate on a single bit of the two's-complement representation of their operand/s. The infinite word size ensures that there are infinitely many virtual sign bits preceding each `BigInteger`. None of the single-bit operations can produce a `BigInteger` with a different sign from the `BigInteger` being operated on, as they affect only a single bit.

```
assert 0x33g.testBit(1)
    //true if bit is 1, indexing beginning
at 0 from righthand side
assert ! 0x33g.testBit(2)
(2..100).each{
    assert (-0x3g).testBit(it)
    //negative BigIntegers have virtual
infinite sign-extension
}
```

Unlike with fixed-width integers, `BigIntegers` don't have a method to show the hex, octal, or binary representation of a negative number. We can use this code instead to look at the first 16 lowest-order virtual bits:

```
def binRepr={n->
  (15..0).inject(''){flo,it->
    flo<< (n.testBit(it)? 1: 0)
  }
}
assert 0x33g.toString(2) == '110011'
assert binRepr(0x33g) as String ==
'0000000000110011'
assert (-0x33g).toString(2) == '-110011'
//not what we want to see
assert binRepr(-0x33g) as String ==
'1111111111001101'
  //notice the negative sign bit extended
  virtually
```

More bit-manip methods:

```

assert 0x33g.setBit(6) == 0x73g //0x33g is
binary 110011
assert 0x33g.clearBit(4) == 0x23g
assert 0x33g.flipBit(1) == 0x31g
assert 0x33g.flipBit(2) == 0x37g

assert 0x1g.getLowestSetBit() == 0
    //index of the rightmost one bit in this
BigInteger
assert 0x2g.getLowestSetBit() == 1
assert 0x8g.getLowestSetBit() == 3
assert 0x33g.bitLength() == 6
    //number of bits in minimal
representation of number
assert (-0x33g).bitLength() == 6 //exclude
sign bit
assert 0x33g.bitCount() == 4 //number of
bits that differ from sign bit
assert (-0x33g).bitCount() == 3

```

Setting, clearing, or flipping bit in virtual sign makes that bit part of the number:

```

assert (-0x33g).clearBit(9) == -0x233g

```

We can perform bit-shifting on BigIntegers. The shortcut operators >> and << can't be used, only the method names can be (they're also spelt differently to the fixed-size integer versions of the names, eg, "shiftLeft" instead of "leftShift"). There's no shift-right-unsigned method because this doesn't make sense for BigIntegers with virtual infinite-length sign bits.

```

assert 0xB4Fg.shiftLeft( 4 ) == 0xB4F0g
//shift 4 bits to the left
assert 0xD23Cg.shiftRight( 4 ) == 0xD23g
    //shift 4 bits to the right, dropping
off digits
assert (-0xFFFg).shiftRight( 4 ) == -0x100g
    //sign-extension performed when
right-shifting
[ 0xABCg, -0x98765g ].each{
    it.shiftLeft( 8 ) == it.shiftRight( -8 )
}

```

We can perform 'not', 'and', 'or', and 'xor' bitwise operations on BigIntegers:

```

assert 123g.not() == -124g //in
2's-complement, negate and add 1
assert -0xFFg.not() == 0x100g

assert ( (0x33g & 0x11g) == 0x11g)  &&
0x33g.and(0x11g) == 0x11g
assert ( (0x33g | 0x11g) == 0x33g)  &&
0x33g.or(0x11g) == 0x33g
assert ( (0x33g ^ 0x11g) == 0x22g)  &&
0x33g.xor(0x11g) == 0x22g
assert 0x33g.andNot(0x11g) == 0x22g &&
(0x33g & (~ 0x11g)) == 0x22g
    //convenience operation

```

For negative numbers:

```
//and returns a negative if both operands  
are negative...
```

```
assert (-1g & -1g) == -1g
```

```
//or returns a negative number if either  
operand is negative...
```

```
assert (1g | -1g) == -1g
```

```
//xor returns a negative number if exactly  
one operand is negative...
```

```
assert (1g ^ -1g) == -2g
```

```
assert (-1g ^ -2g) == 1g
```

When the two operands are of different lengths, the sign on the shorter of the two operands is virtually extended prior to the operation:

```
assert 11g.and(-2g) == 10g //01011 and 11110  
is 01010, ie, 10g
```

## JN0525-Decimals

We can only use base-10 notation to represent decimal numbers, not hexadecimal or octal. Decimals are written with a decimal part and/or an exponent part, each with an optional + -. The leading zero is required.

```
[ 1.23e-23, 4.56, -1.7E1, 98.7e2, -0.27e-54  
].each{ assert it } //decimals  
assert (-1.23).class == BigDecimal  
assert (-1.23g).class == BigDecimal  
//BigInteger 'g' suffix after a  
decimal-formatted number means BigDecimal
```

Such `BigDecimal`s are arbitrary-precision signed decimal numbers. They consist of an unscaled infinitely-extendable value and a 32-bit Integer scale. The value of the number represented by it is (`unscaledValue × 10-(scale)`). This means a zero or positive scale is the number of digits to the right of the decimal point; a negative scale is the unscaled value multiplied by ten to the power of the negation of the scale. For example, a scale of -3 means the unscaled value is multiplied by 1000.

We can construct a `BigDecimal` with a specified scale:

```
assert new BigDecimal( 0, 1 ) == 0.0
assert new BigDecimal( 123, 0 ) == 123
assert new BigDecimal( 123 ) == 123
//default scale is 0
assert new BigDecimal( -123, 0 ) == -123
assert new BigDecimal( 123, -1 ) == 1.23e3
assert new BigDecimal( 12, -3 ) == 12000.0
assert new BigDecimal( 120, 1 ) == 12.0
assert new BigDecimal( 123, 5 ) == 0.00123
assert new BigDecimal( -123, 14 ) ==
-1.23e-12

assert (2 as BigDecimal).unscaledValue() ==
2
assert (2 as BigDecimal).scale() == 0
assert (2 as BigDecimal).scale == 0 //parens
optional
assert 2.0.unscaledValue() == 20
assert 2.0.scale == 1
```

All methods and constructors for this class throw `NullPointerException` when passed a null object reference for any input parameter.

We can enquire the scale of a `BigDecimal`:

```
assert (1234.567).unscaledValue() ==
1234567g
    //returns the unscaled portion of a
    BigInteger

assert (1234.567).scale() == 3 //returns the
scale
```

The precision of a BigDecimal is the number of digits in the unscaled value. The precision of a zero value is 1.

```
assert 7.7.precision() == 2
assert (-7.7).precision() == 2
assert 1.000.precision() == 4
```

We can construct a BigDecimal from a string. The value of the resulting scale must lie between Integer.MIN\_VALUE and Integer.MAX\_VALUE, inclusive.

```
assert '23.45'.toBigDecimal() == 23.45
assert new BigDecimal( '23.45' ) == 23.45
assert new BigDecimal( '-32.8e2' ) ==
-32.8e2
assert new BigDecimal( '+.9E-7' ) == 0.9e-7
assert new BigDecimal( '+7.E+8' ) == 7e8
assert new BigDecimal( '0.0' ) == 0.0

try{ new BigDecimal( ' 23.45' ); assert 0 }
catch(e){ assert e instanceof
NumberFormatException } //whitespace in
string
```

If we have the String in a char array and are concerned with efficiency, we can supply that array directly to the BigDecimal:

```
def ca1= ['1', '2', '.', '5'] as char[]
assert new BigDecimal( ca1 ) == 12.5
def ca2= [ 'a', 'b', '9', '3', '.', '4',
'5', 'x', 'y', 'z' ] as char[]

assert new BigDecimal( ca2, 2, 5 ) == 93.45
    //use 5 chars from the array beginning
    from index 2
```

There are some different ways of displaying a BigDecimal:

```
assert 1.2345e7.toString() == '1.2345E+7'
    //one digit before decimal point, if
    exponent used

assert 1.2345e7.toPlainString() ==
'12345000' //no exponent portion
assert 1.2345e7.toEngineeringString() ==
'12.345E+6' //exponent divisible by 3
```

From Java 5.0, every distinguishable BigDecimal value has a unique string representation as a result of using toString(). If that string representation is converted back to a BigDecimal, then the original value (unscaled-scale pair) will be recovered. This means it can be used as a string representation for exchanging decimal data, or as a key in a HashMap.

```
[ 1.2345e7, 98.76e-3, 0.007, 0.000e4 ].each{
    assert new BigDecimal( it.toString() ) ==
it
}
```

## Conversions

We can construct a BigDecimal from integers:

```
assert new BigDecimal( 45i ).scale == 0  
assert new BigDecimal( 45L ).scale == 0
```

If we want to buffer frequently-used BigDecimal values for efficiency, we can use the valueOf() method:

```
def a= BigDecimal.valueOf( 12L, -3 )  
assert a == 12000.0g && a.scale == -3  
  
def b= BigDecimal.valueOf( 12L )  
assert b == 12.0 && b.scale == 0 //default  
scale is 0  
  
assert BigDecimal.ZERO == 0.0 //These  
commonly-used values are pre-supplied  
assert BigDecimal.ONE == 1.0  
assert BigDecimal.TEN == 10.0
```

The BigDecimal can be converted between the BigInteger, Integer, Long, Short, and Byte classes. Numbers converted to fixed-size integers may be truncated, or have the opposite sign.

```
assert 123g as BigDecimal == 123.0
assert 45i as BigDecimal == 45.0
assert 73L as BigDecimal == 73.0
assert 73L.toBigDecimal() == 73.0
//alternative syntax

assert 123.456 as BigInteger == 123g //lost
information about the precision
assert 123.456.toBigInteger() == 123g
//alternative syntax
assert 73.0 as Long == 73g
assert 73.0 as long == 73g
assert 73.0.toLong() == 73g
assert 73.0.longValue() == 73g //another
alternative syntax
assert 45.6789.intValue() == 45g //truncated
assert 259.0.byteValue() == 3 //truncated,
only lowest 8 integral bits returned
assert 200.789.byteValue() == -56
    //truncated, only lowest 8 integral bits
returned, with opposite sign
```

By appending 'Exact' to the asLong()-style method names, we can ensure an ArithmeticException is thrown if any information would be lost in the conversion:

```

assert 123.0.toBigIntegerExact() == 123g
//lost information about the precision
try{ 123.456.toBigIntegerExact(); assert
false }
catch(e){ assert e instanceof
ArithmeticException }

assert 73.0.longValueExact() == 73g

[ { 73.21.longValueExact() },
  { 45.6789.intValueExact() },
  { 73.21.shortValueExact() },
  { 259.0.byteValueExact() },
  { 200.789.byteValueExact() },
].each{
  try{ it(); assert false }catch(e){ assert
e instanceof ArithmeticException }
}

```

#### BigDecimal Arithmetic

We can use the same methods and operators on BigDecimal we use with BigInteger:

```

assert 3.4.plus( 3.3 ) == 3.4 + 3.3
assert 3.4.add( 3.3 ) == 3.4 + 3.3
//alternative name for plus
assert 3.4.minus( 2.1 ) == 3.4 - 2.1
assert 3.4.subtract( 2.1 ) == 3.4 - 2.1
//alternative name for minus
assert 3.0.multiply( 3.1 ) == 3.0 * 3.1
assert 3.0.multiply( 3g ) == 3.0 * 3g
assert 7.7.negate() == -7.7 //unary
operation/method
assert (-7.7).negate() == -(-7.7)
assert (-7.7).plus() == +(-7.7) //this
method provided for symmetry with negate

try{ 3.4.multiply(null); assert 0 }
catch(e){ assert e instanceof
NullPointerException }
    //all BigDecimal methods throw
NullPointerException if passed a null

```

The scale resulting from add or subtract is the maximum scale of each operand; that resulting from multiply is the sum of the scales of the operands:

```

def a = 3.414, b = 3.3
assert a.scale() == 3 && b.scale() == 1
assert (a+b).scale() == 3 //max of 3 and 1
assert (a*b).scale() == 4 //sum of 3 and 1

```

For + - and \*, a BigDecimal with any integer type converts it to a BigDecimal:

```
assert (123.45g * 789).class == BigDecimal  
assert (123.45g * 789L).class == BigDecimal  
assert (123.45g * (89 as byte)).class ==  
BigDecimal
```

We can use a MathContext to change the precision of operations involving BigDecimals:

```

def mc= new java.math.MathContext( 3 )
    //precision of 3 in all constructors and
methods where used
assert new BigDecimal( 123456, 0, mc ) ==
123000g
assert new BigDecimal( -12345, 14, mc ) ==
-1.23e-10
assert new BigDecimal( '23.4567', mc ) ==
23.5
assert new BigDecimal(
    ['2', '3', '.', '4', '5', '6', '7'] as
char[], mc ) == 23.5
assert new BigDecimal(
    ['2', '3', '.', '4', '5', '6', '7'] as
char[], 1, 5, mc ) == 3.46
assert new BigDecimal( 1234i, mc ) == 1230
assert new BigDecimal( 1234L, mc ) == 1230

assert 3.45678.add( 3.3, mc ) == 6.76
assert 0.0.add( 3.333333, mc ) == 3.33
assert 3.4567.subtract( 2.1, mc ) == 1.36
assert 0.0.subtract( 2.12345, mc ) == -2.12
assert 3.0.multiply( 3.1234, mc ) == 9.37
assert (-7.77777).negate( mc ) == 7.78

assert (-7.77777).plus( mc ) == -7.78
    //effect identical to that of
round(MathContext) method

```

We can create BigDecimals by dividing integers, both fixed-size and BigInteger, for which the result is a decimal number:

```
assert 7g / 4g == 1.75
assert (-7g) / 4g == -1.75
assert ( 1 / 2 ).class == BigDecimal
assert ( 1L / 2L ).class == BigDecimal
assert ( 1g / 2g ).class == BigDecimal

assert ( 1.5 * 2g ).class == BigDecimal
    //an expression with a BigDecimal never
converts to an integer

assert 1.0.div( 2 ).class == BigDecimal
    //we can use a method instead of the
operator

try{ 17g / 0; assert 0 }catch(e){ assert e
instanceof ArithmeticException }
    //division by 0 not allowed
```

Sometimes, the division can return a recurring number. This leads to a loss of exactness:

```

assert 1/3 == 0.3333333333
    //BigDecimals with recurring decimals
round their result to 10 places...
assert ( (1/3) * 3 ) != 1
    //...which leads to inaccuracy in
calculations
assert (1/3).precision() == 10
assert 100000/3 == 33333.3333333333
    //accuracy before the decimal point is
always retained

```

When the scales of both operands in division are quite different, we can lose precision, sometimes even completely:

```

assert (1.0 / 7.0) == 0.1428571429
    //instead of "0.142857 with last 6
digits recurring"
assert (1e-5 / 7.0) == 0.0000014286
//precision is 10
assert (1e-9 / 7.0) == 0.0000000001
assert (1e-11 / 7.0) == 0.0
    //difference in scale of operands can
cause full loss of precision

```

The `ulp()` of a `BigDecimal` returns the "Units of the Last Place", the difference between the value and next larger having the same number of digits:

```
assert 123.456.ulp() == 0.001 //always 1,  
but with same scale  
assert 123.456.ulp() == (-123.456).ulp()  
assert 0.00.ulp() == 0.01
```

Another way of dividing numbers is to use the `divide()` method, different to the `div()` method and `/` operator. The result must be exact when using `divide()`, or an `ArithmeticException` is thrown.

```
assert 1.0.divide( 4.0 ) == 0.25  
  
try{ 1.0.divide( 7.0 ); assert 0 }  
catch(e){ assert e instanceof  
ArithmeticException }  
    //result must be exact when using  
divide()  
  
assert 1.234.divide( 4.0 ) == 0.3085  
assert 1.05.divide( 1.25 )  
assert 1.234.scale() == 3 && 4.0.scale() ==  
1 && 0.3085.scale() == 4  
    //scale of result unpredictable  
assert 1.05.scale() == 2 && 1.25.scale() ==  
2 && 0.84.scale() == 2
```

We can change the precision of `divide()` by using a `MathContext`:

```

assert (1.0).divide( 7.0, new
java.math.MathContext(12) ) ==
0.142857142857

//precision is 12

assert (1.0).divide( 7.0, new
java.math.MathContext(10) ) == 0.1428571429

assert (1.0).divide( 7.0, new
java.math.MathContext(5) ) == 0.14286

try{ 1.0.divide( 7.0, new
java.math.MathContext(0) ); assert 0 }
catch(e){ assert e instanceof
ArithmeticException }
    //precision of 0 same as if no
MathContext was supplied

```

#### MathContext Rounding Modes

As well as specifying required precision for operations in a MathContext, we can also specify the rounding behavior for operations discarding excess precision. Each rounding mode indicates how the least significant returned digit of a rounded result is to be calculated.

If fewer digits are returned than the digits needed to represent the exact numerical result, the discarded digits are called "the discarded fraction", regardless their contribution to the value of the number returned. When rounding increases the magnitude of the returned result, it is possible for a new digit position to be created by a carry propagating to a leading 9-digit. For example, the value 999.9 rounding up with three digits precision would become 1000.

We can see the behaviour of rounding operations for all rounding modes:

```

import java.math.MathContext
import java.math.RoundingMode

```

```
//so we don't have to qualify these with
java.math when we refer to them
```

```
import static java.math.RoundingMode.*
```

```
//so we don't have to qualify UP, DOWN,
etc with java.math.RoundingMode
```

```
def values= [ +5.5, +2.5, +1.6, +1.1,
+1.0, -1.0, -1.1, -1.6, -2.5, -5.5 ]
```

```
def results= [
    (UP): [ 6, 3, 2, 2,
1, -1, -2, -2, -3, -6 ],
    (DOWN): [ 5, 2, 1, 1,
1, -1, -1, -1, -2, -5 ],
    (CEILING): [ 6, 3, 2, 2,
1, -1, -1, -1, -2, -5 ],
    (FLOOR): [ 5, 2, 1, 1,
1, -1, -2, -2, -3, -6 ],
    (HALF_UP): [ 6, 3, 2, 1,
1, -1, -1, -2, -3, -6 ],
    (HALF_DOWN): [ 5, 2, 2, 1,
1, -1, -1, -2, -2, -5 ],
    (HALF_EVEN): [ 6, 2, 2, 1,
1, -1, -1, -2, -2, -6 ],
]
```

```
results.keySet().each{ roundMode->
```

```
    def mc= new MathContext( 1, roundMode )
```

```
    results[ roundMode ].eachWithIndex{ it,
i->
```

```
        assert new BigDecimal( values[i], mc )
== it
```

```
    }  
  }
```

```
def mcu= new MathContext( 1, UNNECESSARY )  
assert new BigDecimal( 1.0, mcu ) == 1  
assert new BigDecimal( -1.0, mcu ) == -1  
[ +5.5, +2.5, +1.6, +1.1, -1.1, -1.6, -2.5,  
-5.5 ].each{  
  try{ new BigDecimal( it, mcu ); assert 0 }  
}
```

```
catch(e){ assert e instanceof  
ArithmeticException }  
}
```

We can thus see:

UP rounds away from zero, always incrementing the digit prior to a non-zero discarded fraction.

DOWN rounds towards zero, always truncating.

CEILING rounds towards positive infinity (positive results behave as for UP; negative results, as for DOWN).

FLOOR rounds towards negative infinity (positive results behave as for DOWN; negative results, as for UP).

HALF\_UP rounds towards nearest neighbor; if both neighbors are equidistant, rounds as for UP. (The rounding mode commonly taught in US schools.)

HALF\_DOWN rounds towards nearest neighbor; if both neighbors are equidistant, rounds as for DOWN.

HALF\_EVEN rounds towards the nearest neighbor; if both neighbors are equidistant, rounds towards the even neighbor. (Known as "banker's rounding.")

UNNECESSARY asserts that the operation has an exact result; if there's an inexact result, throws an ArithmeticException.

There are some default rounding modes supplied for use:

```

import java.math.*
    //imports all such classes, including
both MathContext and RoundingMode

MathContext.UNLIMITED
    //for unlimited precision arithmetic
(precision=0 roundingMode=HALF_UP)
MathContext.DECIMAL32
    //for "IEEE 754R" Decimal32 format
(precision=7 roundingMode=HALF_EVEN)
MathContext.DECIMAL64
    //Decimal64 format (precision=16
roundingMode=HALF_EVEN)
MathContext.DECIMAL128
    //Decimal128 format (precision=34
roundingMode=HALF_EVEN)

assert MathContext.DECIMAL32.precision == 7

assert MathContext.DECIMAL32.roundingMode ==
RoundingMode.HALF_EVEN
    //precision and roundingMode are
properties

assert new BigDecimal( 123456789, 0,
MathContext.DECIMAL32 ) == 123456800g

```

Other constructors for MathContext are:

```

import java.math.*
def mc1= new MathContext( 3 )
    //by default, uses RoundingMode.HALF_UP
rounding mode
assert mc1.roundingMode ==
RoundingMode.HALF_UP

def mc2= new MathContext( 3,
RoundingMode.HALF_UP )
assert mc2.toString() == 'precision=3
roundingMode=HALF_UP'
def mc3= new MathContext( mc2.toString() )
    //we can initialize a MathContext from
another's string
assert mc3.precision == 3
assert mc3.roundingMode ==
RoundingMode.HALF_UP

```

The rounding mode setting of a MathContext object with a precision setting of 0 is not used and thus irrelevant.

Cloning BigDecimal but with different scale

We can create a new BigDecimal with the same overall value as but a different scale to an existing one:

```

import java.math.*

def num= 2.2500
assert num.scale == 4 && num.unscaledValue()
== 22500
def num2= num.setScale(5)
assert num2 == 2.25000 && num2.scale == 5 &&

```

```
num2.unscaledValue() == 225000
    //usual use of changing scale is to
increase the scale
def num3= num.setScale(3)
assert num3 == 2.25000 && num3.scale == 3 &&
num3.unscaledValue() == 2250

assert num.setScale(2) == 2.25
    //only BigDecimal returned from method
call has changed scale...
assert num.scale == 4 //...while original
BigDecimal still has old scale...
num.scale= 3 //...so there's no point using
the allowable property syntax
assert num.scale == 4

try{
    num.setScale(1) //we can't change the
value when we reduce the scale...
    assert false
}catch(e){ assert e instanceof
ArithmeticException }
assert 1.125.setScale(2,
RoundingMode.HALF_UP) == 1.13

//...unless we use a rounding mode
```

```
assert 1.125.setScale(2,  
BigDecimal.ROUND_HALF_UP) == 1.13  
//pre-Java-5 syntax
```

These 8 BigDecimal static fields are older pre-Java-5.0 equivalents for the values in the RoundingMode enum:

```
BigDecimal.ROUND_UP  
BigDecimal.ROUND_DOWN  
BigDecimal.ROUND_CEILING  
BigDecimal.ROUND_FLOOR  
BigDecimal.ROUND_HALF_UP  
BigDecimal.ROUND_HALF_DOWN  
BigDecimal.ROUND_HALF_EVEN  
BigDecimal.ROUND_UNNECESSARY
```

There's two methods that let us convert such older names to the newer RoundingMode constants (enums):

```
import java.math.RoundingMode  
assert RoundingMode.valueOf( 'HALF_UP' ) ==  
RoundingMode.HALF_UP  
  
assert RoundingMode.valueOf(  
BigDecimal.ROUND_HALF_DOWN ) ==  
RoundingMode.HALF_DOWN
```

#### Further operations

For the other arithmetic operations, we also usually have the choice of supplying a MathContext or not.

There's two main ways to raise a number to a power. Using \*\* and power() returns a fixed-size floating-point number, which we'll look at in the next topic on Groovy Floating-Point Math.

```
assert (4.5**3).class == Double  
assert 4.5.power(3).class == Double //using  
equivalent method instead
```

We can raise a BigDecimal to the power using the pow() method instead, which always returns an exact BigDecimal. However, this method will be very slow for high exponents. The result can sometimes differ from the rounded result by more than one ulp (unit in the last place).

```
assert 4.5.pow(3) == 91.125 //pow() is
different to power()
assert (-4.5).pow(3) == -91.125
assert 4.5.pow(0) == 1.0
assert 0.0.pow(0) == 1.0
try{ 4.5.pow(-1); assert 0 }catch(e){ assert
e instanceof ArithmeticException }
    //exponent must be integer >=0
try{ 1.1.pow(1000000000); assert 0 }
catch(e){ assert e instanceof
ArithmeticException }
    //exponent too high for Java 5

//println( 1.1.pow(999999999) )
    //warning: this runs for a VERY LONG
time when uncommented
```

When we supply a MathContext, the "ANSI X3.274-1996" algorithm is used:

```

import java.math.MathContext
assert 4.5.pow( 3, new MathContext(4) ) ==
91.13 //can supply a MathContext
assert 4.5.pow( -1, new MathContext(10) )
    //negative exponents allowed when
MathContext supplied
try{ 4.5.pow( -1, new MathContext(0) );
assert 0 }
catch(e){ assert e instanceof
ArithmeticException }
    //ArithmeticException thrown if
mc.precision == 0 and n < 0
try{ 4.5.pow( 123, new MathContext(2) );
assert 0 }
catch(e){ assert e instanceof
ArithmeticException }
    //ArithmeticException thrown if
mc.precision > 0 and
    //n has more than mc.precision decimal
digits

```

Instead of giving a precision via the MathContext, we can give the desired scale directly:

```

import java.math.RoundingMode
assert 25.497.divide( 123.4567, 5,
RoundingMode.UP ) == 0.20653
    //specify desired scale of 4, and
rounding mode UP
assert 25.497.divide( 123.4567, 5,
BigDecimal.ROUND_UP ) == 0.20653
    //cater for pre-Java-5.0 syntax
assert 25.497.divide( 123.4567,
RoundingMode.UP ) == 0.207
    //if no scale given, use same one as
dividend (here, 25.497)
assert 25.497.divide( 123.4567,
BigDecimal.ROUND_UP ) == 0.207

```

We can divide to an integral quotient, and/or find the remainder. (The preferred scale of the integral quotient is the dividend's less the divisor's.)

```

import java.math.*
mc= new MathContext( 9, RoundingMode.HALF_UP
)
assert 25.5.divide( 2.4, mc ) == 10.625

assert 25.5.divideToIntegralValue( 2.4 ) ==
10 //rounding mode always DOWN...
assert 25.5.remainder( 2.4 ) == 1.5
assert 25.5.divideToIntegralValue( 2.4, mc )
== 10
                                     //...even when
a MathContext says otherwise

```

```
assert 25.5.remainder( 2.4, mc ) == 1.5
assert (-25.5).divideToIntegralValue( 2.4,
mc ) == -10
assert (-25.5).remainder( 2.4, mc ) == -1.5

try{ 25.5.divideToIntegralValue( 0 ); assert
0 }
catch(e){ assert e instanceof
ArithmeticException }

try{ 25.5.remainder( 0 ); assert 0 }
catch(e){ assert e instanceof
ArithmeticException }

assert 25.525.remainder( 2.345, new
MathContext(1) ) == 2.075
    //MathContext's precision only affects
quotient calculation;
    //remainder always exact so may have
more decimal digits

[ [25.5, 2.4], [-27.1, 3.3] ].each{ x, y->
    assert x.remainder( y ) ==
        x.subtract( x.divideToIntegralValue( y
).multiply( y ) )
}

try{
    2552.0.divideToIntegralValue( 2.4, new
MathContext(2) )
    assert 0
```

```
}catch(e){ assert e instanceof
ArithmeticException }
    //if result needs more decimal digits
than supplied MathContext's precision

try{
    2552.0.remainder( 2.4, new MathContext(2)
)
    assert 0
}catch(e){ assert e instanceof
ArithmeticException }
    //throw if implicit
divideToIntegralValue() result needs more
decimal digits
    //than supplied MathContext's precision

def qr= 25.5.divideAndRemainder( 2.4 )
```

```
assert qr[0] == 10 && qr[1] == 1.5
    //same results as divideToIntegralValue()
and remainder(), but more efficient
```

We can find the absolute value of a BigDecimal:

```
import java.math.*
assert 7.89.abs() == 7.89 //same scale if no
MathContext
assert (-7.89).abs() == 7.89
assert (-7.89).abs( new MathContext(2) ) ==
7.9
```

The round() operation only has a version with a MathContext parameter. Its action is identical to that of the plus(MathContext) method.

```
assert 7.89.round( new MathContext(2) ) ==
7.9
assert 7.89.round( new MathContext(0) ) ==
7.89 //no rounding if precision is 0
```

#### *Operations without a MathContext*

Not all BigDecimal operations have a MathContext.

Auto-incrementing and -decrementing work on BigDecimals:

```
def a= 12.315
a++
assert a == 13.315
--a
assert a == 12.315
```

The signum method:

```
assert 2.34.signum() == 1  
assert (-2.34).signum() == -1  
assert 0.0.signum() == 0
```

As with integers, we can compare BigDecimals:

```
assert (2.50 <=> 2.5) == 0 &&  
2.50.compareTo(2.5) == 0  
assert (-3.45 <=> 1.23) == -1 &&  
(-3.45).compareTo(1.23) == -1  
assert (1.23 <=> -0.12) == 1 &&  
1.23.compareTo(-0.12) == 1  
assert (1.23 > -0.12) &&  
1.23.compareTo(-0.12) > 0
```

The equals() method and == operator are different for BigDecimals. (So we must be careful if we use BigDecimal objects as elements in a SortedSet or keys in a SortedMap, since BigDecimal's natural ordering is inconsistent with equals().)

```
assert ! ( 2.00.equals(2.0) )  
    //considers whether both unscaledValue  
and scale are equal  
assert 2.00 == 2.0 //only considers the  
sequence of the two numbers on a line  
  
assert 0.0 == -0.0 && 0.0.equals( -0.0 )
```

We can find the minimum and maximum of two BigDecimals:

```
assert (-2.0).min( 7.3 ) == -2.0
assert 3.5.max( 4.2 ) == 4.2
```

We can move the decimal point to the left or right:

```
import java.math.*
def num= 123.456
assert num.scale == 3
def mpl= num.movePointLeft( 2 )
assert mpl.scale == 5 //scale should be max(
number.scale + movement, 0 )
assert mpl == 1.23456
def mpr= num.movePointRight( 4 )
assert mpr.scale == 0 //scale should be max(
number.scale - movement, 0 )
assert mpr == 1234560
assert( 3.456.movePointLeft(2) == 0.03456 )
[ -2, -1, 0, 1, 2 ].each{
  assert 123.456.movePointLeft( it ) ==
123.456.movePointRight( -it )
}
try{ //throw ArithmeticException if scale
will overflow on moving decimal point
  new BigDecimal( 123456, 128*256*256*256 -
1 ).movePointLeft( 1 )
  assert 0
}catch(e){ assert e instanceof
ArithmeticException }
```

Another method for moving the decimal point, but by consistent change to the scale:

```
import java.math.*
def num= 123.456
assert num.scale == 3
def mpl= num.scaleByPowerOfTen( 16 )
assert mpl == 1.23456e18
assert mpl.scale == -13 //num.scale - 16
```

We can strip trailing zeros:

```
assert 45.607000.stripTrailingZeros() ==
45.607
assert 600.0.stripTrailingZeros() == 6e2
assert new BigDecimal( 6000, 1
).stripTrailingZeros() == new BigDecimal( 6,
-2 )
```

### JN0535-Floats

As well as BigDecimal, decimals can have type Float or Double. Unlike BigDecimal which has no size limit, Float and Double are fixed-size, and thus more efficient in calculations. BigDecimal stores its value as base-10 digits, while Float and Double store their values as binary digits. So although using them is more efficient in calculations, the result of calculations will not be as exact as in base-10, eg,  $3.1f + 0.4f$  computes to 3.499999910593033, instead of 3.5.

We can force a decimal to have a specific type other than BigDecimal by giving a suffix (F for Float, D for Double):

```
assert 1.200065d.class == Double
assert 1.234f.class == Float
assert (-1.23E23D).class == Double
assert (1.167g).class == BigDecimal
    //although g suffix here is optional, it
makes examples more readable
```

We can enquire the minimum and maximum values for Floats and Doubles:

```
assert Float.MIN_VALUE == 1.4E-45f
assert Float.MAX_VALUE == 3.4028235E38f
assert Double.MIN_VALUE == 4.9E-324d
assert Double.MAX_VALUE ==
1.7976931348623157E308d
```

We can represent infinities by using some predefined constants (prefixed by either Float or Double):

```
assert (1f / 0f) == Double.POSITIVE_INFINITY
assert (-1f / 0f) ==
Double.NEGATIVE_INFINITY
assert Double.POSITIVE_INFINITY ==
Float.POSITIVE_INFINITY

assert 0.0f != -(0.0f)
    //positive and negative zeroes not
equal, when negative is written -(0.0f)
assert 0.0f == -0.0f
    //but when negative is written -0.0f,
it's evaluated as positive
```

If a nonzero Double literal is too large or too small, it's represented by Double.POSITIVE\_INFINITY or Double.NEGATIVE\_INFINITY or 0.0:

```
assert Double.MAX_VALUE * Double.MAX_VALUE
== Double.POSITIVE_INFINITY
assert Double.MIN_VALUE * Double.MIN_VALUE
== 0.0d
assert -Double.MAX_VALUE * Double.MAX_VALUE
== Double.NEGATIVE_INFINITY
assert -Double.MAX_VALUE * -Double.MAX_VALUE
== Double.POSITIVE_INFINITY
```

Classes Float and Double can both be written uncapitalized, ie, float and double.

```
assert Float.TYPE == float
assert Double.TYPE == double
```

There's a special variable called Double.NaN (and Float.NaN), meaning "Not a Number", which is sometimes returned from math calculations. Once introduced into a math calculation, the result will (usually) be NaN.

#### Conversions

The Float and Double classes, along with BigDecimal, BigInteger, Integer, Long, Short, and Byte, can all be converted to one another.

Converting numbers to integers may involve rounding or truncation:

```
assert 45.76f as int == 45i //truncated
assert 45.76d as int == 45i
assert 45.76f.toInteger() == 45i //method
name
assert 45.76f.toLong() == 45L
assert 200.8f as byte == -56 as byte //sign
reversed after truncation
assert 45.76f.toBigInteger() == 45
```

Converting from integers to float or double (may involve rounding):

```
assert 789g as Float == 789f
assert 45i.toFloat() == 45f //method name
assert 789g.toFloat() == 789f
assert 789g.floatValue() == 789f
//alternative method name
assert 45i as double == 45d
assert 6789g.toDouble() == 6789d //method
name
assert 6789g.doubleValue() == 6789d
//alternative method name

assert new BigInteger( '1' + '0'*40
).floatValue() == Float.POSITIVE_INFINITY
//one with 40 zeroes after it
assert new BigInteger( '1234567890' * 3
).floatValue() == 1.2345679e29f
//precision lost on conversion
```

Converting from BigDecimal to float or double (may involve rounding):

```

assert 89.980 as float == 89.98f
assert 1.432157168 as float == 1.4321572f
//rounded
assert 78.9g.toFloat() == 78.9f
assert 456.789g.floatValue() == 456.789f
assert 6.789g.toDouble() == 6.789d
assert 2345.6789g.doubleValue() ==
2345.6789d
assert new BigDecimal( '-' + '1' *45
).floatValue() == Float.NEGATIVE_INFINITY
assert new BigDecimal( '0.' + '0'*45 + '1'
).floatValue() == 0.0f
assert new BigDecimal( '0.' + '1234567890'
*3 ).floatValue() == 0.12345679f

//precision lost on conversion

```

We can convert a double to a float. but there's no Double() constructor accepting a float as an argument.

```

assert 23.45e37d as float == 23.45e37f
assert new Float( 23.45e37d ) == 23.45e37f
assert new Float( 23.45e67d ) ==
Float.POSITIVE_INFINITY
assert 123.45e12f as double //conversion
inexact

```

We can create a Float or Double from a string representation of the number, either base-10 or hex:

```
[ '77', '1.23e-23', '4.56', '-1.7E1',
```

```

'98.7e2', '-0.27e-30' ].each{
  assert it.toFloat()
  assert new Float(it)
  assert it.toDouble()
  assert new Double(it)
}
assert new Float( ' 1.23e-23 ' )
//leading and trailing whitespace removed
try{ new Float( null ); assert 0 }
catch(e){ assert e instanceof
NullPointerException }
[ 'NaN', '-NaN', 'Infinity', '-Infinity',
'+Infinity' ].each{
  assert new Float(it)
}

assert new Float( ' -0Xabc.defP7' )
  //we can have hexadecimal mantissa, with
P indicating exponent
assert new Float( ' 0xABC.DEFp17 ' )
  //part after P must be base-10, not more
hex
assert new Float( '0X.defP-3f \n' )
  //any whitespace OK (spaces, tabs,
newlines, carriage returns, etc)
try{ new Float( ' @0X6azQ/3d' ); assert 0 }
catch(e){ assert e instanceof
NumberFormatException }
  //because the string doesn't contain a
parsable number in the form of a Float
assert Float.valueOf( '0xABp17' )

```

```
//alternate means of constructing float  
from string representation  
assert Float.parseFloat( '0xABp17' )  
    //another alternate means of constructing  
float from string  
assert new Double( '0x12bc.89aP7d  ' )
```

The string is first converted to a double, then if need be converted to a float.

Converting from double to BigDecimal is only exact when the double has an exact binary representation, eg, 0.5, 0.25. If a float is supplied, it's converted to a double first, then given to the BigDecimal constructor. The scale of the returned BigDecimal is the smallest value such that  $(10^{**scale} * val)$  is an integer.

```
assert new BigDecimal(0.25d) == 0.25
    //exact conversion because 0.25 has an
exact binary representation

assert new BigDecimal(0.1d) ==

0.10000000000000000005551115123125782702118158
3404541015625

(0.1d).toBigDecimal() == new
BigDecimal(0.1d) //alternative method name

assert new BigDecimal(0.1f) ==
0.100000001490116119384765625
    //inexact conversion as 0.1 has a
recurring decimal part in binary

assert (0.1f as BigDecimal) ==
0.100000001490116119384765625

assert new BigDecimal(0.1d, new
java.math.MathContext(25) ) ==
    0.100000000000000000055511151 //rounds to
25 places as specified
```

A more exact way to convert a double to a BigDecimal:

```

assert BigDecimal.valueOf( 0.25d ) == 0.25
assert BigDecimal.valueOf( 0.1d ) == 0.1
    //always exact, because converts double
to a string first
assert new BigDecimal( Double.toString( 0.1d
) ) == 0.1
    //explicitly convert double to string,
then to BigDecimal
assert BigDecimal.valueOf( -23.456e-17d ) ==
-2.3456E-16
assert BigDecimal.valueOf( -23.456e-17f ) ==
-2.3455999317674643E-16
    //result inexact because float converted
to double first

try{ BigDecimal.valueOf(
Double.POSITIVE_INFINITY ); assert 0 }
catch(e){ assert e instanceof
NumberFormatException }

try{ BigDecimal.valueOf( Double.NaN );
assert 0 }
catch(e){ assert e instanceof
NumberFormatException }
    //however, infinities and NaN won't
convert that way

```

We can convert a float or double to a unique string representation in base-10. There must be at least one digit to represent the fractional part, and beyond that as many, but only as many, more digits as are needed to uniquely

distinguish the argument value from adjacent values of type float. (The returned string must be for the float value nearest to the exact mathematical value supplied; if two float representations are equally close to that value, then the string must be for one of them and the least significant bit of the mantissa must be 0.)

```
assert Float.toString( 3.0e6f ) ==  
'3000000.0' //no leading zeros  
assert Float.toString( 3.0e0f ) == '3.0'  
//at least one digit after the point  
assert Float.toString( 3.0e-3f ) == '0.0030'  
assert Float.toString( 3.0e7f ) == '3.0E7'  
    //exponent used if it would be > 6 or <  
-3  
assert Float.toString( 3.0e-4f ) == '3.0E-4'  
//mantissa >= 1 and < 10
```

We can also convert a float or double to a hexadecimal string representation:

```

[
    0.0f: '0x0.0p0',
    (-0.0f): '0x0.0p0', //no negative sign
in hex string rep'n of -0.0f
    1.0f: '0x1.0p0', //most returned
strings begin with '0x1.' or '-0x1.'
    2.0f: '0x1.0p1',
    3.0f: '0x1.8p1',
    5.0f: '0x1.4p2',
    (-1.0f): '-0x1.0p0',
    0.5f: '0x1.0p-1',
    0.25f: '0x1.0p-2',
    (Float.MAX_VALUE):
'0x1.fffffep127',
    (Float.MIN_VALUE):
'0x0.000002p-126',
                                //low values
beginning with '0x0.' are called 'subnormal'
    (Float.NEGATIVE_INFINITY): '-Infinity',
    (Float.NaN): 'NaN',
].each{ k, v->
    assert Float.toHexString(k) == v
}

```

We can format integers and decimals using `String.format()`:

```

//Integers ('d')
assert String.format('%d', 45) == '45'
assert String.format('%5d,%1$5o', 46L) == '
46,    56'
    //octal format; each minimum 5 chars

```

```

wide; use an argument twice
assert String.format('%-4d,%<-5x', 47g) ==
'47  ,2f  '
    //hex format without leading '0x';
left-justified with '-';
    //shortcut ('<') for using argument
again
assert String.format('%2d,%<1X', 123) ==
'123,7B'
    //hex in uppercase with capital 'X'
assert String.format('%04d', 34) == '0034'
//zero-pad
assert String.format('% ,5d', 12345) ==
'12,345' //use grouping-separators
assert String.format('%+3d,%2$ 3d', 123L,
456g) == '+123, 456'
    //always use plus sign; always use a
leading space
assert String.format('%(3d', -789 as short)
== '(789)' //parens for negative
assert String.format('%(3o,%2$(3x,%3$(3X',
123g, 456g, -789g) == '173,1c8,(315)'
    //neg octal/hex only for BigInteger

//Floating-Point ('f', 'a', 'e', 'g')
assert String.format('e = %f', Math.E) == 'e
= 2.718282'
    //default 'f' format is 7.6
assert String.format('e=%+6.4f', Math.E) ==
'e=+2.7183'
    //precision is digits after decimal

```

```
point
assert String.format('$ %(,6.2f', -6217.58)
== '$ (6,217.58)'
    //'(' flag gives parens, ',' uses
separators
assert String.format('%a, %A', 2.7182818f,
Math.PI) ==
    '0x1.5bf0a8p1, 0X1.921FB54442D18P1'
// 'a' for hex
assert String.format('%+010.4a', 23.25d) ==
'+0x001.7400p4'
    //'+' flag always includes sign; '0'
flag zero-fills
assert String.format('%e, %10.4e', Math.E,
12345.6789) ==
    '2.718282e+00, 1.2346e+04' // 'e' for
scientific format
assert String.format('%(10.5E', -0.0000271)
== '(2.71000E-05)'
```

```
assert String.format('%g, %10.4G', Math.E,
12345.6789) == '2.71828, 1.235E+04'
    //'f' or 'e', depending on input
```

#### Floating-Point Arithmetic

We can perform the same basic operations that integers and BigDecimal can:

```
assert 3.4f.plus( 3.3f ) == 3.4f + 3.3f
assert 3.4f.minus( 2.1f ) == 3.4f - 2.1f
assert 3.0f.multiply( 3.1f ) == 3.0f * 3.1f
assert 3.0f.multiply( 3f ) == 3.0f * 3f
assert 3.0.multiply( 3f ) == 3.0 * 3f
assert 7.7f.negate() == -7.7f //unary
operation/method
assert (-7.7f).negate() == -(-7.7f)
assert +(7.7f) == 7.7f

try{ 3.4f.multiply(null); assert false }
catch(e){ assert e instanceof
NullPointerException }
    //methods throw NullPointerException if
passed a null
```

For + - and \*, anything with a Double or Float converts both arguments to a Double:

```
assert (23.4f + 7.998d).class == Double
assert (23.4f - 123.45g).class == Double
assert (7.998d * 123.45g).class == Double
assert (23.4f - i=789).class == Double
```

We can divide using floats and doubles:

```
assert 2.4f.div( 1.6f ) == ( 2.4f / 1.6f )
assert ( 2.5f / 1i ).class == Double
    //produces double result if either
operand is float or double
assert ( 2.5f / 1.25 ).class == Double
```

We can perform mod on floats and doubles:

```
def a= 34.56f % 5
assert a == 34.56f.mod(5) && a < 5.0f && a
>= 0.0f

def b= 34.56f % 5.1f
assert b == 34.56f.mod(5.1f) && b < 5.0f &&
b >= 0.0f

def c= -34.56f % 5.1f
assert c == (-34.56f).mod(5.1f) && c <= 0.0f
&& c > -5.0f
```

IEEEremainder resembles mod in some ways:

```

def Infinity=Double.POSITIVE_INFINITY,
NaN=Double.NaN, Zero=0.0d
assert Math.IEEEremainder( 33d, 10d ) == 3d
//give
remainder after rounding to nearest value
assert Math.IEEEremainder( 37d, 10d ) == -3d
assert Math.IEEEremainder( -33d, 10d ) ==
-3d
assert Math.IEEEremainder( -37d, 10d ) == 3d
assert Math.IEEEremainder( 35d, 10d ) == -5d
//when two
values equally near, use even number
assert Math.IEEEremainder( 45d, 10d ) == 5d
assert Math.IEEEremainder( Zero, 10d ) ==
Zero
assert Math.IEEEremainder( -Zero, 10d ) ==
-Zero
assert Math.IEEEremainder( Infinity, 10d )
== NaN
assert Math.IEEEremainder( 35d, Zero ) ==
NaN
assert Math.IEEEremainder( 35d, Infinity )
== 35d

```

We can perform other methods:

```
assert (-23.4f).abs() == 23.4f
assert (-23.414d).abs() == 23.414d

assert 14.49f.round() == 14i
assert 14.5f.round() == 15i
assert (-14.5f).round() == -14i
assert 14.555d.round() == 15L
```

We can raise a float or double to a power:

```
assert 4.5f**3 == 91.125d //double returned
assert 4.5f.power(3) == 4.5f**3 //using
equivalent method instead
assert 1.1.power(1000000000) ==
Double.POSITIVE_INFINITY
```

We can test whether a float or double is a number and whether it's an infinite number:

```
def Infinity=Double.POSITIVE_INFINITY,
NaN=Double.NaN, Zero=0.0d
assert NaN.isNaN()
assert Double.isNaN( NaN )
assert Infinity.isInfinite()
assert (-Infinity).isInfinite()
assert Double.isInfinite( Infinity )
assert Double.isInfinite( -Infinity )
assert Float.isInfinite(
Float.NEGATIVE_INFINITY )
```

We can test whether two floats or doubles have equal values using operators or methods:

```
assert 345f.equals( 3.45e2f ) && 345f ==
3.45e2f
    //equals() and == behave the same in all
cases
assert ! 34.5f.equals( 13.4f ) && 34.5f !=
13.4f //equivalent

assert Float.NaN.equals( Float.NaN ) &&
Float.NaN == Float.NaN

assert 0.0f == -0.0f && 0.0f.equals( -0.0f )
    //-0.0f is evaluated as positive zero
assert 0.0f != -(0.0f) && ! 0.0f.equals(
-(0.0f) )
    //negative zero must be written -(0.0f)

assert 345d.equals( 3.45e2d ) && 345d ==
3.45e2d

assert Float.POSITIVE_INFINITY.equals(
Float.POSITIVE_INFINITY ) &&
    Float.POSITIVE_INFINITY ==
Float.POSITIVE_INFINITY
assert ! Float.POSITIVE_INFINITY.equals(
Float.NEGATIVE_INFINITY ) &&
    ! ( Float.POSITIVE_INFINITY ==
Float.NEGATIVE_INFINITY )
```

We can compare floats and doubles using the <=> operator, the compareTo() method, and the compare() static method:

```

assert (2.50f <=> 2.5f) == 0 &&
2.50f.compareTo(2.5f) == 0
assert (-3.45f <=> 1.23f) == -1 &&
(-3.45f).compareTo(1.23f) == -1
assert (1.23d <=> -0.12d) == 1 &&
1.23d.compareTo(-0.12d) == 1
assert (-1.23d < -0.12d) &&
(-1.23d).compareTo(-0.12d) < 0

assert (Float.NaN > Float.POSITIVE_INFINITY)
&&

Float.NaN.compareTo(Float.POSITIVE_INFINITY)
> 0
assert (0.0f <=> -0.0f) == 0
assert (Float.NaN <=> Float.NaN) == 0 &&
Float.NaN.compareTo(Float.NaN) == 0
assert Float.compare( 3.4f, 7.9f ) == -1
assert Double.compare( 3.4d, -7.9d ) == 1

```

Auto-incrementing and -decrementing work on floats and doubles:

```

def a= 12.315d
a++
assert a == 13.315d
--a
assert a == 12.315d

```

Non-zero floats and doubles evaluate as true in boolean contexts:

```
assert (1.23d? true: false)
assert ! (0.0f? true: false)
```

#### Bitwise Operations

We can convert a float to the equivalent int bits, or a double to equivalent float bits. For a float, bit 31 (mask 0x80000000) is the sign, bits 30-23 (mask 0x7f800000) are the exponent, and bits 22-0 (mask 0x007fffff) are the mantissa. For a double, bit 63 is the sign, bits 62-52 are the exponent, and bits 51-0 are the mantissa.

```
assert Float.floatToIntBits( 0.0f ) == 0
assert Float.floatToIntBits( 15.15f ) ==
0x41726666
assert Float.floatToIntBits( Float.NaN ) ==
0x7fc00000
assert Float.floatToIntBits(
Float.POSITIVE_INFINITY ) == 0x7f800000
assert Float.floatToIntBits(
Float.NEGATIVE_INFINITY ) == (0xff800000 as
int)
assert Double.doubleToLongBits( 15.15d ) ==
0x402e4ccccccccc
```

The methods `floatToRawIntBits()` and `doubleToRawLongBits()` act similarly, except that they preserve Not-a-Number (NaN) values. So if the argument is NaN, the result is the integer or long representing the actual NaN value produced from the last calculation, not the canonical `Float.NaN` value to which all the bit patterns encoding a NaN can be collapsed (ie, 0x7f800001 through 0x7fffffff and 0xff800001 through 0xffffffff).

The `intBitsToFloat()` and `longBitsToDouble()` methods act oppositely. In all cases, giving the integer resulting from calling `Float.floatToIntBits()` or `Float.floatToRawIntBits()` to the `intBitsToFloat(int)` method will produce the original floating-point value, except for a few NaN values. Similarly with doubles. These methods are the only operations that can distinguish between two NaN values of the same type with different bit patterns.

```
assert Float.intBitsToFloat( 0x7fc00000 ) ==
Float.NaN
assert Float.intBitsToFloat( 0x7f800000 ) ==
Float.POSITIVE_INFINITY
assert Float.intBitsToFloat( 0xff800000 as
int ) == Float.NEGATIVE_INFINITY
assert Float.intBitsToFloat( 0 ) == 0.0f
assert Float.intBitsToFloat( 0x41726666 ) ==
15.15f
assert Double.longBitsToDouble(
0x402e4cccccccccd ) == 15.15d
assert Float.intBitsToFloat(
Float.floatToIntBits( 15.15f ) ) == 15.15f
```

As well as infinities and NaN, both Float and Double have other constants:

```
assert Float.MAX_VALUE ==
Float.intBitsToFloat(0x7f7fffff)
assert Float.MIN_NORMAL ==
Float.intBitsToFloat(0x00800000)
    //the smallest positive nonzero normal
value
assert Float.MIN_VALUE ==
Float.intBitsToFloat(0x1)
    //the smallest positive nonzero value,
including subnormal values
assert Float.MAX_EXPONENT ==
Math.getExponent(Float.MAX_VALUE)
assert Float.MIN_EXPONENT ==
Math.getExponent(Float.MIN_NORMAL)
assert Float.MIN_EXPONENT ==
Math.getExponent(Float.MIN_VALUE) + 1
    //for subnormal values
```

#### Floating-Point Calculations

There are two constants of type Double, Math.PI and Math.E, that can't be represented exactly, not even as a recurring decimal.

The trigonometric functions behave as expected with the argument in radians, but 0.0 isn't represented exactly. For example, sine:

```
assert Math.sin( 0.0 ) == 0.0
assert Math.sin( 0.5 * Math.PI ) == 1.0
assert Math.sin( Math.PI ) < 1e-15 //almost
0.0, but not quite
assert Math.sin( 1.5 * Math.PI ) == -1.0
assert Math.sin( 2 * Math.PI ) > -1e-15
//almost 0.0
assert Math.sin( -0.5 * Math.PI ) == -1.0
assert Math.sin( -Math.PI ) > -1e-15
//almost 0.0
assert Math.sin( -1.5 * Math.PI ) == 1.0
assert Math.sin( -2 * Math.PI ) < 1e-15
//almost 0.0
assert Math.sin( Double.POSITIVE_INFINITY )
== Double.NaN
assert Math.sin( Double.NEGATIVE_INFINITY )
== Double.NaN
```

Other trig functions are:

```
assert Math.cos( Double.POSITIVE_INFINITY )  
== Double.NaN  
assert Math.tan( Double.NEGATIVE_INFINITY )  
== Double.NaN  
assert Math.asin( 0.0 ) == 0.0  
assert Math.asin( 1.0 ) == 0.5 * Math.PI  
assert Math.asin( 1.001 ) == Double.NaN  
assert Math.acos( -1.0 ) == Math.PI  
assert Math.acos( -1.001 ) == Double.NaN  
assert Math.atan( 0.0 ) == 0.0
```

Some logarithmic functions:

```
def Infinity= Double.POSITIVE_INFINITY, NaN=
Double.NaN, Zero= 0.0d
```

```
[ (Infinity): Infinity,
    10000: 4,
    10: 1,
    1: 0,
    0.1: -1,
    0.00001: -5,
    0.0: -Infinity,
    (-0.001): NaN,
].each{ k, v -> assert Math.log10(k) == v }
//returns base-10 logarithm
```

```
[ (Infinity): Infinity,
    (Math.E): 1,
    1: 0,
    0.0: -Infinity,
    (-0.001): NaN,
].each{ k, v -> assert Math.log(k) == v }
//returns natural logarithm
```

```
assert Math.exp( Infinity ) == Infinity
//returns Math.E raised to a power
assert Math.exp( -Infinity ) == 0.0
```

Math.ulp(d) returns the size of the units of the last place for doubles (the difference between the value and the next larger in magnitude).

```

assert Math.ulp( 123.456d ) == Math.ulp(
-123.456d )
assert Math.ulp( 0.123456789d ) != Math.ulp(
0.123456789f )
    //if Float, a different scale is used

assert Math.ulp( Double.POSITIVE_INFINITY )
== Double.POSITIVE_INFINITY
assert Math.ulp( Double.NEGATIVE_INFINITY )
== Double.POSITIVE_INFINITY
assert Math.ulp( 0.0d ) == Double.MIN_VALUE
assert Math.ulp( Double.MIN_VALUE ) ==
Double.MIN_VALUE
assert Double.MAX_VALUE > Math.ulp(
Double.MAX_VALUE )


```

Accuracy of the Math methods is measured in terms of such ulps for the worst-case scenario. If a method always has an error less than 0.5 ulps, the method always returns the floating-point number nearest the exact result, and so is always correctly rounded. However, doing this and maintaining floating-point calculation speed together is impractical. Instead, for the Math class, a larger error bound of 1 or 2 ulps is allowed for certain methods. But most methods with more than 0.5 ulp errors are still required to be semi-monotonic, ie, whenever the mathematical function is non-decreasing, so is the floating-point approximation, and vice versa. Not all approximations that have 1 ulp accuracy meet the monotonicity requirements. sin, cos, tan, asin, acos, atan, exp, log, and log10 give results within 1 ulp of the exact result that are semi-monotonic.

#### Further Calculations

We can find the polar coordinate of two (x,y) coordinates. The result is within 2 ulps of the exact result, and is semi-monotonic.

```

def Infinity= Double.POSITIVE_INFINITY, NaN=
Double.NaN, Zero= 0.0d

[
    [ 1d, 1d ]: 0.25d * Math.PI,
    [ 1d, -1d ]: 0.75d * Math.PI,


```

```

    [ -1d, 1d ]: -0.25d * Math.PI,
    [ -1d, -1d ]: -0.75d * Math.PI,

    [ 0d, 1d ]: 0d,
    [ -(0d), 1d ]: -(0d),
    [ 0d, -1d ]: Math.PI,
    [ -(0d), -1d ]: -Math.PI, // -(0d) gives
huge difference in result to 0d
    [ 1d, 0d ]: 0.5d * Math.PI,
    [ 1d, -(0d) ]: 0.5d * Math.PI,
    [ -1d, 0d ]: -0.5d * Math.PI,
    [ -1d, -(0d) ]: -0.5d * Math.PI,

    [ Double.NaN, 1d ]: Double.NaN, //NaN
returned if either argument is NaN

    [ 1d, Infinity ]: 0d,
    [ 1d, -Infinity ]: Math.PI,
    [ -1d, Infinity ]: -(0d),
    [ -1d, -Infinity ]: -Math.PI,
    [ Infinity, 1d ]: 0.5d * Math.PI,
    [ Infinity, -1d ]: 0.5d * Math.PI,
    [ -Infinity, 1d ]: -0.5d * Math.PI,
    [ -Infinity, -1d ]: -0.5d * Math.PI,
    [ Infinity, Infinity ]: 0.25d * Math.PI,
    [ Infinity, -Infinity ]: 0.75d * Math.PI,
    [ -Infinity, Infinity ]: -0.25d *
Math.PI,
    [ -Infinity, -Infinity ]: -0.75d *
Math.PI,
].each{k,v->

```

```
if( Math.atan2( k[0], k[1] ) != v )
```

```
println "( ${k[0]}, ${k[1]} ):  
${Math.atan2(k[0],k[1])}; $v"  
}
```

We can perform the hyperbolic trigonometric functions:

```

assertClose= {it1,it2,ulp->
    assert it1 > it2 - ulp*Math.ulp(it2) &&
it1 < it2 + ulp*Math.ulp(it2)
}
def Infinity=Double.POSITIVE_INFINITY,
Zero=0d, NaN=Double.NaN, E=Math.E
assertClose Math.sinh( 2d ), 0.5d*(E**2d -
E**-2d), 2.5d
    //sinh() result will be with 2.5 ulp of
exact value
assert Math.sinh( Infinity ) == Infinity
assert Math.sinh( -Infinity ) == -Infinity
assert Math.sinh( Zero ) == Zero
assert Math.sinh( -Zero ) == -Zero
assertClose Math.cosh( 2d ), 0.5d*(E**2d +
E**-2d), 2.5d
assert Math.cosh( Infinity ) == Infinity
assert Math.cosh( -Infinity ) == Infinity
assert Math.cosh( Zero ) == 1d
assert Math.cosh( -Zero ) == 1d
assertClose Math.tanh( 2d ), Math.sinh( 2d
)/Math.cosh( 2d ), 2.5d
assert Math.tanh( Infinity ) == 1d
assert Math.tanh( -Infinity ) == -1d
assert Math.tanh( Zero ) == Zero
assert Math.tanh( -Zero ) == -Zero
    //once the exact result of tanh is
within 1/2 of an ulp of
    //the limit value of +/- 1, a correctly
signed +/- 1.0 will be returned

```

We can convert between degrees and radians. The conversion is generally inexact.

```
assert Math.toDegrees( Math.PI ) == 180.0  
assert Math.toRadians( 90.0 ) == 0.5 *  
Math.PI
```

We can calculate  $(E^x)-1$  (1 + x) in one call. For values of x near 0, `Math.expm1( x ) + 1d` is much closer than `Math.exp( x )` to the true result of  $e^x$ . The result will be semi-monotonic, and within 1 ulp of the exact result. Once the exact result of  $e^x - 1$  is within 1/2 ulp of the limit value -1, -1d will be returned.

```
assertClose= {it1,it2,ulp->  
  assert it1 > it2 - ulp*Math.ulp(it2) &&  
  it1 < it2 + ulp*Math.ulp(it2)  
}  
def Infinity=Double.POSITIVE_INFINITY,  
NaN=Double.NaN, Zero= 0d, E= Math.E  
  
assertClose Math.expm1( 123.4d ), E**123.4d  
- 1, 31  
assertClose Math.expm1( 23.4d ), E**23.4d -  
1, 10  
assertClose Math.expm1( 3.4d ), E**3.4d - 1,  
3  
  
assert Math.expm1( Infinity ) == Infinity  
assert Math.expm1( -Infinity ) == -1d  
assert Math.expm1( Zero ) == Zero  
assert Math.expm1( -Zero ) == -Zero
```

We can also calculate  $\ln(1 + x)$  in one call. For small values of x, `Math.log1p( x )` is much closer than `Math.log(1d + x)` to the true result of  $\ln(1 + x)$ . The result will be semi-monotonic, and within 1 ulp of the exact result.

```

def Infinity=Double.POSITIVE_INFINITY,
NaN=Double.NaN, Zero= 0d
assert Math.log1p( 123.4d ) == Math.log(1d +
123.4d)
assert Math.log1p( 23.4d ) == Math.log(1d +
23.4d)
assert Math.log1p( 3.4d ) == Math.log(1d +
3.4d)
assert Math.log1p( -1.1d ) == NaN
assert Math.log1p( Infinity ) == Infinity
assert Math.log1p( -1d ) == -Infinity
assert Math.log1p( Zero ) == Zero
assert Math.log1p( -Zero ) == -Zero

```

Scale binary `scalb(x,y)` calculates  $(x * y^{**2})$  using a single operation, giving a more accurate result. If the exponent of the result would be larger than `Float/Double.MAX_EXPONENT`, an infinity is returned. If the result is subnormal, precision may be lost. When the result is non-NaN, the result has the same sign as `x`.

```

def Infinity= Double.POSITIVE_INFINITY, NaN=
Double.NaN, Zero= 0.0d
assert Math.scalb(5, 3) == 5 * 2**3
assert Math.scalb(NaN, 3) == NaN
assert Math.scalb(Infinity, 3) == Infinity
//same sign
assert Math.scalb(Zero, 3) == Zero //same
sign

```

We have square root and cube root methods. For `cbrt`, the computed result must be within 1 ulp of the exact result.

```

def ten= Math.sqrt( 10 ) * Math.sqrt( 10 )
def error= 1e-14
assert ten > 10 - error && ten < 10 + error

assert Math.sqrt( -0.001 ) == Double.NaN
assert Math.sqrt( 0 ) == 0
assert Math.sqrt( Double.POSITIVE_INFINITY )
== Double.POSITIVE_INFINITY

def ten= Math.cbrt( 10 ) * Math.cbrt( 10 ) *
Math.cbrt( 10 )
def error= 1e-14
assert ten > 10 - error && ten < 10 + error

assert Math.cbrt( -123.456 ) == -Math.cbrt(
123.456 )
assert Math.cbrt( 0 ) == 0
assert Math.cbrt( Double.POSITIVE_INFINITY )
== Double.POSITIVE_INFINITY
assert Math.cbrt( Double.NEGATIVE_INFINITY )
== Double.NEGATIVE_INFINITY

```

We can find the ceiling and floor of doubles:

```
assert Math.ceil( 6.77 ) == 7 &&
Math.floor( 6.77 ) == 6
assert Math.ceil( -34.43 ) == -34 &&
Math.floor( -34.43 ) == -35
assert Math.ceil( 0.73 ) == 1.0 &&
Math.floor( 0.73 ) == 0.0
assert Math.ceil( -0.73 ) == -0.0d &&
Math.floor( -0.73 ) == -1.0
    //sign required for -0.0d
assert Math.ceil( 13.0 ) == 13.0 &&
Math.floor( 13.0 ) == 13.0
assert Math.ceil( 0.0 ) == 0.0 &&
Math.floor( 0.0 ) == 0.0
assert Math.ceil( 23.45 ) == -Math.floor(
-23.45 )
    //Math.ceil(x) always equals
-Math.floor(-x)
```

We can round doubles to the nearest long (or floats to the nearest integer). The calculation is `Math.floor(a + 0.5d)` as Long, or `Math.floor(a + 0.5f)` as Integer

```

[
    7.45: 7,
    7.5: 8,
    (-3.95): -4,
    (-3.5): -3,
    (Double.NaN): 0,
    (Double.NEGATIVE_INFINITY):
Long.MIN_VALUE,
    (Long.MIN_VALUE as Double):
Long.MIN_VALUE,
    (Double.POSITIVE_INFINITY):
Long.MAX_VALUE,
    (Long.MAX_VALUE as Double):
Long.MAX_VALUE,
].each{ k, v -> assert Math.round( k ) == v
}

```

Unlike the numerical comparison operators, `max()` and `min()` considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

```

assert Math.max( 7i, 9i ) == 9i //returns
the same class as its arguments
assert Math.min( 23L, 19L ) == 19L
assert Math.min( 1.7f, 0.3f ) == 0.3f
assert Math.min( -6.7d, 1.3d ) == -6.7d
assert Math.min( 7i, 9L ) == 7L //converts
result to most precise type of argument
assert Math.min( 1L, 3.3f ) == 1f
assert Math.min( -6.7f, 1.3d ) ==
-6.699999809265137d

```

Some other methods:

```
[    7.49d: 7.0d,  
      7.5d: 8.0d,  
      8.5d: 8d,  
    (-7.5d): -8d,  
      7d: 7d,  
      0d: 0d,  
    (Double.POSITIVE_INFINITY):  
Double.POSITIVE_INFINITY,  
].each{ k, v-> assert Math rint( k ) == v }  
    //round to nearest integer (or even  
integer)
```

```
assert Math.abs( -23i ) == 23i  
assert Math.abs( 234L ) == 234L  
assert Math.abs( 0i ) == 0i  
assert Math.abs( Integer.MIN_VALUE ) ==  
Integer.MIN_VALUE
```

//WARNING:

```
this result not intuitive  
assert Math.abs( Long.MIN_VALUE ) ==  
Long.MIN_VALUE
```

```
assert Math.abs( -23.45f ) == 23.45f  
assert Math.abs( -123.4d ) == 123.4d  
assert Math.abs( 0.0f ) == 0.0f  
assert Math.abs( -0.0f ) == 0.0f  
assert Math.abs( Float.NEGATIVE_INFINITY )  
== Float.POSITIVE_INFINITY
```

```
[ -23.45f, 781.23f, Float.NEGATIVE_INFINITY
].each{
  assert Math.abs(it) ==
    Float.intBitsToFloat(0x7fffffff &
Float.floatToIntBits(it))
  assert Math.abs(it) ==
```

```

Float.intBitsToFloat((Float.floatToIntBits(i
t)<<1)>>>1)
} //there's related assertions for doubles

```

The pow() method returns the value of the first argument raised to the power of the second argument. If both arguments are integers, then the result is exactly equal to the mathematical result of raising the first argument to the power of the second argument if that result can in fact be represented exactly as a double value. Otherwise, special rules exist for processing zeros and infinities:

```

def Infinity= Double.POSITIVE_INFINITY, NaN=
Double.NaN
[
  [ 3d, 0d ]: 1d,
  [ 3d, -(0d) ]: 1d,
  [ 3d, 1d ]: 3d,
  [ 3d, Infinity ]: Infinity,
  [ -3d, Infinity ]: Infinity,
  [ 0.3d, -Infinity ]: Infinity,
  [ -0.3d, -Infinity ]: Infinity,
  [ 3d, -Infinity ]: 0d,
  [ -3d, -Infinity ]: 0d,
  [ 0.3d, Infinity ]: 0d,
  [ -0.3d, Infinity ]: 0d,
  [ 1d, Infinity ]: Double.NaN,
  [ 0d, 1d ]: 0d,
  [ Infinity, -1d ]: 0d,
  [ 0d, -1d ]: Infinity,
  [ Infinity, 1d ]: Infinity,
  [ -(0d), 2d ]: 0d, //exponent >0 but not
finite odd integer
  [ -Infinity, -2d ]: 0d, //exponent <0 but
not finite odd integer

```

```

    [ -(0d), 3d ]: -(0d), //exponent is
positive finite odd integer
    [ -Infinity, -3d ]: -(0d), //exponent is
negative finite odd integer
    [ -(0d), -2d ]: Infinity, //exponent <0
but not finite odd integer
    [ -Infinity, 2d ]: Infinity, //exponent >0
but not finite odd integer
    [ -(0d), -3d ]: -Infinity, //exponent is
negative finite odd integer
    [ -Infinity, 3d ]: -Infinity, //exponent
is positive finite odd integer
    [ -3d, 4i ]: {-> def a= Math.abs(-3d);
a*a*a*a }(),
                                                    //exponent
is finite even integer
    [ -3d, 5i ]: {-> def a= Math.abs(-3d);
-a*a*a*a*a }(),
                                                    //exponent
is finite odd integer
    [ -3d, 2.5 ]: NaN, //exponent is finite
and not an integer
    [ NaN, 0d ]: 1d //exception to the NaN
ripple rule

```

```
].each{k, v->  
  assert Math.pow( k[0], k[1] ) == v  
}
```

More methods:

```
assert Math.random() >= 0d //this method  
uses new Random() when first called  
assert Math.random() < 1d
```

```
assert Math.signum( 17.75d ) == 1d  
assert Math.signum( 17.75f ) == 1f  
assert Math.signum( -19.5d ) == -1d  
assert Math.signum( 0d ) == 0d  
assert Math.signum( -(0d) ) == -(0d)
```

We can use `copySign()` to return a first argument with the sign of the second argument.

```
assert Math.copySign( 34.4f, -2.1f ) ==  
-34.4f  
assert Math.copySign( -1234.4d, 2.23d ) ==  
1234.4d
```

We can compute the hypotenuse with risk of intermediate overflow (or underflow). The computed result is within 1 ulp of the exact result. If one parameter is held constant, the results will be semi-monotonic in the other parameter.

```
def Infinity=Double.POSITIVE_INFINITY,  
NaN=Double.NaN  
assert Math.hypot( 9d, 16d ) == Math.sqrt(  
9d**2 + 16d**2 )  
assert Math.hypot( Infinity, 234d ) ==  
Infinity  
assert Math.hypot( NaN, 234d ) == NaN  
assert Math.hypot( Infinity, NaN ) ==  
Infinity
```

We can get the exponent from the binary representation of a double or float:

```

def Infinity=Double.POSITIVE_INFINITY,
Zero=0d, NaN=Double.NaN, E=Math.E
assert Math.getExponent(2.345e31d) <=
Double.MAX_EXPONENT
assert Math.getExponent(2.345e31d) >=
Double.MIN_EXPONENT
assert Math.getExponent( NaN ) ==
Double.MAX_EXPONENT + 1
assert Math.getExponent( Infinity ) ==
Double.MAX_EXPONENT + 1

assert Math.getExponent( Zero ) ==
Double.MIN_EXPONENT - 1
    //this is also the value of subnormal
exponents

assert Math.getExponent(12.3e4f) <=
Float.MAX_EXPONENT &&
    Math.getExponent(12.3e4f) >=
Float.MIN_EXPONENT

```

We can return the floating point number adjacent to the first arg in the direction of the second arg:

```

def Infinity=Double.POSITIVE_INFINITY,
NaN=Double.NaN, Zero= 0d
assert Math.nextAfter( 12.34d, 999d ) ==
12.34d + Math.ulp( 12.34d )
assert Math.nextAfter( 12.34d, -999d ) ==
12.34d - Math.ulp( 12.34d )
assert Math.nextAfter( 12.34f, 999f ) ==

```

```
12.34f + Math.ulp( 12.34f )
assert Math.nextAfter( 12.34d, 12.34d ) ==
12.34d
    //if numbers equal, return second one
assert Math.nextAfter( Zero, -Zero ) ==
-Zero
    //numbers are 'equal', and second one
returned
assert Math.nextAfter( Double.MIN_VALUE,
-12d ) == Zero
assert Math.nextAfter( -Double.MIN_VALUE,
12d ) == -Zero
assert Math.nextAfter( Double.MAX_VALUE,
Infinity ) == Infinity
assert Math.nextAfter( -Double.MAX_VALUE,
-Infinity ) == -Infinity
assert Math.nextAfter( Infinity, 12d ) ==
Double.MAX_VALUE
assert Math.nextAfter( -Infinity, 12d ) ==
-Double.MAX_VALUE
assert Math.nextAfter( Zero, Infinity ) ==
Double.MIN_VALUE
assert Math.nextAfter( Infinity, Infinity )
== Infinity
assert Math.nextUp( 12.34d ) ==
```

```
Math.nextAfter( 12.34d, Infinity )  
    //shortcut method for both doubles and  
floats
```

The result is NaN if the argument is NaN for `ulp`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `exp`, `log`, `log10`, `sqrt`, `cbrt`, `IEEEremainder`, `ceil`, `floor`, `rint`, `atan2`, `abs`, `max`, `min`, `signum`, `sinh`, `cosh`, `tanh`, `expm1`, `log1p`, `nextAfter`, and `nextUp`.

But not so with `pow`, `round`, `hypot`, `copySign`, `getExponent`, and `scalb`.

There's another math library called `StrictMath` that's a mirror of `Math`, with exactly the same methods. However, some methods (eg, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `exp`, `log`, `log10`, `cbrt`, `atan2`, `pow`, `sinh`, `cosh`, `tanh`, `hypot`, `expm1`, and `log1p`) follow stricter IEEE rules about what values must be returned. For example, whereas the `Math.copySign` method usually treats some NaN arguments as positive and others as negative to allow greater performance, the `StrictMath.copySign` method requires all NaN sign arguments to be treated as positive values.

### JN0545-Dates

We use class `Date` for simple date processing:

```
def today= new Date() //represents the date  
and time when it is created  
println today  
  
//we can add to and subtract from a date...  
def tomorrow= today + 1,  
    dayAfter= today + 2,  
    yesterday= today - 1,  
    dayBefore= today - 2  
println  
"\n$dayBefore\n$yesterday\n$today\n$tomorrow  
\n$dayAfter\n"  
assert today + 7 == today.plus(7) && today -  
15 == today.minus(15)  
    //equivalent methods  
  
//we can increment and decrement a date...
```

```
def d= today.clone()
d++; assert d == tomorrow
d= d.next(); assert d == dayAfter
//equivalent method
d--; assert d == tomorrow
d= d.previous(); assert d == today
//equivalent method

//we can compare dates...
assert tomorrow.after(today)
assert yesterday.before(today)
assert tomorrow.compareTo(today) > 0
assert tomorrow.compareTo(dayAfter) < 0
assert dayBefore.compareTo(dayBefore) == 0

def n= today.time //we can convert a Date to
a Long
println n
today.time = 0 //long 0 is beginning of 1
Jan 1970 GMT
println today
```

```
def sometimeAgo= new Date(0) //we can
construct a date with a Long argument
assert sometimeAgo == today
```

Other date and time processing can be done using the `GregorianCalendar` class:

```
System.setProperty('user.timezone', 'GMT')
//we'll look at timezones later

def c= new GregorianCalendar()
println c.time // 'time' property gives a
Date class
c= Calendar.instance
assert c.class == GregorianCalendar
//another way to create a GregorianCalendar
println c.time
assert c.timeInMillis == c.time.time
    //we can get the time in milliseconds
after 1 Jan 1970 at 0:00:00am GMT

println System.currentTimeMillis() //another
way to get the current time
println System.nanoTime()
    //time in nano-seconds: good for
measuring elapsed computation times

c= new GregorianCalendar(2009,
Calendar.JULY, 22) //creates 22 July 2009
c= new GregorianCalendar(2009,
Calendar.JULY, 22, 2, 35)
    //creates 22 July 2009 at 2:35am GMT
```

```
c= new GregorianCalendar(2009,  
Calendar.JULY, 22, 2, 35, 21)  
    //creates 22 July 2009 at 2:35:21am GMT  
  
c.clear() //if we clear the fields, we  
get...  
assert c.get(Calendar.ERA) ==  
GregorianCalendar.AD &&  
    c.get(Calendar.YEAR) == 1970 &&  
    c.get(Calendar.MONTH) == 0 &&  
        //dates range from  
0 to 11, so this is January  
    c.get(Calendar.WEEK_OF_MONTH) == 1 &&  
//should be: 0  
    c.get(Calendar.DAY_OF_MONTH) == 1 &&  
    c.get(Calendar.DATE) == 1 && //same  
as DAY_OF_MONTH  
    c.get(Calendar.DAY_OF_WEEK) == 5 &&  
    c.get(Calendar.DAY_OF_WEEK_IN_MONTH)  
== 1 &&  
    c.get(Calendar.AM_PM) == Calendar.AM  
&&  
  
    c.get(Calendar.HOUR) == 0 &&  
    c.get(Calendar.HOUR_OF_DAY) == 0 &&  
    c.get(Calendar.MINUTE) == 0 &&  
    c.get(Calendar.SECOND) == 0 &&  
    c.get(Calendar.MILLISECOND) == 0 &&  
  
    c.get(Calendar.WEEK_OF_YEAR) == 1 &&  
    c.get(Calendar.DAY_OF_YEAR) == 1
```

```
def d= new GregorianCalendar()  
d.timeInMillis= 0  
    //we can set the 'time', here 1 Jan 1970
```

```
at 00:00:00.000 GMT (Gregorian)
d.time= new Date(0) //alternative syntax
assert d == c
```

GregorianCalendar supports both the Julian and Gregorian calendar systems, supporting one discontinuity, which by default is when the Gregorian calendar was first instituted in some countries, ie, 4 October 1582 (Julian) followed by 15 October, 1582 (Gregorian). The only difference between the calendars is the leap year rule: the Julian specifies leap years every four years, whereas the Gregorian omits century years which are not divisible by 400. Because dates are computed by extrapolating the current rules indefinitely far backward and forward in time, this calendar generates consistent results for all years, although dates obtained are historically accurate only from March 1, 4 AD onward, when modern Julian calendar rules were adopted. Although New Year's Day was March 25 prior to the institution of the Gregorian calendar, to avoid confusion, this calendar always uses January 1.

From Groovy 1.5.7 / 1.6.x, you may use Date.format() directly. Refer to [GROOVY-3066](#) for details.

Alternatively, Dates and times can be formatted easily with String.format(). The first character is 't' or 'T' for each item:

```
def c1= new GregorianCalendar(1995,
Calendar.SEPTEMBER, 5, 19, 35, 30, 750)

//dates...
assert String.format('%tY/%<tm/%<td', c1) ==
'1995/09/05'
assert String.format('%tA %<te %<tB %<ty',
c1) == 'Tuesday 5 September 95'
assert String.format('century:%tC,
month:%<tb, day:%<te', c1) ==
'century:19, month:Sep, day:5'
assert String.format('month:%th, day of
year:%<tj, day of week:%<ta', c1) ==
'month:Sep, day of year:248, day of
week:Tue' //'h' same as 'b'

//times...
assert String.format('%tH:%<tM:%<tS.%<tL',
```

```

c1) == '19:35:30.750'
assert String.format('%tI%<tp, %<t1%<tp,
nanoseconds:%<tN', c1) ==
    '07pm, 7pm, nanoseconds:750000000'
assert String.format('%ts', c1) ==
'810300930'
    //seconds since start of 1-Jan-1970 GMT
assert String.format('%tQ', c1) ==
'810300930750'
    //milliseconds since start of 1-Jan-1970
GMT
assert String.format('%tk',
    new GregorianCalendar(1995,
Calendar.SEPTEMBER, 5, 6, 35)) == '6'

//shortcut formats...
assert String.format('%tF', c1) ==
'1995-09-05' //date as '%tm/%td/%ty'
assert String.format('%tD', c1) ==
'09/05/95' //date as '%tY-%tm-%td'
assert String.format('%tT', c1) ==
'19:35:30' //time as '%tH:%tM:%tS'
assert String.format('%tR', c1) == '19:35'
//time as '%tH:%tM'
assert String.format('%tr', c1) == '07:35:30
PM' //time as '%tI:%tM:%tS %Tp'

//additionally...
assert String.format('%tF', new Date(0)) ==
'1970-01-01'
    //we can supply a Date instead of a

```

## Calendar

```
assert String.format('%tF', 0L) ==  
'1970-01-01' //we can also supply a long
```

```
assert String.format('...%15tF...', 0L) ==  
'...      1970-01-01...' //width 15
```

```
assert String.format('...%-15tF...', 0L) ==  
'...1970-01-01      ...'  
    // '-' flag to left-justify
```

After setting fields, we must call any `get()`, `add()`, or `roll()` method, or access the `'timeInMillis'` or `'time'` properties, to cause other relevant fields to update themselves:

```
System.setProperty('user.timezone', 'GMT')
```

```
def c= new GregorianCalendar()  
c.set( Calendar.ERA, GregorianCalendar.AD )  
c.set( Calendar.YEAR, 1949 )  
c.set( Calendar.MONTH, Calendar.OCTOBER )  
c.set( Calendar.DATE, 31 )  
assert String.format('%tF %<ta', c) ==  
'1949-10-31 Mon'
```

```
//properties for calculating WEEK_OF_YEAR  
and WEEK_OF_MONTH fields...  
c.firstDayOfWeek = Calendar.SUNDAY //Sunday  
in most countries, Monday in others  
c.minimalDaysInFirstWeek = 1
```

```
assert c.get(Calendar.ERA) ==  
GregorianCalendar.AD &&  
    c.get(Calendar.YEAR) == 1949 &&  
    c.get(Calendar.MONTH) == 9 && //dates  
range from 0 to 11, so October  
    c.get(Calendar.MONTH) ==  
Calendar.OCTOBER && //alternatively
```

```
        c.get(Calendar.DAY_OF_MONTH) == 31 &&
        c.get(Calendar.WEEK_OF_YEAR) == 45 &&
//range from 1 to 53
        c.get(Calendar.WEEK_OF_MONTH) == 6 &&
//range from 1 to 6
        c.get(Calendar.DAY_OF_YEAR) == 304 &&
        c.get(Calendar.DAY_OF_WEEK) == 2 &&
//Monday
        c.get(Calendar.DAY_OF_WEEK_IN_MONTH)
== 5

//changing the month uses the same year and
day of month...
c.set(Calendar.MONTH, Calendar.AUGUST )
c.time //cause other fields to update
themselves
assert String.format('%tF %<ta', c) ==
'1949-08-31 Wed'

c.set(Calendar.MONTH, Calendar.APRIL )
    //...but may cause adjustment to roll
into following month
c.time
assert String.format('%tF %<ta', c) ==
'1949-05-01 Sun'

c.set(Calendar.DATE, 31 )
c.set(Calendar.MONTH, Calendar.FEBRUARY )
c.set(Calendar.MONTH, Calendar.SEPTEMBER )

//rolling into following month only occurs
```

```
when other fields update themselves,  
//call this method to trigger it...  
c.time  
  
assert String.format('%tF %<ta', c) ==  
'1949-10-01 Sat'  
    //...so Feb-28 DIDN'T roll into Mar-03  
  
//changing the day of month uses the same  
month and year...  
c.set( Calendar.DATE, 1 ); c.time  
assert String.format('%tF %<ta', c) ==  
'1949-10-01 Sat'  
  
//changing the day of year adjusts the  
month, day, and other date fields...  
c.set(Calendar.DAY_OF_YEAR,  
c.get(Calendar.DAY_OF_YEAR) + 2 ); c.time  
assert String.format('%tF %<ta', c) ==  
'1949-10-03 Mon'  
  
//changing the week of year keeps the same  
day of week, but adjusts  
//the other date fields...  
c.set(Calendar.WEEK_OF_YEAR,  
c.get(Calendar.WEEK_OF_YEAR) + 3 ); c.time  
assert String.format('%tF %<ta', c) ==  
'1949-10-24 Mon'  
  
//changing the week of month keeps both the  
same month and day of week...
```

```
c.set(Calendar.WEEK_OF_MONTH,  
c.get(Calendar.WEEK_OF_MONTH) - 2 ); c.time  
assert String.format('%tF %<ta', c) ==  
'1949-10-10 Mon'
```

```
//changing the day of week in month also  
keeps both the
```

```
//same month and day of week...
```

```
c.set(Calendar.DAY_OF_WEEK_IN_MONTH,  
c.get(Calendar.DAY_OF_WEEK_IN_MONTH) - 1 )  
c.time  
assert String.format('%tF %<ta', c) ==  
'1949-10-03 Mon'
```

```
//changing the day of week keeps the same  
week in year...
```

```
c.set( Calendar.DAY_OF_WEEK,  
Calendar.SATURDAY ); c.time  
assert String.format('%tF %<ta', c) ==  
'1949-10-08 Sat'
```

```
c.set( Calendar.DAY_OF_WEEK, Calendar.SUNDAY
```

```
); c.time
assert String.format('%tF %<ta', c) ==
'1949-10-02 Sun'
```

We can also set the time in this way:

```
System.setProperty('user.timezone', 'GMT')
def c= new GregorianCalendar( 1949,
Calendar.OCTOBER, 2 )

c.set( Calendar.AM_PM, Calendar.AM )
c.set( Calendar.HOUR, 6 ) //set the AM_PM
and HOUR fields...
c.set( Calendar.MINUTE, 30 )
c.set( Calendar.SECOND, 15 ); c.time
assert String.format('%tF %<tT', c) ==
'1949-10-02 06:30:15'
assert c.get( Calendar.HOUR_OF_DAY ) == 6
//...and the HOUR_OF_DAY field is
updated...

c.set( Calendar.HOUR_OF_DAY, 19 ); c.time
assert String.format('%tF %<tT', c) ==
'1949-10-02 19:30:15'
assert c.get( Calendar.HOUR ) == 7 && c.get(
Calendar.AM_PM ) == Calendar.PM

//...and vice versa

c.set( Calendar.AM_PM, Calendar.AM ); c.time
assert String.format('%tF %<tT', c) ==
```

```
'1949-10-02 07:30:15' &&
    c.get( Calendar.AM_PM ) == Calendar.AM

c.set( Calendar.HOUR, 18 ); c.time
    //if we set the HOUR with a 24-hr value,
it self-adjusts
assert c.get( Calendar.HOUR ) == 6 && c.get(
Calendar.AM_PM ) == Calendar.PM

//there's no 24:00, only 00:00 which is
'am', on the following day...
c= new GregorianCalendar(1950,
Calendar.JANUARY, 26, 23, 59)
assert String.format( '%tF %<tT %<tp', c ) ==
'1950-01-26 23:59:00 pm'
c.add( Calendar.MINUTE, 1 )
assert String.format( '%tF %<tT %<tp', c ) ==
'1950-01-27 00:00:00 am'

//12:00 noon is 'pm'...
c= new GregorianCalendar(1950,
```

```
Calendar.JANUARY, 27, 12, 00)
assert String.format('%tF %<tT %<tp', c) ==
'1950-01-27 12:00:00 pm'
```

More field manipulations:

```
System.setProperty('user.timezone', 'GMT')

//we can set common fields using terser
syntax...
def c= new GregorianCalendar()
c.set( 1947, Calendar.AUGUST, 11 ); c.time
assert String.format('%tF %<ta', c) ==
'1947-08-11 Mon'
c.set( 1947, Calendar.AUGUST, 12, 6, 30 );
c.time
assert String.format('%tF %<ta', c) ==
'1947-08-12 Tue'
c.set( 1947, Calendar.AUGUST, 15, 6, 30, 45
); c.time
assert String.format('%tF %<ta', c) ==
'1947-08-15 Fri'

//we can clear individual fields, and check
if they're set...
assert c.isSet( Calendar.YEAR ) && c.isSet(
Calendar.MONTH )
c.clear( Calendar.YEAR )
assert ! c.isSet( Calendar.YEAR ) &&
c.isSet( Calendar.MONTH )
```

```

//we can check different maximums and
minimums of a field...
c.set( 1947, Calendar.APRIL, 11 ); c.time
assert c.getMinimum( Calendar.DATE ) == 1 &&
    c.getMaximum( Calendar.DATE ) == 31
assert c.getActualMinimum( Calendar.DATE )
== 1 &&
    c.getActualMaximum( Calendar.DATE ) ==
30
assert c.getGreatestMinimum( Calendar.DATE )
== 1 &&
    c.getLeastMaximum( Calendar.DATE ) == 28

//the first week in a year may be numbered
as part of the previous year,
//and in a month as 0...
c.firstDayOfWeek = Calendar.SUNDAY
c.minimalDaysInFirstWeek = 1
c.set( 1954, Calendar.JANUARY, 1 ); c.time
assert String.format( '%tF %<ta', c ) ==
'1954-01-01 Fri'
assert c.get( Calendar.WEEK_OF_YEAR ) == 1
assert c.get( Calendar.WEEK_OF_MONTH ) == 1

assert c.firstDayOfWeek == Calendar.SUNDAY
&&
    c.minimalDaysInFirstWeek == 1
c.firstDayOfWeek = Calendar.MONDAY
c.minimalDaysInFirstWeek = 4 //trigger
different week numbering
assert c.get( Calendar.WEEK_OF_YEAR ) == 53

```

```
assert c.get(Calendar.WEEK_OF_MONTH) == 0
```

```
c.set( 1956, Calendar.DECEMBER, 31 ); c.time
```

```
assert String.format('%tF %<ta', c) ==
```

```
'1956-12-31 Mon'
```

```
assert c.get(Calendar.WEEK_OF_YEAR) == 1
    //last week of year may be numbered as
    first of next
```

We can compare dates:

```
c1= new GregorianCalendar(2008,
Calendar.AUGUST, 8)
c2= new GregorianCalendar(2009,
Calendar.JULY, 22)
assert c1.before( c2 ) && c2.after( c1 )
assert c1.compareTo( c2 ) < 0 &&
    c2.compareTo( c1 ) > 0 &&
    c1.compareTo( c1 ) == 0
```

As well as using set(), calendar fields can be changed using add() and roll(), both of which force all fields to update themselves:

```

def c= new GregorianCalendar(1999,
Calendar.AUGUST, 31)
assert String.format('%tF %<ta', c) ==
'1999-08-31 Tue'
c.add(Calendar.MONTH, 13)
assert String.format('%tF %<ta', c) ==
'2000-09-30 Sat'

//we DON'T roll to Oct-01

c= new GregorianCalendar(1999,
Calendar.AUGUST, 31)
c.roll(Calendar.MONTH, 13) //rolls a field
without changing larger fields
assert String.format('%tF %<ta', c) ==
'1999-09-30 Thu'
c.roll(Calendar.MONTH, true) //rolls +1
assert String.format('%tF %<ta', c) ==
'1999-10-30 Sat'
c.roll(Calendar.MONTH, false) //rolls -1
assert String.format('%tF %<ta', c) ==
'1999-09-30 Thu'

```

We can turn off the lenient mode for field updates to force us to give calendars precisely correct values:

```

System.setProperty('user.timezone', 'GMT')
def c= new GregorianCalendar( 2002,
Calendar.JUNE, 30 )
assert c.lenient
c.set( Calendar.DATE, 31 ); c.time
assert String.format('%tF %<ta', c) ==
'2002-07-01 Mon'

c= new GregorianCalendar( 2002,
Calendar.JUNE, 30 )
c.lenient= false
c.set( Calendar.DATE, 31 )
try{ c.time; assert 0 }catch(e){ assert e in
IllegalArgumentException }

```

#### Durations

We can use durations:

```

import groovy.time.*

class Extras{
    static toString(BaseDuration it){
        def list= []
        if(it.years != 0) list<< "$it.years yrs"
        if(it.months != 0) list<< "$it.months
mths"
        if(it.days != 0) list<< "$it.days days"
        if(it.hours != 0) list<< "$it.hours hrs"
        if(it.minutes != 0) list<< "$it.minutes

```

```

mins"
    if(it.seconds != 0 || it.millis != 0)
list<< "$it.seconds.$it.millis secs"
    list.join(', ')
}
}

//enable utility methods for duration
classes using 'category' syntax,
//introduced in a later tutorial...
use(Extras){
    [ {new TimeDuration( 12, 30, 0, 0 )}:
'12 hrs, 30 mins',
      {new TimeDuration( 4, 12, 30, 0, 0 )}: '4
days, 12 hrs, 30 mins',
      {new Duration( 4, 12, 30, 0, 500 )}: '4
days, 12 hrs, 30 mins, 0.500 secs',
      {new DatumDependentDuration( 7, 6, 0,
12, 30, 0, 0 )}:
'7
yrs, 6 mths, 12 hrs, 30 mins',
    ].each{
        assert it.key().toString() == it.value
    }
}

def convertToMilliseconds= { yr, mth, day,
hr, min, sec, mil->
    mil + 1000g*( sec + 60g*( min + 60g*( hr +
24g*(
    day + 30g*( mth + 12g*yr )

```

```
    )))  
}  
  
assert new TimeDuration( 12, 30, 0, 0  
) .toMilliseconds() ==  
    convertToMilliseconds( 0, 0, 0, 12, 30,  
0, 0 )  
    //ignores 61-second leap minutes  
assert new Duration( 114, 12, 30, 0, 0  
) .toMilliseconds() ==  
    convertToMilliseconds( 0, 0, 114, 12,  
30, 0, 0 )  
    //ignores 25-hour daylight-saving days  
assert new DatumDependentDuration( 5, 1, 14,  
12, 30, 0, 0 ) .toMilliseconds() !=
```

```
    convertToMilliseconds( 5, 1, 14, 12, 30,  
0, 0 )  
    //considers 31-day months and leap-years
```

These durations can be created more easily within the TimeCategory:

```
import groovy.time.*  
  
//reuse Extras category from a previous  
example...  
use( [Extras, groovy.time.TimeCategory] ){  
    assert 10.years.class ==  
DatumDependentDuration  
  
    assert 10.years.toString() ==  
        new DatumDependentDuration( 10, 0, 0,  
0, 0, 0, 0 ).toString()  
    assert 4.months.toString() ==  
        new DatumDependentDuration( 0, 4, 0,  
0, 0, 0, 0 ).toString()  
    assert 7.weeks.toString() == new Duration(  
49, 0, 0, 0, 0 ).toString()  
    assert 5.days.toString() == new Duration(  
5, 0, 0, 0, 0 ).toString()  
    assert 12.hours.toString() == new  
TimeDuration( 12, 0, 0, 0 ).toString()  
    assert 15.minutes.toString() == new  
TimeDuration( 0, 15, 0, 0).toString()  
    assert 13.seconds.toString() == new  
TimeDuration( 0, 0, 13, 0 ).toString()  
    assert 750.milliseconds.toString() ==
```

```
    new TimeDuration( 0, 0, 0, 750
).toString()
```

```
    assert 1.day.toString() == new Duration(
1, 0, 0, 0, 0 ).toString()
```

```
        //we can use the singular name for any
of these...
```

```
    assert 25.minute.toString() == new
TimeDuration( 0, 25, 0, 0 ).toString()
```

```
        //...even when not grammatical in
English
    }
```

We can add and subtract durations of different types together:

```
import groovy.time.*

//reuse Extras category from a previous
example...
use( [Extras, groovy.time.TimeCategory] ){

    assert (10.years + 4.months).class ==
DatumDependentDuration
    assert (10.years + 4.months).toString() ==
        new DatumDependentDuration( 10, 4, 0,
0, 0, 0, 0 ).toString()
    assert (10.years.plus(4.months)
).toString() ==
        (10.years + 4.months).toString()
//alternative method name
    assert (4.months + 10.years).toString() ==
(10.years + 4.months).toString()
        //all duration operations are
commutative

    assert (10.years + 4.weeks).class ==
DatumDependentDuration
    assert (5.days + 7.weeks).class ==
Duration
    assert (5.days + 17.hours).class ==
```

```
TimeDuration
```

```
    assert (10.minutes + 5.seconds).class ==  
TimeDuration
```

```
    //adding a DatumDependentDuration and a  
TimeDuration gives a
```

```
    //specially-defined  
TimeDatumDependentDuration...
```

```
    assert (10.years + 12.hours).toString() ==  
        new TimeDatumDependentDuration( 10, 0,  
0, 12, 0, 0, 0 ).toString()
```

```
    assert (10.years + 12.hours).class ==  
TimeDatumDependentDuration
```

```
    assert ( 10.years + new  
TimeDatumDependentDuration( 0, 0, 0, 12, 0,  
0, 0 )
```

```
        ).class ==  
TimeDatumDependentDuration
```

```
    assert ( 10.days + new  
TimeDatumDependentDuration( 0, 0, 0, 12, 0,  
0, 0 )
```

```
        ).class ==  
TimeDatumDependentDuration
```

```
    assert ( 10.minutes + new  
TimeDatumDependentDuration( 0, 0, 0, 12, 0,  
0, 0 )
```

```
        ).class ==  
TimeDatumDependentDuration
```

```
    assert ( new TimeDatumDependentDuration(  
0, 0, 0, 12, 0, 0, 0 ) +
```

```
        new TimeDatumDependentDuration(  
0, 0, 0, 0, 10, 0, 0 )  
        ).class ==  
TimeDatumDependentDuration  
  
    //subtracting durations...  
    assert (10.years - 4.months).class ==  
DatumDependentDuration  
    assert (10.years - 4.months).toString() ==  
        new DatumDependentDuration( 10, -4, 0,  
0, 0, 0, 0 ).toString()  
    assert (10.years.minus(4.months)  
) .toString() ==  
        (10.years - 4.months).toString()  
    //alternative method name  
  
    assert (10.years - 12.hours).class ==  
DatumDependentDuration  
    assert (5.days - 7.weeks).class ==  
Duration  
    assert (5.days - 17.hours).class ==  
TimeDuration  
    assert (10.minutes - 5.seconds).class ==  
TimeDuration
```

```
    assert (10.years - 4.weeks).class ==  
DatumDependentDuration  
}
```

We can add a Date to a duration to give another Date. A TimeDuration takes leap minutes into account, a Duration also takes daylight saving into account, and a DatumDependentDuration considers 31-day months and leap-years:

```
import groovy.time.*  
  
//reuse Extras category from a previous  
example...  
use( [Extras, groovy.time.TimeCategory] ){  
  
    def today= new Date(),  
        tomorrow= today + 1,  
        dayAfter= today + 2,  
        nextWeek= today + 7 //days-only Date  
arithmetic  
    assert ( today + 7.days ).toString() ==  
nextWeek.toString()  
        //use Date and duration together  
    assert ( today.plus(7.days) ).toString()  
== ( today + 7.days ).toString()  
        //alternative method name  
    assert ( 7.days + today ).toString() ==  
nextWeek.toString()  
        //commutative  
    assert ( nextWeek - 6.days ).toString() ==  
tomorrow.toString()  
    assert ( nextWeek.minus(6.days)  
) .toString() == tomorrow.toString()
```

```
        //alternative method name
    assert ( nextWeek - dayAfter ).toString()
== 5.days.toString()
        //subtract two dates to get a duration

    //some handy operations...
    [2.days.ago, 3.days.from.now,
3.days.from.today].each{
```

```
        assert it.class == java.sql.Date
    }
}
```

#### Time Zones

We can retrieve lists of all time zones on a system:

```

//we can get all available time zone ID's,
and get the time zone for an ID...
TimeZone.availableIDs.toList().groupBy{
TimeZone.getTimeZone(it).rawOffset }.
  entrySet().sort{it.key}.reverse().each{
    println String.format('%6.2f hrs: %2d',
                          it.key /
(60*60*1000), it.value.size())
    it.value.each{
      def tz= TimeZone.getTimeZone(it)
      println "${' '*8}$tz.displayName
($tz.ID): " +
              "${tz.DSTSavings /
(60*60*1000)}, ${tz.useDaylightTime()}"
    }
  }

//we can get all the available time zone
ID's for a specific offset...
TimeZone.getAvailableIDs( 12 * (60*60*1000)
).toList().each{
  def tz= TimeZone.getTimeZone(it)
  println "$tz.displayName ($tz.ID): " +
          "${tz.DSTSavings / (60*60*1000)},
${tz.useDaylightTime()}"
}

```

We can access various time zones individually:

```

def tz= TimeZone.'default' //look at the

```

```

default time zone
println "$tz.displayName ($tz.ID): offset
$tz.rawOffset, " +
    "dstSaving $tz.DSTSavings, useDST
${tz.useDaylightTime()}"

TimeZone.'default'=
TimeZone.getTimeZone('GMT') //set the
default time zone

//get a specific time zone from the
system...
tz =
TimeZone.getTimeZone('America/Los_Angeles')
assert tz.displayName == 'Pacific Standard
Time' &&
    tz.rawOffset == -8 * (60*60*1000) &&
    tz.useDaylightTime() &&
    tz.DSTSavings == 1 * (60*60*1000)

//we can fetch a custom time zone, without
any daylight saving, by
//supplying a string...
[ 'GMT-8': 'GMT-08:00',
  'GMT+11': 'GMT+11:00', //hours must be
less than 24
  'GMT+0300': 'GMT+03:00',
  'GMT-3:15': 'GMT-03:15',
  'moo': 'GMT', //syntax errors give GMT
].each{ assert TimeZone.getTimeZone( it.key
).ID == it.value }

```

We can create a time zone with custom daylight-saving time rules:

```
//in the constructor, we can encode the
rules for starting or ending
//Daylight Saving time...
def stz= new SimpleTimeZone(-8*(60*60*1000),
//base GMT offset: -8:00
    "America/Death_Valley",
    Calendar.MARCH, 1, 0,      //DST starts on 1
March exactly
    2*(60*60*1000),
SimpleTimeZone.STANDARD_TIME,
                                //...at 2:00am in
standard time (wall time)
    Calendar.OCTOBER, 1, -Calendar.SUNDAY,
                                //ends first Sun
on/after 1 Oct (first Sun in Oct)...
    2*(60*60*1000), SimpleTimeZone.WALL_TIME,
                                //...at 2:00am in
daylight time (wall time)
    1*(60*60*1000) ) // save 1 hour

//leave out last parameter which defaults to
'save 1 hour', ie, 1*(60*60*1000)
stz= new SimpleTimeZone(15*(60*60*1000),
//base GMT offset: +15:00
    "Pacific/Happy_Isle",
    Calendar.AUGUST, -21, -Calendar.FRIDAY,
                                //starts on last
Friday on or before 21 August...
```

```

    2*(60*60*1000),          //...at 2:00am in
standard time (wall time, the default)
    Calendar.APRIL,          1, -Calendar.SUNDAY,
                                //ends first Sun
on/after 1 Apr (first Sun in Apr)...
    2*(60*60*1000) )        //...at 2:00am in
daylight time (wall time, the default)

//two extra optional parameters (if present,
both must be)...
stz= new SimpleTimeZone( 1*(60*60*1000),
//base GMT offset: +1:00
    "Europe/Alps",
    Calendar.JUNE,          8, -Calendar.MONDAY,
                                //starts first Mon
on/after 8 Jun (second Mon in Jun)...
    1*(60*60*1000), SimpleTimeZone.UTC_TIME,
//...at 1:00am in UTC time
    Calendar.OCTOBER, -1, Calendar.SUNDAY,
                                //ends on the last
Sunday in October...
    1*(60*60*1000), SimpleTimeZone.UTC_TIME,
//...at 1:00am in UTC time
    1*(60*60*1000) ) // save 1 hour

//we can instead encode the rules in the
same way using methods...
stz= new SimpleTimeZone( -8*(60*60*1000),
//base GMT offset: -8:00
    "America/Death_Valley" ) //no
daylight-saving schedule in constructor

```

```
stz.setStartRule(Calendar.APRIL, 1,
-Calendar.SUNDAY, 2 * 60*60*1000)

//first Sun in Apr
stz.setEndRule(Calendar.OCTOBER, -1,
Calendar.SUNDAY, 2 * 60*60*1000)

//last Sun in Oct

assert stz.dSTSavings == 60*60*1000 //the
default
stz.dSTSavings= 2 * 60*60*1000
assert stz.dSTSavings == 2 * 60*60*1000
assert stz.getDSTSavings() == 2 * 60*60*1000
    //unusually-cased property name
'dSTSavings' has equivalent method names
    //'getDSTSavings()' and
'setDSTSavings()'

stz.setStartRule(Calendar.MAY, 1, 2 *
60*60*1000)
    //shortcut method for fixed date in
month
stz.setStartRule(Calendar.MAY, 10,
Calendar.SUNDAY, 2 * 60*60*1000, true)
    //shortcut for first Sunday on or after
10 May; true means 'after'
stz.setEndRule(Calendar.OCTOBER, 20,
```

```
Calendar.SATURDAY, 2 * 60*60*1000, false)
//shortcut for first Saturday on or
before 20 October; false means 'before'
```

(Coordinated universal time, UTC, being based on an atomic clock, enables an extra second, a "leap second", to be added as the last second of the day on December 31 or June 30.)

We can use time zones in many various ways:

```
System.setProperty('user.timezone', 'GMT')
//we can set the default time zone

def tz= new SimpleTimeZone( -8*(60*60*1000),
'Somewhere',
Calendar.MARCH, 1, 0, 2*(60*60*1000),
Calendar.OCTOBER, 31, 0, 2*(60*60*1000) )
def cal= new GregorianCalendar( tz )
//create a calendar with today's date in
a specified time zone
cal= Calendar.getInstance( tz ) //another
way

cal= new GregorianCalendar(2009,
Calendar.JULY, 22)
//we can create a calendar with the
default time zone...
cal.setTimeZone= tz //...then set the time zone
assert cal.getTimeZone == tz
assert cal.get(Calendar.ZONE_OFFSET) ==
-8*(60*60*1000)
assert cal.get(Calendar.DST_OFFSET) ==
(60*60*1000)
```

```

assert Calendar.FIELD_COUNT == 17
    //the number of fields such as
DAY_OF_YEAR and ZONE_OFFSET in Calendar

//we can test whether two time zones have
the same rules...
assert tz.hasSameRules(
    new SimpleTimeZone( -8*(60*60*1000),
'Somewhere Else',
    Calendar.MARCH, 1, 0, 2*(60*60*1000),
    Calendar.OCTOBER, 31, 0, 2*(60*60*1000)
) )
assert ! tz.hasSameRules(
    new SimpleTimeZone( -8*(60*60*1000),
'Somewhere Else',
    Calendar.APRIL, 1, 0, 2*(60*60*1000),
    Calendar.OCTOBER, 31, 0, 2*(60*60*1000)
) )

//some methods available within
TimeCategory...
use(groovy.time.TimeCategory){
    cal= new GregorianCalendar( tz )
    def today= cal.time
    println today.timeZone
    println today.daylightSavingsOffset
//returns a duration
    def nextWeek= today + 7
    println( (nextWeek -
today).daylightSavingsOffset )
    //a duration also has a daylight

```

```

savings time offset
    println(
nextWeek.getRelativeDaylightSavingsOffset(
today ) )
}

//we can test if a certain date is in
daylight saving time for a time zone...
assert tz.inDaylightTime( new
GregorianCalendar(1990, Calendar.MAY,
5).time )
assert ! tz.inDaylightTime(
    new GregorianCalendar(1990,
Calendar.NOVEMBER, 5).time )

//we can set the first year daylight savings
time operates...
tz.startYear= 1973
assert ! tz.inDaylightTime( new
GregorianCalendar(1971, Calendar.MAY,
5).time )

//some extra format codes for dates...
println String.format('%tZ', cal)
    //to see a string representing the time
zone, eg, GMT-07:00
println String.format('%tz', cal) //numeric
offset from GMT, eg, -0800
assert String.format('%tc', cal) ==
    String.format('%ta %<tb %<td %<tT %<tZ
%<tY', cal)

```

```
//we can view the Gregorian changeover
date...
assert String.format( '%ta %<td %<tb %<tY',
cal.gregorianChange ) ==
    'Fri 15 Oct 1582' //default for GMT time
zone
cal= new GregorianCalendar()
cal.set(1582, Calendar.OCTOBER, 15)
cal.time
assert String.format( '%ta %<td %<tb %<tY',
cal.time - 1 ) ==
    'Thu 04 Oct 1582' //the day before the
big change

//check for leap years (this instance method
acts like a static method)...
[1999, 1998, 1997, 1900, 1800, 1700].each{
assert ! cal.isLeapYear(it) }
[2000, 1996, 1992, 1600, 1500, 1400].each{
```

```
assert cal.isLeapYear(it) }  
    //1500 and before use Julian calendar  
rules
```

## JN1015-Collections

### Lists

A list is an ordered collection of objects:

```
def list = [5, 6, 7, 8]  
assert list.size == 4  
assert list.size() == 4  
assert list.class == ArrayList //the  
specific kind of list being used  
  
assert list[2] == 7 //indexing starts at 0  
assert list.getAt(2) == 7 //equivalent  
method to []  
assert list.get(2) == 7 //alternative method  
  
list[2] = 9  
assert list == [5, 6, 9, 8, ] //trailing  
comma OK  
  
list.putAt(2, 10) //equivalent method to []  
when value being changed  
assert list == [5, 6, 10, 8]  
assert list.set(2, 11) == 10 //alternative  
method that returns old value  
assert list == [5, 6, 11, 8]
```

```
assert [ 'a', 1, 'a', 'a', 2.5, 2.5f, 2.5d,  
'hello', 7g, null, 9 as byte ]
```

```
        //objects can be of  
different types; duplicates allowed
```

```
assert [1,2,3,4,5][-1] == 5 //use negative  
indices to count from the end
```

```
assert [1,2,3,4,5][-2] == 4
```

```
assert [1,2,3,4,5].getAt(-2) == 4 //getAt()  
available with negative index...
```

```
try{ [1,2,3,4,5].get(-2); assert 0 }
```

```
//...but not get()
catch(e){ assert e instanceof
ArrayIndexOutOfBoundsException }
```

Lists can be evaluated as a boolean value:

```
assert ! [] //an empty list evaluates as
false
assert [1] && ['a'] && [0] && [0.0] &&
[false] && [null]
    //all other lists, irrespective of
contents, evaluate as true
```

We can use [] to assign a new empty list and << to append items to it:

```
def list = []; assert list.size() == 0
list << 5; assert list.size() == 1
list << 7 << 'i' << 11; assert list == [5,
7, 'i', 11]
list << ['m', 'o']; assert list == [5, 7,
'i', 11, ['m', 'o'] ]
assert ( [1,2] << 3 << [4,5] << 6 ) ==
[1,2,3, [4, 5], 6]
    //first item in chain of << is target
list
assert ([1,2,3] << 4) ==
([1,2,3].leftShift(4))
    //using this method is equivalent to
using <<
```

We can add to a list in many ways:



```

assert [1,2] + 3 + [4,5] + 6 == [1, 2, 3, 4,
5, 6]
assert [1,2].plus(3).plus([4,5]).plus(6) ==
[1, 2, 3, 4, 5, 6]
    //equivalent method for +
def a= [1,2,3]; a += 4; a += [5,6]; assert a
== [1,2,3,4,5,6]
assert [1, *[222, 333], 456] == [1, 222,
333, 456]
assert [ *[1,2,3] ] == [1,2,3]
assert [ 1, [2,3,[4,5],6], 7, [8,9]
].flatten() == [1, 2, 3, 4, 5, 6, 7, 8, 9]

def list= [1,2]
list.add(3) //alternative method name
list.addAll([5,4]) //alternative method name
assert list == [1,2,3,5,4]

list= [1,2]
list.add(1,3) //add 3 just before index 1
assert list == [1,3,2]
list.addAll(2,[5,4]) //add [5,4] just before
index 2
assert list == [1,3,5,4,2]

list = ['a', 'b', 'z', 'e', 'u', 'v', 'g']
list[8] = 'x'
assert list == ['a', 'b', 'z', 'e', 'u',
'v', 'g', null, 'x']
    //nulls inserted if required

```

We can use the each and eachWithIndex methods to execute code on each item in a list:

```
[1, 2, 3].each{ println "Item: $it" }  
['a', 'b', 'c'].eachWithIndex{ it, i ->  
println "$i: $it" }
```

We can construct a list using another's elements as a template:

```
def list1= ['a','b','c']  
def list2 = new ArrayList( list1 )  
    //construct a new list, seeded with the  
same items as in list1  
assert list2 == list1 // == checks that each  
corresponding element is the same  
def list3 = list1.clone()  
assert list3 == list1
```

We can perform a closure on each item of a list and return the result:

```

assert [1, 2, 3].collect{ it * 2 } == [2, 4,
6]
    //simple call gives single result
assert [1, 2, 3]*.multiply(2) == [1, 2,
3].collect{ it.multiply(2) }
    //shortcut syntax instead of collect

def list= []
assert [1, 2, 3].collect( list ){ it * 2 }
== [2, 4, 6]
    //this style of call gives two identical
results
assert list == [2, 4, 6]

```

Other methods on a list return a value:

```

assert [1, 2, 3].find{ it > 1 } == 2
assert [1, 2, 3].findAll{ it > 1 } == [2, 3]
assert ['a','b','c','d','e'].findIndexOf{ it
in ['c','e','g'] } == 2
    //find first item that satisfies closure
assert [1, 2, 3].every{ it < 5 }
assert ! [1, 2, 3].every{ it < 3 }
assert [1, 2, 3].any{ it > 2 }
assert ! [1, 2, 3].any{ it > 3 }

    // sum anything with a plus() method
assert [1,2,3,4,5,6].sum() == 21
assert ['a','b','c','d','e'].sum{
it=='a'?1: it=='b'?2: it=='c'?3:

```

```
it=='d'?4: it=='e'?5: 0
} == 15
assert ['a','b','c','d','e'].sum{ (char)it -
(char)'a' } == 10
assert ['a','b','c','d','e'].sum() ==
'abcde'
assert [['a','b'], ['c','d']].sum() ==
['a','b','c','d']
    // an initial value can be provided
assert [].sum(1000) == 1000
assert [1, 2, 3].sum(1000) == 1006

assert [1, 2, 3].join('-') == '1-2-3'
assert [1, 2, 3].inject('counting: '){ str,
item -> str + item } ==
```

```
'counting: 123'  
assert [1, 2, 3].inject(0){ count, item ->  
count + item } == 6
```

We can find the maximum and minimum in a collection:

```
def list= [9, 4, 2, 10, 5]  
assert list.max() == 10  
assert list.min() == 2  
assert Collections.max( list ) == 10  
assert Collections.min( list ) == 2  
assert ['x', 'y', 'a', 'z'].min() == 'a'  
    //we can also compare single characters  
  
def list2= ['abc', 'z', 'xyzuvw', 'Hello',  
'321']  
assert list2.max{ it.size() } == 'xyzuvw'  
    //we can use a closure to spec the  
    sorting behaviour  
assert list2.min{ it.size() } == 'z'
```

We can use a "Comparator" to define the comparing behaviour:

```

def mc= [compare:{a,b-> a.equals(b)? 0: a<b?
-1: 1}] as Comparator
    //this syntax to be explained in a later
tutorial
def list= [7,4,9,-6,-1,11,2,3,-9,5,-13]
assert list.max( mc ) == 11
assert list.min( mc ) == -13
assert Collections.max( list, mc ) == 11
assert Collections.min( list, mc ) == -13

def mc2= [
    compare: {a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 }
] as Comparator
    //we should always ensure a.equals(b)
returns 0, whatever else we do,
    //to avoid inconsistent behaviour in
many contexts

assert list.max( mc2 ) == -13
assert list.min( mc2 ) == -1
assert list.max{a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 } == -13
assert list.min{a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 } == -1

```

We can remove elements from a list by referring to the element/s to be removed:

```

assert ['a','b','c','b','b'] - 'c' ==
['a','b','b','b']
    //remove 'c', and return resulting list
assert ['a','b','c','b','b'] - 'b' ==
['a','c']
    //remove all 'b', and return resulting
list
assert ['a','b','c','b','b'] - ['b','c'] ==
['a']
    //remove all 'b' and 'c', and return
resulting list
assert ['a','b','c','b','b'].minus('b') ==
['a','c']
    //equivalent method name for -
assert ['a','b','c','b','b'].minus(
['b','c'] ) == ['a']
def list= [1,2,3,4,3,2,1]
list -= 3
assert list == [1,2,4,2,1] //use -= to
remove 3, permanently
assert ( list -= [2,4] ) == [1,1] //remove
2's and 4's, permanently

```

We can remove an element by referring to its index:

```

def list= [1,2,3,4,5,6,2,2,1]
assert list.remove(2) == 3 //remove the
third element, and return it
assert list == [1,2,4,5,6,2,2,1]

```

We can remove the first occurrence of an element from a list:

```
def list= ['a','b','c','b','b']
assert list.remove('c') //remove 'c', and
return true because element removed
assert list.remove('b')
    //remove first 'b', and return true
because element removed
assert ! list.remove('z') //return false
because no elements removed
assert list == ['a','b','b']
```

We can clear a list of all elements:

```
def list= ['a',2,'c',4]
list.clear()
assert list == []
```

We can pop the last item from a list, and use the list as a simple stack:

```
def stack= [1,2,4,6]
stack << 7
assert stack == [1,2,4,6,7]
assert stack.pop() == 7
assert stack == [1,2,4,6]
```

Other useful operators and methods:

```
assert 'a' in ['a','b','c']
assert ['a','b','c'].contains('a')
assert [1,3,4].containsAll([1,4])

assert [].isEmpty()
assert [1,2,3,3,3,3,4,5].count(3) == 4

assert
[1,2,4,6,8,10,12].intersect([1,3,6,9,12]) ==
[1,6,12]

assert [1,2,3].disjoint( [4,6,9] )
assert ! [1,2,3].disjoint( [2,4,6] )
assert Collections.disjoint( [1,2,3],
[4,6,9] ) //alternative method name
```

There's various ways of sorting:

```

assert [6,3,9,2,7,1,5].sort() ==
[1,2,3,5,6,7,9]

def list= ['abc', 'z', 'xyzuvw', 'Hello',
'321']
assert list.sort{ it.size() } == ['z',
'abc', '321', 'Hello', 'xyzuvw']

def list2= [7,4,-6,-1,11,2,3,-9,5,-13]
assert list2.sort{a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 } ==
[-1, 2, 3, 4, 5, -6, 7, -9, 11, -13]
def mc= [
  compare: {a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 }
] as Comparator
assert list2.sort(mc) == [-1, 2, 3, 4, 5,
-6, 7, -9, 11, -13]

def list3= [6,-3,9,2,-7,1,5]
Collections.sort(list3)
assert list3 == [-7,-3,1,2,5,6,9]
Collections.sort(list3, mc)
assert list3 == [1,2,-3,5,6,-7,9]

```

We can repeat a list or element:

```

assert [1,2,3] * 3 == [1,2,3,1,2,3,1,2,3]
assert [1,2,3].multiply(2) == [1,2,3,1,2,3]
assert Collections.nCopies( 3, 'b' ) ==
['b', 'b', 'b']
    //nCopies works differently
assert Collections.nCopies( 2, [1,2] ) == [
[1,2], [1,2] ] //not [1,2,1,2]

```

We can find the first or last index of items in a list:

```

assert
['a','b','c','d','c','d'].indexOf('c') == 2
//index returned
assert
['a','b','c','d','c','d'].indexOf('z') == -1
    //index -1 means value not in list
assert
['a','b','c','d','c','d'].lastIndexOf('c')
== 4

```

Some very common methods are:

```

def list= [], list2= []
[1,2,3,4,5].each{ list << it*2 }
assert list == [2,4,6,8,10]
[1,2,3,4,5].eachWithIndex{item, index->
list2 << item * index }
    //closure supplied must have 2 params
assert list2 == [0,2,6,12,20]

```

A list may contain itself, but equals() may not always be consistent. Consider this:

```
def list, list2, list3
list= [1, 2, list, 4]
list2= [1, 2, list2, 4]
assert list.equals(list2)
list3= [1, 2, list, 4]
assert ! list.equals(list3)
```

#### Ranges and List-Slicing

Ranges are consecutive lists of sequential values like Integers, and can be used just like a List:

```
assert 5..8 == [5,6,7,8] //includes both
values
assert 5..<8 == [5, 6, 7] //excludes
specified top value
```

They can also be used with single-character strings:

```
assert ('a'..'d') == ['a', 'b', 'c', 'd']
```

Ranges are handy with the each method:

```
def n=0
(1..10).each{ n += it }
assert n == 55
```

We can define lists using a range or ranges within a list. This is called slicing:

```
assert [*3..5] == [3,4,5]
assert [ 1, *3..5, 7, *9..<12 ] ==
[1,3,4,5,7,9,10,11]
```

Lists can be used as subscripts to other lists:

```
assert ('a'..'g')[ 3..5 ] == ['d','e','f']
assert ('a'..'g').getAt( 3..5 ) ==
['d','e','f'] //equivalent method name

assert ('a'..'g')[ 1, 3, 5, 6 ] ==
['b','d','f','g']
assert ('a'..'g')[ 1, *3..5 ] ==
['b','d','e','f']
assert ('a'..'g')[ 1, 3..5 ] ==
['b','d','e','f']
    //range in subscript flattened
automatically
assert ('a'..'g')[-5..-2] ==
['c','d','e','f']
assert ('a'..'g').getAt( [ 1, *3..5 ] ) ==
['b','d','e','f']
    //equivalent method name
assert ('a'..'g').getAt( [ 1, 3..5 ] ) ==
['b','d','e','f']
```

We can view a sublist of a list:

```
def list=[1,2,3,4,5], sl= list.subList(2,4)
sl[0]= 9 //if we change the sublist...
assert list == [1,2,9,4,5] //...backing list
changes...
list[3]= 11
assert sl == [9,11] //...and vice versa
```

We can perform the same methods on the subscripted lists as we can on the lists they're produced from:

```
assert
['a', 'b', 'c', 'd', 'e'][1..3].indexOf('c') ==
1
    //note: index of sublist, not of list
```

We can update items using subscripting too:

```
def list = ['a','b','c','d','e','f','g']
list[2..3] = 'z'
assert list == ['a', 'b', 'z', 'e', 'f', 'g'] //swap two entries for one
list[4..4]= ['u','v']
assert list == ['a', 'b', 'z', 'e', 'u', 'v', 'g'] //swap one entry for two

def list= ['a', 'b', 'z', 'e', 'u', 'v', 'g']
list[0..1]= []
assert list == ['z', 'e', 'u', 'v', 'g']
//remove entries from index range
list[1..1]= []
assert list == ['z', 'u', 'v', 'g'] //remove
entry at index
```

We can also use a method instead of [] with ranges:

```

def list = ['a','b','c','d','e','f','g']
list.putAt(2..3, 'z')
assert list == ['a', 'b', 'z', 'e', 'f', 'g']
list.putAt(4..4, ['u','v'])
assert list == ['a', 'b', 'z', 'e', 'u', 'v', 'g']
list.putAt(1..<3, [])
assert list == ['a', 'e', 'u', 'v', 'g']
list.putAt( 0..<0, 'm' ) //
assert list == ['m', 'a', 'e', 'u', 'v', 'g']
list.removeRange(1,3) //another method to do
similar, means: list[1..<3]= []
list[1..2].clear()
assert list == ['m', 'g']

```

More List Utilities

To reverse a list:

```

assert [1,2,3].reverse() == [3,2,1]

def list= ['a','b','c','d','e']
Collections.reverse( list )
assert list == ['e','d','c','b','a']
use(Collections){ list.reverse() }
//alternative syntax for null-returning
Collections.reverse(List)
assert list == ['a','b','c','d','e']

```

```
def list2= []
[1,2,3,4,5].reverseEach{ list2 << it*2 }
  //same as, but more efficient than:
[...].reverse().each{...}
assert list2 == [10,8,6,4,2]
```

```
assert [1,2,3,4,5,6][3..1] == [4,3,2]
  //use backwards range to reverse
returned sublist
```

```
def list3 = [1, 2, -3, 5, 6, -7, 9]
def rmc= Collections.reverseOrder()
Collections.sort(list3, rmc)
assert list3 == [9, 6, 5, 2, 1, -3, -7]
```

```
def list4 = [1, 2, -3, 5, 6, -7, 9]
def mc= [
  compare: {a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1}
] as Comparator
```

```
def rmc2= Collections.reverseOrder( mc )
Collections.sort(list4, rmc2)
assert list4 == [9, -7, 6, 5, -3, 2, 1]
```

We can perform a binary search on a sorted list:

```
assert
Collections.binarySearch([2,5,6,7,9,11,13,26
,31,33], 26) == 7
    //list must already be sorted
assert
Collections.binarySearch([2,5,6,7,9,11,13,31
,33], 26) == -8
    //if key not there, give negative of one
plus the index before which key
    //would be if it was there

def mc= [
    compare: {a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 }
] as Comparator
assert
Collections.binarySearch([2,-5,-6,7,9,-11,13
,26,31,-33], 26, mc) == 7
    //give comparator list sorted by
```

We can remove or retain elements in bulk. retainAll() gives the intersection of two lists; removeAll() gives the asymmetric difference.

```

def list= ['a','b','c','b','b','e','e']
assert list.removeAll( ['b','z'] )
    //remove 'b' and 'z', return true
because list changed
assert list == ['a','c','e','e']
assert ! list.removeAll( ['b','z'] )
    //return false because list didn't
change
assert list == ['a','c','e','e']
assert list.retainAll( ['a','e'] )
    //retain only 'a' and 'e', return true
because list changed
assert list == ['a','e','e']
assert ! list.retainAll( ['a','e'] )
    //retain only 'a' and 'e', return true
because list didn't change
assert list == ['a','e','e']

```

Some miscellaneous methods:

```

def list= ['a', 7, 'b', 9, 7, 7, 2.4, 7]
Collections.replaceAll( list, 7, 55)
assert list == ['a', 55, 'b', 9, 55, 55,
2.4, 55]

list= ['a', 7, 'b', 9, 7, 7, 2.4, 7]
use(Collections){ list.replaceAll(7, 55) }
//alternative syntax
assert list == ['a', 55, 'b', 9, 55, 55,
2.4, 55]

```

```
list= ['a',2,null,4,'zyx',2.5]
use(Collections){ list.fill( 'g' ) } //or:
Collections.fill( list, 'g' )
assert list == ['g', 'g', 'g', 'g', 'g',
'g']
```

```
list= ['a', 'e', 'i', 'o', 'u', 'z']
use(Collections){ list.swap(2, 4) } //or:
Collections.swap(list, 2, 4)
assert list == ['a', 'e', 'u', 'o', 'i',
'z']
```

```
assert
Collections.frequency(['a','b','a','c','a','a',
a,'d','e'], 'a') == 4
use(Collections){
    assert
    ['a','b','a','c','a','a','d','e'].frequency(
'a') == 4
}
```

```
list= ['a','b','c','d','e']
Collections.rotate(list, 3)
assert list == ['c','d','e','a','b']
use(Collections){ list.rotate(-2) }
assert list == ['e','a','b','c','d']
```

```
list= [1,2,3,4,5]
Collections.shuffle(list, new Random())
    //we can supply our own random number
```

```

generator...
assert list != [1,2,3,4,5]

list= [1,2,3,4,5]
Collections.shuffle(list) //...or use the
default one
assert list != [1,2,3,4,5]

assert [3,5,5,5,2].unique() == [3,5,2]
def mc= [ compare:
  {a,b-> a.equals(b) || a.equals(-b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 }
] as Comparator

assert [3,5,5,-5,2,-7].unique(mc) ==
[3,5,2,-7]
  //remove subsequent items comparator
considers equal
assert [3,5,5,-5,2,-7].unique{a, b->
  a == b || a == -b? 0:
Math.abs(a)<Math.abs(b)? -1: 1
} == [3,5,2,-7]

list= [1,2,3]
Collections.copy( list, [9,8,7] )
assert list == [9,8,7] //overwrites original
data
Collections.copy( list, [11,12] ) //source
list shorter...
assert list == [11,12,7] //...which leaves
remaining entries unchanged

```

```
try{ Collections.copy( list, [21,22,23,24]
); assert 0 } //source list too long
    catch(e){ assert e instanceof
IndexOutOfBoundsException }
```

```
list= [1,8,8,2,3,7,6,4,6,6,2,3,7,5]
assert Collections.indexOfSubList( list,
[2,3,7] ) == 3
assert Collections.lastIndexOfSubList( list,
[2,3,7] ) == 10
```

```
assert Collections.indexOfSubList( list,  
[9,9,13] ) == -1  
    //if sublist doesn't exist
```

#### Sets

A set is an unordered collection of objects, with no duplicates. It can be considered as a list with restrictions, and is often constructed from a list:

```
def s1= [1,2,3,3,3,4] as Set,  
    s2= [4,3,2,1] as Set,  
    s3= new HashSet( [1,4,2,4,3,4] )  
assert s1.class == HashSet && s2.class ==  
HashSet  
    //the specific kind of set being used  
assert s1 == s2  
assert s1 == s3  
assert s2 == s3  
assert s1.asList() && s1.toList()  
    //a choice of two methods to convert a  
set to a list  
assert ( ( [] as Set ) << null << null << null  
) == [null] as Set
```

A set should not contain itself as an element.

Most methods available to lists, besides those that don't make sense for unordered items, are available to sets.

```
[ { it[1] }, { it.getAt(1) }, {
it.putAt(1,4) }, { it.reverse() } ].each{
    try{ it([1,2,3] as Set); assert 0 }
    catch(e){ assert e instanceof
MissingMethodException }
}
```

The add() and addAll() methods return false if the set wasn't changed as a result of the operation:

```
def s= [1,2] as Set
assert s.add(3)
assert ! s.add(2)
assert s.addAll( [5,4] )
assert ! s.addAll( [5,4] )
assert s == [1,2,3,5,4] as Set
```

#### *Some Set Theory Concepts*

Here are a few common set theory concepts and how to make them work in Groovy.

## Union

Two sets, A and B, and a third set, C, where C combines all *distinct* elements of both A and B. Set C would be a *union* of A and B. Use the '+' operator to create a union between two sets.

## Union

```
// Union of two sets
def a = [1,2,3] as Set
def b = [3,4,5] as Set
def union = a + b
assert union == [1,2,3,4,5] as Set

// Union of three sets
def c = [1,2,3,4] as Set
def d = [3,5] as Set
def e = [5,6,7,8] as Set
def tripleUnion = c + d + e
assert tripleUnion == [1,2,3,4,5,6,7,8] as
Set
```

## Symmetric Difference

Two sets, A and B, and a third set, C, where C combines all elements of A and B **except** the items existing in both A and B. Set C would be a *symmetric difference* of A and B. Here is the original example of symmetric difference:

## Symmetric Difference

```
def s1=[1,2,3,4,5,6], s2=[4,5,6,7,8,9]
def diff = (s1 as Set) + s2
tmp = s1 as Set
tmp.retainAll(s2)
diff.removeAll(tmp)
assert diff == [1,2,3,7,8,9] as Set
```

## Intersection

Two sets, A and B, and a third set, C, where C combines only elements of A and B that exist in both A and B. Set C would be an *intersection* of A and B. The *intersect* method is actually a part of the

## Intersection

```
def a = [1,2,3]
def b = [3,4,5]
def intersection = a.intersect(b) // part
of groovy Collections api
assert intersection == [3]
```

## Compliment

Two sets, A and B, and a third set, C, where C contains all elements from A that do not also exist in B. Set C would contain the compliment of A in B. Said differently, set C shows what A has to offer B that B doesn't already have.

## Compliment

```
// using the .minus() method
def a = [1,2,3]
def b = [2,3,4]
def c = a.minus(b) // give me everything in
A that doesn't exist in B
assert c == [1]

// using the '-' operator
def d = [1,2,3]
def e = [2,3,4]
def f = d - e // give me everything in d
that doesn't exist in e
assert f == [1]
```

## Examples with Lists and Sets

For small numbers of items, it's common in Groovy to use a list for set processing, and only convert it to a set when necessary, eg, for output.

Though the uniqueness of set items is useful for some processing, for example, if we want to separate the unique and duplicating items in a list:

```
list=[1,2,7,2,2,4,7,11,5,2,5]
def uniques= [] as Set, dups= [] as Set
list.each{ uniques.add(it) || dups.add(it) }
uniques.removeAll(dups)
assert uniques == [1,4,11] as Set && dups ==
[2,5,7] as Set
```

To calculate the symmetric set difference of two sets non-destructively:

```
def s1=[1,2,3,4,5,6], s2=[4,5,6,7,8,9]
def diff = (s1 as Set) + s2
tmp = s1 as Set
tmp.retainAll(s2)
diff.removeAll(tmp)
assert diff == [1,2,3,7,8,9] as Set
```

## Sorted Sets

A sorted set is one with extra methods that utilize the sorting of the elements. It's often more efficient than doing the same with lists.

```

def list= [3,2,3,3,1,7,5]
assert new TreeSet(list) == new
TreeSet([1,1,1,2,5,7,3,1])
assert new TreeSet(list).toList() ==
list.unique().sort()

assert new TreeSet(list).first() ==
list.unique().min()
assert new TreeSet(list).last() ==
list.unique().max()

```

We can construct a TreeSet by giving a comparator to order the elements in the set:

```

def c= [ compare:
  {a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 }
] as Comparator
def ts= new TreeSet( c )
ts<< 3 << -7 << 9 << -2 << -4
assert ts == new TreeSet( [-2, 3, -4, -7, 9]
)
assert ts.comparator() == c //retrieve the
comparator

```

The range-views, headSet() tailSet() and subSet(), are useful views of the items in a sorted set. These range-views remain valid even if the backing sorted set is modified directly. The sorted set returned by these methods will throw an IllegalArgumentException if the user attempts to insert an element out of the range.

```

def ss= new TreeSet(['a', 'b', 'c', 'd', 'e'])

```

```
def hs= ss.headSet('c')
assert hs == new TreeSet(['a','b'])
  //return all elements < specified
element
hs.remove('a')
assert ss == new TreeSet(['b','c','d','e'])
  //headset is simply a view of the data
in ss

def ts= ss.tailSet('c')
assert ts == new TreeSet(['c','d','e'])
  //return all elements >= specified
element
ts.remove('d')
assert ss == new TreeSet(['b','c','e'])
  //tailset is also a view of data in ss

def bs= ss.subSet('b','e')
assert bs == new TreeSet(['b','c'])
  //return all elements >= but < specified
element
bs.remove('c')
assert ss == new TreeSet(['b','e'])
  //subset is simply a view of the data in
ss

ss << 'a' << 'd'
assert hs == new TreeSet(['a','b'])
  //if backing sorted set changes, so do
```

## **range-views**

```
assert ts == new TreeSet(['d','e'])  
assert bs == new TreeSet(['b','d'])
```

For a SortedSet of strings, we can append '\0' to a string to calculate the next possible string:

```
def dic= new TreeSet(  
    ['aardvark', 'banana', 'egghead',  
    'encephalograph', 'flotsam', 'jamboree']  
)  
assert dic.subSet('banana',  
    'flotsam').size() == 3  
    //incl 'banana' but excl 'flotsam'  
assert dic.subSet('banana',  
    'flotsam\0').size() == 4 //incl both  
assert dic.subSet('banana\0',  
    'flotsam').size() == 2 //excl both  
dic.subSet('e', 'f').clear()  
assert dic == new TreeSet(  
    ['aardvark', 'banana', 'flotsam',  
    'jamboree']  
)//clear all words beginning with 'e'
```

To go one element backwards from an element elt in a SortedSet:

```
Object predecessor = ss.headSet( elt  
).last()
```

## **Immutable Collections**

We can convert a list or set into one that can't be modified:

```
def imList= ['a', 'b', 'c'].asImmutable()  
try{ imList<< 'd'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

```
imList= Collections.unmodifiableList( ['a',  
'b', 'c'] ) //alternative way  
try{ imList<< 'd'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

```
def imSet= (['a', 'b', 'c'] as  
Set).asImmutable()  
try{ imSet<< 'd'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

```
imSet= Collections.unmodifiableSet( ['a',  
'b', 'c'] as Set ) //alternative way  
try{ imSet<< 'd'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

```
def imSortedSet= ( new TreeSet(['a', 'b',  
'c'] ) ).asImmutable()  
try{ imSortedSet<< 'd'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

```
imSortedSet=  
Collections.unmodifiableSortedSet( new
```

```
TreeSet(['a', 'b', 'c']) )
```

```
//alternative way
```

```
try{ imSortedSet<< 'd'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

We can create an empty list or set that can't be modified:

```
def list= Collections.emptyList()  
assert list == []  
try{ list<< 'a'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }  
list= Collections.EMPTY_LIST  
assert list == []  
try{ list<< 'a'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }  
  
def set= Collections.emptySet()  
assert set == [] as Set  
try{ set<< 'a'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }  
set= Collections.EMPTY_SET  
assert set == [] as Set  
try{ set<< 'a'; assert 0 }  
catch(e){ assert e instanceof  
UnsupportedOperationException }
```

We can create a single-element list that can't be modified:

```
def singList= Collections.singletonList('a')
assert singList == ['a']
try{ singList<< 'b'; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
```

```
def singSet = Collections.singleton('a')
assert singSet == ['a'] as Set
try{ singSet<< 'b'; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
```

## JN1025-Arrays

An object array is a fixed-size sequence of objects:

```
def a= new Object[4] //we must specify the
size of the fixed-size array
assert a.size() == 4
assert a.length == 4 //field alternative to
size()
a.each{ assert it == null } //default value
is null
assert a instanceof Object[]
assert a.class == Object[]

a[0]= 'a'
a[1]= 2 //elements can be any value
a.putAt(2, 'c') //alternative method name
syntax
```

```
a[3]= false
assert a[0] == 'a' && a[1] == 2 &&
a.getAt(2) == 'c' && a.getAt(3) == false
    //either subscript or method name
assert a[-4] == 'a' && a[-3] == 2 && a[-2]
== 'c' && a[-1] == false
    //subscripts can be negative

try{ a[4]; assert 0 }
catch(e){ assert e instanceof
ArrayIndexOutOfBoundsException }
try{ a[-5]; assert 0 }
catch(e){ assert e instanceof
ArrayIndexOutOfBoundsException }

assert a[1..2] == [2, 'c'] //we can use the
same subscripting as for lists
assert a[2..2] == ['c']
assert a[0, 2..3] == ['a', 'c', false]

assert a.toList() == ['a', 2, 'c', false]
```

```
assert a as List == ['a', 2, 'c', false]
assert a.toArrayString() == '{"a", 2, "c",
false}'
```

The subscript used in constructing object arrays is evaluated as an integer:

```
assert new Object[ 0x100000003 ].size() == 3
    //index coerced to integer, positive or
negative
try{ new Object[ 0x80000000 ]; assert 0 }
catch(e){ assert e instanceof
NegativeArraySizeException }
```

We can specify the initial collection of contained objects when we construct the array. Those objects can be any other entity in Groovy, eg, numbers, boolean values, characters, strings, regexes, lists, maps, closures, expandos, classes, class instances, or even other object arrays:

```
assert [
    14.25,
    17g,
    [1,2,3],
    'Hello, world',
    ['a', false, null, 5] as Object[],
    new Object[200],
    { it*it },
    ArrayList,
] as Object[]
```

We can make a shallow copy using clone():

```
def aq= [1,2]
assert ([ aq, 3 ] as
Object[]).clone()[0].is( aq )
    //clone() makes a shallow copy only
```

We have a special syntax for constructing multi-dimensional object arrays with null initial values:

```
assert [ new Object[3], new Object[2], new
Object[1] ] as Object[]
    //usual syntax
assert [ new Object[3], new Object[3], new
Object[3] ] as Object[]
    //usual syntax when each constituent
array of equal size

def m= new Object[3][3]
    //special syntax when each constituent
array of equal size
(0..<m.size()).each{i->
    (0..<m[i].size()).each{j->
        assert m[i][j] == null
            //we can also subscript with special
syntax using consecutive indexes
    }
}
```

We must specify the size of at least the first, outermost, dimension of an object array when we first create it:

```
//ar= new Object[] //compile error when
uncommented
ar= new Object[10][]
ar= new Object[10][][]
ar= new Object[10][10][]
```

A multidimensional array need not have arrays of the same length at each level. Thus, a triangular matrix may be created by:

```
def triangle= new Object[100][]
(0..<triangle.length).each{
  triangle[it] = new Object[it+1]
}
```

There are strict rules concerning evaluation when subscripting object arrays:

```
class MyException extends Exception{}
def exception(){ throw new MyException() }
def i, a, b

i= 4
a= new Object[i][i=3] //first subscript
evaluated before next one
assert a.size() == 4 && a[0].size() == 3

a= [ 11, 12, 13, 14 ] as Object[]
b= [ 3, 2, 1, 0 ] as Object[]
assert a[(a=b)[2]] == 12
//outside of subscript evaluated before
inside, ie, a[b[2]] or a[1] or 12
```

```
i= 1 //if what's outside subscript throws
exception, subscript isn't evaluated
try{ exception()[i=2] }catch(e){ assert i ==
1 }
```

```
i= 1
a= new Object[2][2]
    //if subscript evaluation throws
exception, subscripts to right not evaluated
try{ a[ exception() ][i=2] }catch(e){ assert
i == 1 }
```

```
//index evaluated before indexing occurs
(including checking whether
//what's outside subscript is null)...
a= null
try{ a[exception()]; assert 0 }catch(e){
assert e instanceof MyException }
    //NullPointerException never occurs here
i= 1
```

```
try{ a[i=2]; assert 0 }
catch(e){ assert i == 2 && e instanceof
NullPointerException }
```

#### Implementing an ArrayList with an Object Array

ArrayLists are implemented with object arrays internally. Each ArrayList instance has a capacity, the size of a fixed-size array used to store the elements. This array is always at least as large as the list size, and its capacity grows automatically as elements are added to the list. To see the internal capacity of lists constructed with various values:

```
class Extras{
  static enq(List l){ l.elementData.size() }
}
def measure= { list, times->
  def sizes= []
  times.times{
    def size
    use(Extras){ size= list.enq() }
    (size - list.size() + 1).times{ list <<
'a' }
    sizes << size
  }
  sizes
}

def list1= new ArrayList()
def measure1= measure(list1, 10)
assert measure1 == [10, 16, 25, 38, 58, 88,
133, 200, 301, 452]

def list2= new ArrayList(10)
def measure2= measure(list2, 10)
```

```
assert measure2 == measure1
```

```
def list3= new ArrayList(5)
```

```
def measure3= measure(list3, 10)
```

```
assert measure3 == [5, 8, 13, 20, 31, 47,  
71, 107, 161, 242]
```

```
def list4= []
```

```
def measure4= measure(list4, 10)
```

```
assert measure4 == [0, 1, 2, 4, 7, 11, 17,  
26, 40, 61]
```

```
def list5= new ArrayList(0)
def measure5= measure(list5, 10)
assert measure5 == measure4
```

For efficiency, we can increase the capacity of a list before adding a large number of elements:

```
class Extras{ static enq(List
l){l.elementData.size()} }
use(Extras){
  list= []
  list.ensureCapacity(200)
  assert list.enq() == 200
  list<< 'a'<< 'b'<< 'c'
  assert list.enq() == 200
  list.trimToSize()
  //we can also trim the internal
  fixed-size array to the list size
  assert list.enq() == 3
}
```

We can see how many times a list has been modified:

```
list= []<< 'a' << 'b'; assert list.modCount
== 2
list.remove('a'); assert list.modCount == 3
```

## JN1035-Maps

A map is a mapping from unique unordered keys to values:

```
def map= ['id':'FX-11', 'name':'Radish',
'no':1234, 99:'Y']
```

```

    //keys can be of any type, and mixed
    together; so can values
    assert map == ['name':'Radish',
    'id':'FX-11', 99:'Y', 'no':1234]
        //order of keys irrelevant
    assert map.size() == 4
    assert [1:'a', 2:'b', 1:'c' ] == [1:'c',
    2:'b'] //keys unique

    def map2= [
        'id': 'FX-17',
        name: 'Turnip', //string-keys that are
    valid identifiers need not be quoted
        99: 123, //any data can be a key
        (-97): 987, //keys with complex syntax
    must be parenthesized
        "tail's": true, //trailing comma OK
    ]

    assert map2.id == 'FX-17'
        //we can use field syntax for keys that
    are valid identifiers
    assert map2['id'] == 'FX-17' //we can always
    use subscript syntax
    assert map2.getAt('id') == 'FX-17' //some
    alternative method names
    assert map2.get('id') == 'FX-17'
    assert map2['address'] == null //if key
    doesn't exist in map
    assert map2.get('address', 'No fixed abode')
    == 'No fixed abode'

```

```

    //default value for non-existent keys

assert map2.class == null
    //field syntax always refers to value of
key, even if it doesn't exist
//use getClass() instead of class for
maps...
assert map2.getClass() == LinkedHashMap
//the kind of Map being used

assert map2."tail's" == true
    //string-keys that aren't valid
identifiers used as field by quoting them
assert ! map2.'99' && ! map2.'-97' //doesn't
work for numbers, though

map2.name = 'Potato'
map2[-107] = 'washed, but not peeled'
map2.putAt('alias', 'Spud')
    //different alternative method names
when assigning value
map2.put('address', 'underground')
assert map2.name == 'Potato' && map2[-107]
== 'washed, but not peeled' &&
    map2.alias == 'Spud' && map2.address
== 'underground'
assert map2 == [ id: 'FX-17', name:
'Potato', alias: 'Spud',
                address: 'underground', 99:
123, (-97): 987,
                (-107): 'washed, but not

```

```
peeled', "tail's": true ]
```

```
def id= 'address'
```

```
def map3= [id: 11, (id): 22]
```

```
//if we want a variable's value to  
become the key, we parenthesize it  
assert map3 == [id: 11, address: 22]
```

It's a common idiom to construct an empty map and assign values:

```
def map4= [:]  
map4[ 1 ]= 'a'  
map4[ 2 ]= 'b'  
map4[ true ]= 'p' //we can use boolean  
values as a key  
map4[ false ]= 'q'  
map4[ null ]= 'x' //we can also use null as  
a key  
map4[ 'null' ]= 'z'  
assert map4 == [1:'a', 2:'b', (true):'p',  
(false):'q', (null):'x', 'null':'z' ]
```

To use the value of a String as the key value of a map, simply surround the variable with parenthesis.

```
def foo = "test"  
def map = [(foo):"bar"]  
  
println map // will output ["test":"bar"]  
map = [foo:"bar"]  
println map // will output ["foo":"bar"]
```

We can use `each()` and `eachWithIndex()` to access keys and values:

```

def p= new StringBuffer()
[1:'a', 2:'b', 3:'c'].each{ p << it.key +':
'+ it.value +'; ' }
    //we supply a closure with either 1
param...
assert p.toString() == '1: a; 2: b; 3: c; '

def q= new StringBuffer()
[1:'a', 2:'b', 3:'c'].each{ k, v-> q << k
+': '+ v +'; ' } //...or 2 params
assert q.toString() == '1: a; 2: b; 3: c; '

def r= new StringBuffer()
[1:'a', 2:'b', 3:'c'].eachWithIndex{ it, i->
//eachIndex() always takes 2 params
    r << it.key +'(' + i +')': '+ it.value +'; '
}
assert r.toString() == '1(0): a; 2(1): b;
3(2): c; '

```

We can check the contents of a map with various methods:

```

assert [:].isEmpty()
assert ! [1:'a', 2:'b'].isEmpty()
assert [1:'a', 2:'b'].containsKey(2)
assert ! [1:'a', 2:'b'].containsKey(4)
assert [1:'a', 2:'b'].containsValue('b')
assert ! [1:'a', 2:'b'].containsValue('z')

```

We can clear a map:

```
def m= [1:'a', 2:'b']
m.clear()
assert m == [:]
```

Further map methods:

```
def defaults= [1:'a', 2:'b', 3:'c', 4:'d'],
overrides= [2:'z', 5:'x', 13:'x']
def result= new HashMap(defaults)
result.putAll(overrides)
assert result == [1:'a', 2:'z', 3:'c',
4:'d', 5:'x', 13:'x']
result.remove(2)
assert result == [1:'a', 3:'c', 4:'d',
5:'x', 13:'x']
result.remove(2)
assert result == [1:'a', 3:'c', 4:'d',
5:'x', 13:'x']
```

Great care must be exercised if mutable objects are used as map keys. The behavior of a map is not specified if the value of an object is changed in a manner that affects equals comparisons while the object is a key in the map. A special case of this prohibition is that a map should not contain itself as a key.

#### Collection views of a map

We can inspect the keys, values, and entries in a view:

```
def m2= [1:'a', 2:'b', 3:'c']

def es=m2.entrySet()
es.each{
  assert it.key in [1,2,3]
```

```
    assert it.value in ['a','b','c']
    it.value *= 3 //change value in entry
set...
}
assert m2 == [1:'aaa', 2:'bbb', 3:'ccc']
//...and backing map IS updated

def ks= m2.keySet()
assert ks == [1,2,3] as Set
ks.each{ it *= 2 } //change key...
assert m2 == [1:'aaa', 2:'bbb', 3:'ccc']
//...but backing map NOT updated
ks.remove( 2 ) //remove key...
assert m2 == [1:'aaa', 3:'ccc'] //...and
backing map IS updated

def vals= m2.values()
assert vals.toList() == ['aaa', 'ccc']
vals.each{ it = it+'z' } //change value...
assert m2 == [1:'aaa', 3:'ccc'] //...but
backing map NOT updated
vals.remove( 'aaa' ) //remove value...
assert m2 == [3:'ccc'] //...and backing map
IS updated

vals.clear() //clear values...
assert m2 == [:] //...and backing map IS
updated

assert es.is( m2.entrySet() ) //same
```

```
instance always returned
assert ks.is( m2.keySet() )
assert vals.is( m2.values() )
```

We can use these views for various checks:

```
def m1= [1:'a', 3:'c', 5:'e'], m2= [1:'a',
5:'e']
assert
m1.entrySet().containsAll(m2.entrySet())
    //true if m1 contains all of m2's
mappings
def m3= [1:'g', 5:'z', 3:'x']
m1.keySet().equals(m3.keySet()) //true if
maps contain mappings for same keys
```

These views also support the removeAll() and retainAll() operations:

```
def m= [1:'a', 2:'b', 3:'c', 4:'d', 5:'e']
m.keySet().retainAll( [2,3,4] as Set )
assert m == [2:'b', 3:'c', 4:'d']
m.values().removeAll( ['c','d','e'] as Set )
assert m == [2:'b']
```

Some more map operations:

```

def m= [1:'a', 2:'b', 3:'c', 4:'d', 5:'e']
assert [86: m, 99: 'end'].clone()[86].is( m
) //clone() makes a shallow copy

def c= []
def d= ['a', 'bb', 'ccc', 'dddd', 'eeeeee']
assert m.collect{ it.value * it.key } == d
assert m.collect(c){ it.value * it.key } ==
d
assert c == d

assert m.findAll{ it.key == 2 || it.value ==
'e' } == [2:'b', 5:'e']
def me= m.find{ it.key % 2 == 0 }
assert [me.key, me.value] in [ [2,'b'],
[4,'d'] ]

assert m.toMapString() == '[1:"a", 2:"b",
3:"c", 4:"d", 5:"e"]'

def sm= m.subMap( [2,3,4] )
sm[3]= 'z'
assert sm == [2:'b', 3:'z', 4:'d']
assert m == [1:'a', 2:'b', 3:'c', 4:'d',
5:'e'] //backing map is not modified

assert m.every{ it.value.size() == 1 }
assert m.any{ it.key % 4 == 0 }

```

```
def family = [dad:"John" , mom:"Jane",  
son:"John"]
```

```
def val = "John"
```

The simplest way to achieve this with the previous map:

```
assert family.find{it.value == "John"}?.key  
== "dad"  
//or  
assert family.find{it.value == val}?.key ==  
"dad"
```

Note that the return is only the key *dad*. As you can see from the *family Map* both *dad* and *son* are keys for the same values.

So, let's get all of the keys with the value "John"

Basically, `findAll` returns a collection of Mappings with the value "John" that we then iterate through and print the key *if* the key is groovy true.

This will place your results for the keys into a *List* of keys

```
def retVal = []  
family.findAll{it.value == val}.each{retVal  
<< it?.key}  
  
assert retVal == ["son", "dad"]
```

If you just wanted the collection of Mappings:

```

assert family.findAll{it.value == val} ==
["son":"John", "dad":"John"]
//or
def returnValue = family.findAll{it.value ==
val}
assert returnValue == ["son":"John",
"dad":"John"]

```

#### Special Notations

We can use special notations to access all of a certain key in a list of similarly-keyed maps:

```

def x = [ ['a':11, 'b':12], ['a':21, 'b':22]
]
assert x.a == [11, 21] //GPath notation
assert x*.a == [11, 21] //spread dot
notation

x = [ ['a':11, 'b':12], ['a':21, 'b':22],
null ]
assert x*.a == [11, 21, null] //caters for
null values
assert x*.a == x.collect{ it?.a }
//equivalent notation
try{ x.a; assert 0 }catch(e){ assert e
instanceof NullPointerException }
    //GPath doesn't cater for null values

class MyClass{ def getA(){ 'abc' } }
x = [ ['a':21, 'b':22], null, new MyClass()

```

```

]
assert x*.a == [21, null, 'abc']
//properties treated like map subscripting

def c1= new MyClass(), c2= new MyClass()
assert [c1, c2]*.getA() == [c1.getA(),
c2.getA()]
    //spread dot also works for method calls
assert [c1, c2]*.getA() == ['abc', 'abc']

assert ['z':900, *:['a':100, 'b':200],
'a':300] == ['a':300, 'b':200, 'z':900]
    //spread map notation in map definition
assert [ *:[3:3, *:[5:5] ], 7:7] == [3:3,
5:5, 7:7]

def f(){ [ 1:'u', 2:'v', 3:'w' ] }
assert [*:f(), 10:'zz'] == [1:'u', 10:'zz',
2:'v', 3:'w']
    //spread map notation in function
arguments
def f(m){ m.c }
assert f(*:['a':10, 'b':20, 'c':30], 'e':50)
== 30

def f(m, i, j, k){ [m, i, j, k] }
    //using spread map notation with mixed
unnamed and named arguments
assert f('e':100, *[4, 5], *:['a':10,

```

```
'b':20, 'c':30], 6) ==  
  [ ["e":100, "b":20, "c":30, "a":10], 4,  
5, 6 ]
```

#### Grouping

We can group a list into a map using some criteria:

```
assert [ 'a', 7, 'b', [2,3] ].groupBy{  
it.class } == [  
  (String.class): ['a', 'b'],  
  (Integer.class): [ 7 ],  
  (ArrayList.class): [[2,3]]  
]  
  
assert [  
  [name:'Clark', city:'London'],  
  [name:'Sharma', city:'London'],  
  [name:'Maradona', city:'LA'],  
  [name:'Zhang', city:'HK'],  
  [name:'Ali', city:'HK'], [name:'Liu',  
city:'HK'],  
].groupBy{ it.city } == [  
  London: [ [name:'Clark', city:'London'],  
            [name:'Sharma', city:'London']  
],  
  LA: [ [name:'Maradona', city:'LA'] ],  
  HK: [ [name:'Zhang', city:'HK'],  
        [name:'Ali', city:'HK'],  
        [name:'Liu', city:'HK'] ],  
]
```

By using `groupBy()` and `findAll()` on a list of similarly-keyed maps, we can emulate SQL:

```
assert ('The quick brown fox jumps over the
lazy dog'.toList()*
        toLowerCase() - ' ').
    findAll{ it in 'aeiou'.toList() }.
    //emulate SQL's WHERE clause with
findAll() method
    groupBy{ it }.
    //emulate GROUP BY clause with
groupBy() method
    findAll{ it.value.size() > 1 }.
    //emulate HAVING clause with findAll()
method after the groupBy() one
    entrySet().sort{ it.key }.reverse().
    //emulate ORDER BY clause with sort()
and reverse() methods
    collect{ "$it.key:${it.value.size()}"
}.join(', ') == 'u:2, o:4, e:3'
```

An example with more than one "table" of data:

```

//find all letters in the "lazy dog"
sentence appearing more often than those
//in the "liquor jugs" one...
def dogLetters= ('The quick brown fox jumps
over the lazy dog'.toList)*.
                    toLowerCase() - ' '),
    jugLetters= ('Pack my box with five
dozen liquor jugs'.toList)*.
                    toLowerCase() - ' ')
assert dogLetters.groupBy{ it }.
    findAll{ it.value.size() >
jugLetters.groupBy{ it }[ it.key ].size()
}.
    entrySet().sort{it.key}.collect{
"$it.key:${it.value.size()}" }.join(', ') ==
    'e:3, h:2, o:4, r:2, t:2'

```

#### HashMap Internals

A HashMap is constructed in various ways:

```

def map1= new HashMap() //uses initial
capacity of 16 and load factor of 0.75
def map2= new HashMap(25) //uses load factor
of 0.75
def map3= new HashMap(25, 0.8f)
def map4= [:] //the shortcut syntax

```

The capacity is the number of buckets in the HashMap, and the initial capacity is the capacity when it's created. The load factor measures how full the HashMap will get before its capacity is automatically increased. When the number of entries exceeds the product of the load factor and the current capacity, the HashMap is reshaped so it has about twice the number of buckets. A HashMap gives constant-time performance for lookup (getting and putting). Iterating

over collection views gives time performance proportional to the capacity of the HashMap instance plus its the number of keys. So don't set the initial capacity too high or the load factor too low. As a general rule, the default load factor (0.75) offers a good tradeoff between time and space costs. Higher values decrease the space overhead but increase the lookup cost. Creating a HashMap with a sufficiently large capacity will allow mappings to be stored more efficiently than letting it perform automatic rehashing as needed to grow the table.

A HashSet is implemented with a HashMap, and is constructed with the same choices of parameters:

```
def set1= new HashSet() //uses initial  
capacity of 16 and load factor of 0.75  
def set2= new HashSet(25) //uses load factor  
of 0.75  
def set3= new HashSet(25, 0.8f)  
def set4= Collections.newSetFromMap( [:] )  
//we can supply our own empty map for  
the implementation
```

#### Sorted Maps

A sorted map is one with extra methods that utilize the sorting of the keys. Some constructors and methods:

```
def map= [3:'c', 2:'d' ,1:'e', 5:'a',  
4:'b'], tm= new TreeMap(map)  
assert tm.firstKey() == map.keySet().min()  
&& tm.firstKey() == 1  
assert tm.lastKey() == map.keySet().max() &&  
tm.lastKey() == 5  
assert tm.findIndexOf{ it.key==4 } == 3
```

We can construct a TreeMap by giving a comparator to order the elements in the map:

```

def c= [ compare:
  {a,b-> a.equals(b)? 0:
Math.abs(a)<Math.abs(b)? -1: 1 }
] as Comparator

def tm= new TreeMap( c )
tm[3]= 'a'; tm[-7]= 'b'; tm[9]= 'c'; tm[-2]=
'd'; tm[-4]= 'e'
assert tm == new TreeMap( [(-2):'d', 3:'a',
(-4):'e', (-7):'b', 9:'c'] )
assert tm.comparator() == c //retrieve the
comparator

def tm2= new TreeMap( tm ) //use same map
entries and comparator
assert tm2.comparator() == c

def tm3= new TreeMap( tm as HashMap )
  //special syntax to use same map entries
but default comparator only
assert tm3.comparator() == null

```

The range-views, headMap() tailMap() and subMap(), are useful views of the items in a sorted map. They act similarly to the corresponding range-views in a sorted set.

```

def sm= new TreeMap(['a':1, 'b':2, 'c':3,
'd':4, 'e':5])
def hm= sm.headMap('c')
assert hm == new TreeMap(['a':1, 'b':2])
    //headMap() returns all elements with
key < specified key
hm.remove('a')
assert sm == new TreeMap(['b':2, 'c':3,
'd':4, 'e':5])
    //headmap is simply a view of the data
in sm
sm['a']= 1; sm['f']= 6
assert sm == new TreeMap(['a':1, 'b':2,
'c':3, 'd':4, 'e':5, 'f':6])
    //if backing sorted map changes, so do
range-views
def tm= sm.tailMap('c')
assert tm == new TreeMap(['c':3, 'd':4,
'e':5, 'f':6])
    //tailMap() returns all elements with
key >= specified element
def bm= sm.subMap('b','e')
assert bm == new TreeMap(['b':2, 'c':3,
'd':4])
    //subMap() returns all elements with key
>= but < specified element
try{ bm['z']= 26; assert 0 }
catch(e){ assert e instanceof
IllegalArgumentException }
    //attempt to insert an element out of
range

```

## Immutable Maps

We can convert a map into one that can't be modified:

```
def imMap= (['a':1, 'b':2, 'c':3] as
Map).asImmutable()
try{ imMap['d']= 4; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
imMap= Collections.unmodifiableMap( ['a':1,
'b':2, 'c':3] as Map )
    //alternative way
try{ imMap['d']= 4; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }

def imSortedMap= ( new TreeMap(['a':1,
'b':2, 'c':3]) ).asImmutable()
try{ imSortedMap['d']= 4; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
imSortedMap=
Collections.unmodifiableSortedMap(
    new TreeMap(['a':1, 'b':2, 'c':3])
)    //alternative way
try{ imSortedMap['d']= 4; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
```

We can create an empty map that can't be modified:

```
def map= Collections.emptyMap()
assert map == [:]
try{ map['a']= 1; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
map= Collections.EMPTY_MAP
assert map == [:]
try{ map['a']= 1; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
```

We can create a single-element list that can't be modified:

```
def singMap = Collections.singletonMap('a',
1)
assert singMap == ['a': 1]
try{ singMap['b']= 2; assert 0 }
catch(e){ assert e instanceof
UnsupportedOperationException }
```

#### Observable Maps

We can convert a map into an observable one with the 'as' keyword too. An observable map will trigger a `PropertyChangeEvent` every time a value changes:

```

// don't forget the imports
import java.beans.*
def map = [:] as ObservableMap
map.addPropertyChangeListener({ evt ->
    println "${evt.propertyName}:
    ${evt.oldValue} -> ${evt.newValue}"
} as PropertyChangeListener)

map.key = 'value' // prints key: null ->
value
map.key = 'Groovy' // prints key: value ->
Groovy

```

We can also wrap an existing map with an ObservableMap

```

import java.beans.*
def sorted = [a:1,b:2] as TreeMap
def map = new ObservableMap(sorted)
map.addPropertyChangeListener({ evt ->
    println "${evt.propertyName}:
    ${evt.oldValue} -> ${evt.newValue}"
} as PropertyChangeListener)
map.key = 'value'
assert ['a', 'b', 'key'] == (sorted.keySet()
as List)
assert ['a', 'b', 'key'] == (map.keySet() as
List)

```

Lastly we can specify a closure as an additional parameter, it will work like a filter for properties that should or should not trigger a PropertyChangeEvent when their values change, this is useful in conjunction with Expando. The

filtering closure may take 2 parameters (the property name and its value) or less (the value of the property).

```
import java.beans.*
def map = new ObservableMap({!(it instanceof
Closure)})
map.addPropertyChangeListener({ evt ->
    println "${evt.propertyName}:
${evt.oldValue} -> ${evt.newValue}"
} as PropertyChangeListener)
def bean = new Expando( map )
bean.lang = 'Groovy' // prints lang: null
-> Groovy
bean.sayHello = { name -> "Hello ${name}" }
// prints nothing, event is skipped
assert 'Groovy' == bean.lang
assert 'Hello Groovy' ==
bean.sayHello(bean.lang)
```

Infinite nested maps tip

With the help of [withDefault](#) we can easily create infinite nested maps structure :

```
Map infiniteNestedMaps = { [:].withDefault{
owner.call() } }()

infiniteNestedMaps.a.b.c.d.e.f.g = 42

assert infiniteNestedMaps.a.b.c.d.e.f.g ==
42
```

## JN1515-Characters

A Character is a single token from the Unicode basic multilingual plane. It can also convert to the lowermost 16 bits

of an integer.

```
assert Character.SIZE == 16 &&
Character.SIZE == Short.SIZE //16 bits in
size
assert Character.MIN_VALUE as int == 0x0000
assert Character.MAX_VALUE as int == 0xFFFF
assert Character.TYPE == char //often, we
can write 'char' instead
```

Each Unicode character belongs to a certain category, which we can inspect using getType():

```
def categories= [
  'LOWERCASE_LETTER',
    //unicode category "Ll": a lowercase
letter that has an uppercase variant
  'UPPERCASE_LETTER',
    //Lu: an uppercase letter that has a
lowercase variant
  'TITLECASE_LETTER',
    //Lt: a letter beginning a word with
only the first letter capitalized
  'MODIFIER_LETTER',
    //Lm: a special character that is used
like a letter
  'OTHER_LETTER',
    //Lo: a letter or ideograph not having
lowercase and uppercase variants

  'NON_SPACING_MARK',
    //Mn: a combining character that doesnt
```

take up space (eg accents, umlauts)  
'COMBINING\_SPACING\_MARK',  
    //Mc: a combining character that takes  
up space (eg vowel signs in the East)  
'ENCLOSING\_MARK',  
    //Me: an enclosing character (eg circle,  
square, keycap)

'SPACE\_SEPARATOR',  
    //Zs: an invisible whitespace character  
that takes up space  
'LINE\_SEPARATOR',  
    //Zl: line separator character 0x2028  
'PARAGRAPH\_SEPARATOR',  
    //Zp: paragraph separator character  
0x2029

'MATH\_SYMBOL',        //Sm: any mathematical  
symbol  
'CURRENCY\_SYMBOL', //Sc: any currency sign  
'MODIFIER\_SYMBOL',  
    //Sk: a combining character that's also  
a full character on its own  
'OTHER\_SYMBOL',  
    //So: various other symbols (eg  
dingbats, box-drawing)

'DECIMAL\_DIGIT\_NUMBER',  
    //Nd: a digit zero through nine in any  
script except ideographic scripts  
'LETTER\_NUMBER',

//Nl: a number that looks like a letter  
(eg Roman numerals)

'OTHER\_NUMBER',

//No: a superscript or subscript digit,  
or number that's not a digit 0..9

//

(excluding from ideographic scripts)

'DASH\_PUNCTUATION', //Pd: any kind of  
hyphen or dash

'START\_PUNCTUATION', //Ps: any kind of  
opening bracket

'END\_PUNCTUATION', //Pe: any kind of  
closing bracket

'INITIAL\_QUOTE\_PUNCTUATION', //Pi: any  
kind of opening quote

'FINAL\_QUOTE\_PUNCTUATION', //Pf: any  
kind of closing quote

'CONNECTOR\_PUNCTUATION',  
//Pc: a punctuation character that  
connects words (eg underscore)

'OTHER\_PUNCTUATION',  
//Po: any other kind of punctuation  
character

'FORMAT', //Cf: invisible formatting  
indicator

'CONTROL',  
//Cc: 65 ISO control characters  
(0x00..0x1F and 0x7F..0x9F)

'PRIVATE\_USE', //Co: any code point

```
reserved for private non-unicode use
  'SURROGATE', //Cs: one half of a surrogate
pair
  'UNASSIGNED', //Cn: any code point to
which no character has been assigned
]
```

```
def stats= (0x0000..0xFFFF).groupBy{
Character.getType(it) }
stats.entrySet().sort{ it.value.size
}.reverse().each{ cat->
  def keyName= Character.fields.
    find{ it.get() == cat.key
```

```
&& it.name in categories }.name  
    println "$keyName: $cat.value.size"  
}
```

The surrogate category is divided into the high surrogates and the low surrogates. A Unicode supplementary character is represented by two Characters, the first from the high surrogates, the second from the low. Integers, known as code points, can also represent all Unicode characters, including supplementary ones. The code point is the same as a Character converted to an integer for basic plane characters, and its values continue from 0x10000 for supplementary characters. The upper 11 bits of the code point Integer must be zeros. Methods accepting only char values treat surrogate characters as undefined characters.

```
assert Character.MIN_HIGH_SURROGATE ==  
0xD800 &&  
    Character.MIN_SURROGATE == 0xD800  
assert Character.MAX_HIGH_SURROGATE ==  
0xDBFF  
assert Character.MIN_LOW_SURROGATE == 0xDC00  
assert Character.MAX_LOW_SURROGATE == 0xDFFF  
&&  
    Character.MAX_SURROGATE == 0xDFFF  
assert Character.isSurrogatePair(  
Character.MIN_HIGH_SURROGATE,  
  
Character.MIN_LOW_SURROGATE )  
assert Character.isHighSurrogate(  
Character.MIN_HIGH_SURROGATE )  
assert Character.isLowSurrogate(  
Character.MIN_LOW_SURROGATE )  
  
assert Character.MIN_CODE_POINT == 0x0000  
assert  
Character.MIN_SUPPLEMENTARY_CODE_POINT ==  
0x10000 //an integer
```

```
assert Character.MAX_CODE_POINT == 0x10FFFF
assert Character.isValidCodePoint(
Character.MIN_CODE_POINT )
assert ! Character.isValidCodePoint(
Character.MAX_CODE_POINT + 1 )
assert Character.isSupplementaryCodePoint(
Character.MIN_SUPPLEMENTARY_CODE_POINT )
assert ! Character.isSupplementaryCodePoint(
Character.MIN_SUPPLEMENTARY_CODE_POINT - 1 )

assert Character.charCount(0xFFFF) == 1
    //number of Characters needed to
represent a certain integer as Unicode
assert Character.charCount(0x10FFFF) == 2

assert Character.isDefined(0xFFFD)
```

```
assert ! Character.isDefined(0xFFFF)
//doesn't include unassigned characters
assert Character.isDefined(0x10000)
```

To convert a Unicode character between a code point and a Character array:

```
def minLowSurr= Character.MIN_LOW_SURROGATE,
    maxLowSurr= Character.MAX_LOW_SURROGATE,
    minHighSurr=
Character.MIN_HIGH_SURROGATE,
    maxHighSurr=
Character.MAX_HIGH_SURROGATE
assert Character.toChars(0xFFFF).collect{ it
as int }.toList() == [0xFFFF]
    //convert integer into array of
Characters
assert Character.toChars(0x10000).collect{
it as int }.toList() ==
    [minHighSurr as int, minLowSurr as int]
assert Character.toChars(0x10FFFF).collect{
it as int }.toList() ==
    [maxHighSurr as int, maxLowSurr as int]

def charArray= new char[6] //an array that
can only contain Characters
assert Character.toChars(0x10000, charArray,
2) == 2 &&
    charArray.collect{ it as int
}.toList() ==
                                [0, 0, minHighSurr
as int, minLowSurr as int, 0, 0]
```

```
charArray= new char[4]
assert Character.toChars(0xFFFF, charArray,
1) == 1 &&
    charArray.collect{ it as int
}.toList() == [0, 0xFFFF, 0, 0]

assert Character.toCodePoint(minHighSurr,
```

```
minLowSurr) == 0x10000
    //converts surrogate pair to integer
representation
```

We can enquire of code points in a char array or string:

```
def minLowSurr= Character.MIN_LOW_SURROGATE,
    minHighSurr=
Character.MIN_HIGH_SURROGATE

def ca1= ['a', 'b', 'c', minHighSurr,
minLowSurr, 'e', 'f', 'g'] as char[]
def ca2= ['a', 'b', 'c', 0xFFFF, 'e', 'f',
'g'] as char[]
assert Character.codePointAt(ca1, 3) ==
0x10000
    //beginning at index 3, look at as many
chars as needed
assert Character.codePointAt(ca2, 3) ==
0xFFFF
assert Character.codePointAt(ca1, 3, 4) ==
minHighSurr
    //extra parameter limits sequence of
chars to index <4
assert Character.codePointAt(ca2, 3, 4) ==
0xFFFF
assert Character.codePointBefore(ca1, 4) ==
minHighSurr
assert Character.codePointBefore(ca1, 5) ==
0x10000
    //if low surrogate, look back more for
```

```
high one, and use both
assert Character.codePointBefore(ca1, 5, 4)
== minLowSurr
    //extra param limits lookback to index
    >=4
assert Character.codePointCount(ca1, 1, 5)
== 4
    //number of code points in a subarray
    given by offset 1 and count 5
assert Character.codePointCount(ca1, 1, 4)
== 3
    //lone high surr counted as 1 code point
assert Character.offsetByCodePoints(ca1, 0,
6, 1, 3) == 5
    //index of ca1[0..<6] that's offset by 3
    code points

//versions of these methods exist for
strings...
def s1= 'abc'+ minHighSurr + minLowSurr
+'efg'
def s2= 'abcdefg'
assert Character.codePointAt(s1, 3) ==
0x10000
    //if high surrogate, add on low
surrogate
assert Character.codePointAt(s1, 4) ==
minLowSurr
    //if low surrogate, use it only
assert Character.codePointAt(s1, 5) == 'e'
as int
```

```
assert Character.codePointAt(s2, 3) == 'd'
as int
    //enquire code point in string
assert Character.codePointBefore(s1, 4) ==
minHighSurr
assert Character.codePointBefore(s1, 5) ==
0x10000
    //if low surrogate, look back more for
high one, and use both
assert Character.codePointCount(s1, 1, 5) ==
3
    //number of code points in a substring
with indexes >=1 and <5
assert Character.offsetByCodePoints(s1, 1,
```

```
3) == 5
```

```
    //index from 1 that's offset by 3 code  
points
```

Every character also has a directionality:

```
def directionalities= [:]  
Character.fields.each{  
    if( it.name =~ /^DIRECTIONALITY_/ )  
directionalities[ it.get() ]= it.name  
}  
  
def stats= (0x0000..0xFFFF).groupBy{  
Character.getDirectionality(it) }  
    //will also work for supplementary chars  
stats.entrySet().sort{ it.value.size  
}.reverse().each{ dir->  
    def keyName= Character.fields.  
        find{ it.get() == dir.key && it.name  
in directionalities.values() }.name  
    println "$keyName: $dir.value.size"  
}
```

Every character is part of a Unicode block:

```

(0x0000..0xFFFF).groupBy{
Character.UnicodeBlock.of( it as char ) }.
    entrySet().sort{it.value.size}.reverse().
    each{ println "$it.key: $it.value.size" }
//this uses basic plane only

//this one uses supplementary characters
also...
(0x0000..0x10FFFF).groupBy{
Character.UnicodeBlock.of( it as int ) }.
    entrySet().sort{it.value.size}.reverse().
    each{ println "$it.key: $it.value.size" }

try{ Character.UnicodeBlock.of( 0x110000 );
assert 0 }
catch(e){ assert e instanceof
IllegalArgumentException }

```

Character assists integers using different radixes:

```

assert Character.MIN_RADIX == 2
    //the minimum and maximum radices
available for conversion to/from strings
assert Character.MAX_RADIX == 36 //0 to 9,
and A to Z
assert Character.forDigit(12, 16) == 'c'
    //character representation for a digit
in a certain radix
assert Character.digit('c' as char, 16) ==
12
    //digit of a character rep'n in a
certain radix

```

We can find the Unicode block for a loosely-formatted textual description of it:

```

[ 'BASIC LATIN', 'basic latin',
'BasicLatin', 'baSiCLaTiN', 'BaSiC LaTiN',
    'BASIC_LATIN',
'BaSiC_LaTiN' ].
    each{ assert
Character.UnicodeBlock.forName(it).toString(
) == 'BASIC_LATIN' }

```

#### Constructing and Using Characters

We can't represent Characters directly in our programs, but must construct them from a string:

```

assert 'a'.class == String
def c1= 'a' as char, c2= (char)'b'
//constructing
def c3= new Character(c2), c4=
c2.charValue() //cloning
[c1, c2, c3, c4].each{ assert it.class ==
Character }
assert c2 == c3 && c1 != c2
assert c1 < c2 && c1.compareTo(c2) == -1
    //comparing works just the same as for
numbers
assert c2.toString().class == String

```

There's a number of Character utility methods, accepting either a code point or a basic-plane character, that test some attribute of the character:

```

def categories= [
    'digit': { Character.isDigit(it) },
    'letter': { Character.isLetter(it) },
    'letter or digit': {
Character.isLetterOrDigit(it) },
    'identifier ignorable': {
Character.isIdentifierIgnorable(it) },
    //an ignorable character in a Java or
Unicode identifier
    'ISO control': {
Character.isISOControl(it) }, //an ISO
control character
    'Java identifier part': {
Character.isJavaIdentifierPart(it) },

```

```

    //be part of a Java identifier as other
    than the first character
    'Java identifier start': {
Character.isJavaIdentifierStart(it) },
    //permissible as the first character in
    a Java identifier
    'Unicode identifier part': {
Character.isUnicodeIdentifierPart(it) },
    //be part of a Unicode identifier other
    than first character
    'Unicode identifier start': {
Character.isUnicodeIdentifierStart(it) },
    //permissible as first character in a
    Unicode identifier
    'lower case': { Character.isLowerCase(it)
},
    'upper case': { Character.isUpperCase(it)
},
    'title case': { Character.isTitleCase(it)
},
    'space char': { Character.isSpaceChar(it)
}, //a Unicode space character
    'whitespace': { Character.isWhitespace(it)
}, //white space according to Java
    'mirrored': { Character.isMirrored(it) },
    //mirrored according to the Unicode spec
]
def stats= [:]
categories.keySet().each{ stats[it]= 0 }
(0x0000..0xFFFF).each{ch-> //also works with
supplementaries (0x0000..0x10FFFF)

```

```
categories.each{cat->
  if( cat.value(ch) ) stats[ cat.key ] +=
1
  }
}
```

```
stats.entrySet().sort{ it.value
}.reverse().each{ println "$it.key:
$it.value" }
```

We can use characters instead of numbers in arithmetic operations:

```

assert 'a' as char == 97 && 'd' as char ==
100
assert ('a' as char) + 7 == 104 && 7 + ('a'
as char) == 104
    //either first or second arg
assert ('a' as char) + ('d' as char) == 197
//two chars
assert ('a' as char).plus(7) == ('a' as
char) + 7 //alternative method name
assert ('a' as char) - 27 == 70 && ('a' as
char).minus(27) == 70
assert ('a' as char) * ('d' as char) == 9700
&&
    ('a' as char).multiply('d' as char)
== 9700
assert 450 / ('d' as char) == 4.5 &&
450.div('d' as char) == 4.5
assert 420.intdiv('d' as char) == 4

assert ('a' as char) > 90 && ('a' as
char).compareTo(90) == 1
assert 90 < ('a' as char) &&
90.compareTo('a' as char) == -1
assert ('a' as char) == ('a' as char) &&
    ('a' as char).compareTo('a' as char)
== 0

```

We can auto-increment and -decrement characters:

```
def c= 'p' as char
assert c++ == 'p' as char && c == 'q' as
char &&
    c-- == 'q' as char && c == 'p' as
char &&
    ++c == 'q' as char && c == 'q' as
char &&
    --c == 'p' as char && c == 'p' as
char
assert c.next() == 'q' && c.previous() ==
'o' && c == 'p'
```

Some miscellaneous methods:

```
assert Character.getNumericValue('6' as
char) == 6
assert Character.reverseBytes(0x37ae as
char) == 0xae37 as char

assert Character.toUpperCase('a' as char) ==
'A' as char
assert Character.toLowerCase('D' as char) ==
'd' as char
assert Character.titleCase('a' as char) ==
'A' as char
```

## JN1525-Strings

We can use either single- or double-quotes around strings:

```
assert 'hello, world' == "hello, world"
assert "Hello, Groovy's world" == 'Hello,
Groovy\'s world'
    //backslash escapes the quote
assert 'Say "Hello" to the world' == "Say
\"Hello\" to the world"
```

Backslashes can escape other characters in Strings. We can use letter codes (eg '\b') or octal codes (eg '\010'):

```
assert '\b' == '\010' //backspace
assert '\t' == '\011' //horizontal tab
assert '\n' == '\012' //linefeed
assert '\f' == '\014' //form feed
assert '\r' == '\015' //carriage return
assert '\\ ' == '\\ ' //use backslash to
escape the backslash
```

To span multiple lines, use either triple quotes or a backslash at the end of the continuing lines to join them with the next:

```
assert '''hello,
world''' == 'hello,\nworld'
    //triple-quotes for multi-line strings,
adds '\n' regardless of host system
assert 'hello, \
world' == 'hello, world' //backslash joins
lines within string
```

We can also use three double-quotes.

```
def text = """\
Good morning.
Good night again."""
```

When using double-quotes, either one or three, we can embed code within them using \$. Here, they're called GStrings:

```
def name = 'Groovy'
assert "hello $name, how are you today?" ==
"hello Groovy, how are you today?"
```

Anything more complex than a variable name must be surrounded by curlies:

```
def a = 'How are you?'
assert "The phrase '$a' has length
${a.size()}" ==
"The phrase 'How are you?' has length
12"
```

We can change the variable's value in the GString:

```
def i= 1, list= []
3.times{ list<< "${i++}" }
assert list.join() == '123'
```

#### String methods

We can convert other objects in Groovy to their string representation in different ways:

```
def o= new Object()
assert String.valueOf( o ) == o.toString()
//this works for any object in Groovy
assert String.valueOf( true ) ==
true.toString() //boolean value
assert String.valueOf( 'd' as char ) == ('d'
as char).toString() //character
assert String.valueOf( 7.5d ) ==
7.5d.toString() //double
assert String.valueOf( 8.4f ) ==
8.4f.toString() //float
assert String.valueOf( 13i ) ==
13i.toString() //integer
assert String.valueOf( 14L ) ==
14L.toString() //long
assert String.valueOf( ['a', 'b', 'c'] ) ==
['a', 'b', 'c'].toString()
    //list, etc, etc, etc
```

To find the size and substrings:

```
def s= 'abcdefg'
assert s.length() == 7 && s.size() == 7
assert s.substring(2,5) == 'cde' &&
s.substring(2) == 'cdefg'
assert s.subSequence(2,5) == 'cde'
```

There's different ways to construct a string:

```

assert new String() == ''
assert new String('hello') == 'hello'

def minLowSurr= Character.MIN_LOW_SURROGATE,
    minHighSurr=
Character.MIN_HIGH_SURROGATE
def str= 'abc' + minHighSurr + minLowSurr +
'efg'
def ca= ['a', 'b', 'c', minHighSurr,
minLowSurr, 'e', 'f', 'g'] as char[]
def ia= ['a', 'b', 'c', 0x10000, 'e', 'f',
'g'] as int[]
assert new String(ca) == str
assert new String(ca, 2, ca.size()-2) ==
str[2..-1]
assert new String(ia, 2, ia.size()-2) ==
str[2..-1]

def ca2= new char[8]
str.getChars(0, str.size(), ca2, 0)
    //copy characters from string into
character array
assert ca2.size() == str.size()
ca2.eachWithIndex{ elt, i-> assert elt ==
str[i] }

def ca3= ['a', 'b', 'c', 'd', 'e'] as char[]
'abcde'.toCharArray().eachWithIndex{ it, i->
assert it == ca3[i] }
    //convert String to char array
assert String.valueOf(ca3) == 'abcde'

```

```
//convert char array to String
assert String.copyOfValueOf(ca3) == 'abcde'
//alternative method name
```

```
assert String.valueOf(ca3, 2, 2) == 'cd'  
//use substring  
assert String.copyValueOf(ca3, 2, 2) == 'cd'
```

We can pad and center strings:

```
assert  
'hello'.padRight(8, '+').padLeft(10, '+') ==  
'++hello+++'  
assert 'hello'.padLeft(7).padRight(10) == '  
hello    '  
assert 'hello'.center(10, '+').center(14, '  
' ) == '  ++hello++  '
```

We can split a string into tokens:

```

assert 'he she\t it'.tokenize() == ['he',
'she', 'it']
    //tokens for split are ' \t\n\r\f'
assert 'he she\t it'.tokenize() ==
    new StringTokenizer('he she\t
it').collect{ it }

assert 'he, she; it, ; they'.tokenize(', ;') ==
['he', 'she', 'it', 'they']
    //supply our own tokens
assert new
StringTokenizer('he, she; it, ; they',
', ;').collect{ it } ==
    'he, she; it, ; they'.tokenize(', ;')

assert new StringTokenizer('he, she, ; it',
', ;', true).collect{ it } ==
    ['he', ',', 'she', ',', ';', 'it']
    //long form provides extra option to
return the tokens with the split-up data

```

Some additional methods:

```

assert 'abcde'.find{ it > 'b' } == 'c'
//first one found
assert 'abcde'.findAll{ it > 'b' } == ['c',
'd', 'e'] //all found
assert 'abcde'.findIndexOf{ it > 'c' } == 3
//first one found

```

```
assert 'abcde'.every{ it < 'g' } && !
'abcde'.every{ it < 'c' }
assert 'abcde'.any{ it > 'c' } && !
'abcde'.any{ it > 'g' }

assert 'morning'.replace('n','t') ==
'mortitg' &&
    'boo'.replace('o', 'at') == 'batat'
&&
    'book'.replace('oo','ie') == 'biek'

assert 'EggS'.toLowerCase() == 'eggs' &&
'EggS'.toUpperCase() == 'EGGS'
assert '    Bacon    '.trim() == 'Bacon'
assert 'noodles'.startsWith('nood') &&
'noodles'.endsWith('dles')
assert 'corn soup'.startsWith('rn', 2) //2
is offset

assert 'abc'.concat('def') == 'abcdef'
assert 'abcdefg'.contains('def')
assert ''.isEmpty() && ! 'abc'.isEmpty()

assert 'morning'.indexOf('n') == 3
assert 'morning'.indexOf('n', 4) == 5
//ignore first 4 characters
assert 'morning'.indexOf('ni') == 3
assert 'morning'.indexOf('ni', 4) == -1
//not found
assert 'morning'.lastIndexOf('n') == 5
assert 'morning'.lastIndexOf('n', 4) == 3
```

```
//only search first 4 characters  
assert 'morning'.lastIndexOf('ni') == 3
```

```
assert 'morning'.lastIndexOf('ni', 4) == 3
    //only search first 4 characters for
first char of search string
```

We can use operators on strings:

```
assert 'hello, ' + 'balloon' - 'lo' == 'hel,
balloon'
    //'-' subtracts one instance at most of
string
assert 'hello, balloon' - 'abc' == 'hello,
balloon'
assert 'hello, '.plus('balloon').minus('lo')
== 'hel, balloon'
    //alternative method syntax
assert 'value is ' + true == 'value is true'
&&
    'value is ' + 1.54d == 'value is
1.54' &&
        //first converts double to
String (without info loss)
    'value is ' + 7 == 'value is 7' //we
can add on various types of values
assert 7 + ' is value' == '7 is value'
assert 'telling true lies' - true ==
'telling lies' &&
    'week has 7 days' - 7 == 'week has
days'
        //we can subtract various types
of values
assert 'a' * 3 == 'aaa' && 'a'.multiply(3)
```

```
== 'aaa'
```

```
assert 'hello'.reverse() == 'olleh'
```

```
assert 'hello'.count('l') == 2
```

```
assert 'abc'.collect{ it * 2 } == ['aa',  
'bb', 'cc']
```

```
def s= [], t= [:]
```

```
'abc'.each{ s << it }
```

```
'abc'.eachWithIndex{ elt, i-> t[i]= elt }
```

```
assert s == ['a', 'b', 'c'] && t == [0:'a',  
1:'b', 2:'c']
```

```
assert 'abcde'.toList() == ['a', 'b', 'c',  
'd', 'e']
```

```
assert 'abc'.next() == 'abd' &&
'abc'.previous() == 'abb'
```

We can subscript strings just as we can lists, except of course strings are read-only:

```
assert 'abcdefg'[ 3 ] == 'd'
assert 'abcdefg'.getAt( 3 ) == 'd'
//equivalent method name
assert 'abcdefg'.charAt( 3 ) == 'd'
//alternative method name
assert 'abcdefg'[ 3..5 ] == 'def'
assert 'abcdefg'.getAt( 3..5 ) == 'def'
assert 'abcdefg'[ 1, 3, 5, 6 ] == 'bdfg'
assert 'abcdefg'[ 1, *3..5 ] == 'bdef'
assert 'abcdefg'[ 1, 3..5 ] == 'bdef'
    //range in subscript flattened
automatically
assert 'abcdefg'[-5..-2] == 'cdef'
assert 'abcdefg'.getAt( [ 1, *3..5 ] ) ==
'bdef'
assert 'abcdefg'.getAt( [ 1, 3..5 ] ) ==
'bdef'

assert 'abcde' == 'ab' + 'c' + 'de'
assert 'abcde'.equals('ab' + 'c' + 'de')
//equivalent method name
assert 'abcde'.contentEquals('ab' + 'c' +
'de') //alternative method name
assert 'AbcdE'.equalsIgnoreCase('ABCDE')
assert 'abcde' < 'abcdf' && 'abcde' <
```

```
'abcdef'
assert 'abcde'.compareTo('abcdf') == -1 &&
'abcde'.compareTo('abcdef') == -1

//equivalent method
assert 'AbcdEF'.compareToIgnoreCase('aBCDe')
== 1
assert 'AbcdE'.compareToIgnoreCase('aBCDef')
== -1

assert Collections.max( 'abC'.toList(),
String.CASE_INSENSITIVE_ORDER ) == 'C'
assert Collections.min(
    ['abC', 'ABd', 'AbCd'],
String.CASE_INSENSITIVE_ORDER ) == 'abC'

assert 'abcde'.regionMatches(2, 'cccd', 3,
2)
    //match from index 2 in 'abcde' to 2
chars from index 3 in 'cccd'
```

```
assert 'abcDE'.regionMatches(true, 2,
    'CCCCd', 3, 2)
    //if first arg is true, ignores case
```

We can format values into a string, using format():

```
//Strings (conversion type 's')
assert String.format('%1$8s', 'hello') == '
hello'
    //width (here, 8) is minimum characters
to be written
assert String.format('%2$6s,%1$2s', 'a',
    'hello') == ' hello, a'
    //we can re-order arguments
assert String.format('%1$2s', 7, 'd') == '
7'
    //we can give any type of input; we can
ignore arguments
assert String.format('%1s,%2s', null,
    'null') == 'null,null'
    //null treated as 'null'
assert String.format('%1$2.4s', 'hello') ==
'hell'
    //precision (here, 4) is maximum
characters to be written

//Characters ('c')
assert String.format('%1$c,%2$3c', 65, 66 as
byte) == 'A, B'
    //convert argument to character; 2nd
value 3 chars wide
```

```
assert String.format('%-3c', 67 as short) ==
'C '
    //left-justified with '-' flag; we
needn't specify parameter number (1$, etc)
assert String.format('%c', 'D' as char) ==
'D'

//Special conversion types:
assert String.format('hello %n world %%') ==
'hello \r\n world %'
    //platform-specific newline; double % to
quote it

//Boolean ('b')
assert String.format('%b, %b, %b, %b, %b,
%b',
```

```
        null, true, false, 0,  
1, new Object()) ==  
    'false, true, false, true, true, true'
```

#### StringBuffers

A StringBuffer is a mutable string. (But from Java 5.0 onwards, we should use a StringBuilder instead, because StringBuffers are normally reserved for multi-threaded processing.)

```
def sb1= new StringBuffer(),  
    sb2= new StringBuffer('Hello'),  
    sb3= new StringBuffer(sb2)  
assert sb1.toString() == '' &&  
    sb2.toString() == 'Hello' &&  
    sb2.toString() == sb3.toString()
```

To find the size and substrings:

```
def sb= new StringBuffer('abcdefg')
assert sb.size() == 7 && sb.length() == 7
//different ways to find size
sb.length= 6 //change size
assert sb.toString() == 'abcdef'
assert sb.reverse().toString() == 'fedcba'
assert sb.toString() == 'fedcba' //reverse()
method reverses order permanently
assert sb.substring(2) == 'dcba' //substring
from index 2
assert sb.substring(2, 5) == 'dcb'
//substring from index 2 to <5
assert sb.subSequence(2, 5) == 'dcb'
//substring from index 2 to <5
assert sb + 'zyx' == 'fedcbazyx'
```

To append to a StringBuffer:

```
def sb1= new StringBuffer()
sb1 << 'abc'
sb1 << 'def' << 'ghi' //can chain two <<
operators
sb1.leftShift('jkl') //equivalent method
name
sb1.append('mno') //alternative method name
sb1.append( ['p', 'q', 'r'] as char[] )
sb1.append( ['r', 's', 't', 'u', 'v'] as
char[], 1, 3 )
assert sb1.toString() ==
'abcdefghijklmnopqrstu'
```

If we append to a String, a StringBuffer is returned:

```
def s= 'foo'
s= s << 'bar'
assert s.class == StringBuffer &&
s.toString() == 'foobar'
```

As with strings, we can subscript a StringBuffer, returning a string:

```
def sb= new StringBuffer('abcdefg')
assert sb[ 3 ] == 'd'
assert sb[ 3 ].class == String
assert sb.getAt( 3 ) == 'd' //equivalent
method name
assert sb.charAt( 3 ) == 'd' //alternative
method name
assert sb[ 3..5 ] == 'def'
assert sb[ 1, 3, 5, 6 ] == 'bdfg'
assert sb[ 1, 3..5 ] == 'bdef'
assert sb[-5..-2] == 'cdef'
sb[ 3..5 ] = 'xy' //use subscripts to update
StringBuffer
assert sb.toString() == 'abcxyg'
sb.putAt( 2..4, 'z' ) //equivalent method
name
assert sb.toString() == 'abzg'
sb.setCharAt(1, 'm' as char) //alternative
method name
assert sb.toString() == 'amzg'
```

We can insert into, replace within, and delete from StringBuffers using methods:

```

def sb= new StringBuffer('hello park')
sb.delete(4, 7)
assert sb.toString() == 'hellark'
sb.deleteCharAt(3)
assert sb.toString() == 'helark'
def ca= new char[6]
sb.getChars(2, 5, ca, 1)
    //for indexes 2 to <5, copy into ca
beginning from index 1
(['\0', 'l', 'a', 'r', '\0', '\0'] as
char[]).
    eachWithIndex{ elt, i-> assert ca[i] ==
elt }

sb.insert(4, 'se')
assert sb.toString() == 'helaserk'
sb.insert(4, new StringBuffer('ct ') )
assert sb.toString() == 'helact serk'
sb.insert(10, ['i', 'c'] as char[] )
assert sb.toString() == 'helact serick'
sb.insert(6, ['m', 'a', 'l', 't'] as char[],
1, 2)
    //insert 2 chars from subscript 1
assert sb.toString() == 'helactal serick'
sb.insert(10, 'snapla', 3, 5) //insert chars
from subscript 3 to <5
assert sb.toString() == 'helactal splerick'
sb.replace(4, 13, 'dor') //replace chars
from subscript 4 to <13
assert sb.toString() == 'heladorrick'

```

We can find the index of substrings:

```
def sb= new StringBuffer('hello elm')
assert sb.indexOf('el') == 1
assert sb.indexOf('el', 3) == 6 //first
occurrence of 'el' from index 3
assert sb.lastIndexOf('el') == 6
assert sb.lastIndexOf('el', 3) == 1 //last
occurrence of 'el' up to index 3
```

Some miscellaneous methods:

```
def s= new String( new
StringBuffer('abcdefg') )
assert s == 'abcdefg'
assert s.contains('def')
assert s.contentEquals('abcdefg')
assert s.contentEquals( new
StringBuffer('abcdefg') )
def s2= s.replace('def', 'xyz')
assert s2 == 'abcxyzg'
```

We can enquire of code points in a String or StringBuffer using methods on them, just as we can with methods on Character:

```
def minLowSurr= Character.MIN_LOW_SURROGATE,
minHighSurr=
Character.MIN_HIGH_SURROGATE

def s1= 'abc'+ minHighSurr + minLowSurr
+'efg'
```

```
assert s1.codePointAt(3) == 0x10000 //if
high surrogate, add on low surrogate
assert s1.codePointAt(4) == minLowSurr //if
low surrogate, use it only
assert s1.codePointAt(5) == 'e' as int
assert s1.codePointBefore(4) == minHighSurr
assert s1.codePointBefore(5) == 0x10000
    //if low surrogate, look back more for
high one, and use both
assert s1.codePointCount(1, 5) == 3
    //number of code points in a substring
with indexes >=1 and <5
assert s1.offsetByCodePoints(1, 3) == 5
    //index from 1 that's offset by 3 code
points
```

```
def sb= new StringBuffer( 'abc'+ minHighSurr
+ minLowSurr +'efg' )
```

```
    //also, for StringBuffers
assert sb.codePointAt(5) == 'e' as int
assert sb.codePointBefore(4) == minHighSurr
assert sb.codePointCount(1, 5) == 3
assert sb.offsetByCodePoints(1, 3) == 5
```

```
sb.appendCodePoint(0x10000)
```

```
assert sb.toString() ==
    'abc'+ minHighSurr + minLowSurr +'efg'+
minHighSurr + minLowSurr
```

We can manipulate the implementation of a StringBuffer:

```
def sb1= new StringBuffer() //default
initial capacity is 16
assert sb1.capacity() == 16

def sb2= new StringBuffer(5) //we can
specify initial capacity
assert sb2.capacity() == 5
sb2<< 'abc'
assert sb2.capacity() == 5 && sb2.size() ==
3
sb2.trimToSize()
assert sb2.capacity() == 3
sb2.ensureCapacity(10)
assert sb2.capacity() == 10

def sb3= new StringBuffer(0) //capacity
approximately doubles when required
def cap= 0, caps=[]
100.times{
    if((sb3<< 'a').capacity() != cap) caps<<
(cap= sb3.capacity())
}
assert caps == [2, 6, 14, 30, 62, 126]
```

We can define string patterns, aka "Regular Expressions" or "Regexes", and see if a String matches it:

```
assert 'abc' ==~ /abc/ //pattern on
righthand side between single-slashes
assert ! ( 'abc' ==~ /ace/ )
assert ! ( 'abc' ==~ /ab/ )

assert 'abc' ==~ /a.c/
    //the . in the pattern matches any
character, except \n (or \r\n on Windows)
assert 'abc'.matches( /a.c/ ) //alternative
method name
assert java.util.regex.Pattern.matches(
/a.c/, 'abc' ) //alternative syntax
assert java.util.regex.Pattern.compile(
/a.c/ ).matcher( 'abc' ).matches()

//alternative syntax

assert '\t\n\f\r' ==~ /\t\n\f\r/
    //some control chars have same notation
as in strings
assert '\t\n\f\r' ==~ /\x09\x0a\x0c\x0D/
    //alternatively use hex codes (leading
zero required to make 2 digits)
assert '\t\n\f\r' ==~ /\011\012\014\015/
    //alternatively use octal codes (leading
zero required)
assert '\b' ==~ /\x08/ && ! ( '\b' ==~ /\b/
)
```

```
// \b has different meaning in regex  
than in string
```

```
assert '\07\013\033' ==~ /\a\v\e/  
    //regex-only notation: bell \a, vertical  
tab \v, escape \e
```

Twelve characters that are special syntax for regexes need to be quoted:

```
assert 'a.c' ==~ /a\.c/ //backslash before .  
to quote it  
assert '.{[()\|^$|?*+' ==~  
/\.\{[\(\)\|\^\$|\|?\*\|+\/  
    //the 12 chars that need quoting  
assert '.{[()\|^$|?*+' ==~  
/\Q.{[()\|^$|?*+\E/  
    //another way to quote text is to  
bracket with \Q and \E  
import java.util.regex.Pattern  
assert Pattern.quote( /.{[()\|^$|?*+\/ ) ==  
/\Q.{[()\|^$|?*+\E/  
    //a special method to quote text in this  
way
```

The chars `\c@`, `\cA`, `\cB`, ..., `\cZ`, `\c[`, `\c]`, `\c^`, and `\c_` map to the special characters 0x0 to 0x1f, except 0x1c:

```
assert "${0x0 as char}" =~ /\c@/  
for(int c= 'A'; int d= 0x1; c <= 'Z'; c++;  
d++){  
    assert "${d as char}" =~ /\c${c as char}/  
}  
assert "${0x1b as char}" =~ /\c[/  
assert "${0x1d as char}" =~ /\c]/  
assert "${0x1e as char}" =~ /\c^/  
assert "${0x1f as char}" =~ /\c_/  

```

We have special pattern syntax for whitespace `\s`, word characters `\w`, digits `\d`, and their complements:

```

assert (0x0..0x7F).findAll{ (it as char) ==~
/\s/ } ==
    ['\t', '\n', '\013', '\f', '\r', '
'].collect{it as int}
assert (0x0..0x7F).findAll{ (it as char) ==~
/\w/ } ==
    [*'0'..'9', *'A'..'Z', '_',
*'a'..'z'].collect{it as int}
assert (0x0..0x7F).findAll{ (it as char) ==~
/\d/ } ==
    ('0'..'9').collect{it as int}

[ [/\w/, /\W/], [/\d/, /\D/], [/\s/, /\S/]
].each{ pair->
    assert (0x0..0x7F).findAll{ (it as char)
==~ pair[0] &&
        (it as char) ==~ pair[1] }.size()
== 0
} // \S means not \s; \W means not \w; \D
means not \d

```

There's certain characters that the dot . doesn't match, except when (?s) is used:

```

assert (0x0..0x7F).findAll{ !( (it as char)
==~ /. / ) } ==
    ['\n' as int, '\r' as int]
    //chars that . doesn't match //also:
0x85, 0x2028, 0x2029
assert 'abc\ndef' ==~ /a.c\ndef/
assert !( 'abc\ndef' ==~ /abc.def/ ) //the .
doesn't match \n

assert (0x0..0x7F).findAll{ !( (it as char)
==~ /( ?s ). / ) } == []
    //when (?s) used, . matches every
character
assert 'abc\r\ndef' ==~ /( ?s )abc..def/ && !(
'abc\r\ndef' ==~ /( ?s )abc.def/ )
    //on Windows, \r\n needs .. for match

assert (0x0..0x7F).findAll{ !( (it as char)
==~ /( ?d ). / ) } == ['\n' as int]
    //only char that . doesn't match for
(?d) flag
assert (0x0..0x7F).findAll{ !( (it as char)
==~ /( ?sd ). / ) } == []
    // (?sd) together same as (?s) alone

```

Some other flags:

```

assert ( 'gOoDbYe' ==~ /(?i)goodbye/ )
    //when (?i) used, case-insensitive
matching for ASCII characters

assert 'an ace' ==~ /(?x) an\ ace #comment
here after hash/
    //quote the space, ignore unquoted
whitespace and comments

```

Some other ways to use flags:

```

assert 'abcDEFG' ==~ /abc(?i)defg/
    //turn on flag halfway thru pattern
assert 'abCDefg' ==~ /ab(?i)cd(?-i)efg/
    //turn flag on, then off again
assert 'abCDEfg' ==~ /ab(?i:cde)fg/
    //turn flag on for only a certain span
of text
assert 'ABcdeFG' ==~ /(?i)ab(?-i:cde)fg/
    //turn flag on, but off for only a
certain span

assert 'abcdefg' ==~ /abc(?ix) d e f g
#comment here/
    //turn more than one flag on together
assert 'abcdefg' ==~ /(?ix) a b c
(?-ix)defg/
    //turn more than one flag off together
assert 'abcdefg' ==~ /(?ix) a b c
(?s-ix)defg/

```

```
//turn some flag(s) on and other flag(s)
off together
```

```
import java.util.regex.Pattern
assert Pattern.compile(/abc.def/,
Pattern.DOTALL).matcher('abc\ndef').matches(
)
```

```
//alternative to (?s)
assert ! Pattern.compile(/abc.def/,
Pattern.UNIX_LINES).
```

```
matcher('abc\ndef').matches()
//alternative to (?d)
```

```
assert Pattern.compile(/goodbye/,
Pattern.CASE_INSENSITIVE).
```

```
matcher('gOoDbYe').matches()
//alternative to (?i)
```

```
assert Pattern.compile(/ an\ ace #comment
here/, Pattern.COMMENTS).
```

```
matcher('an ace').matches()
//alternative to (?x)
```

```
//we can enquire the flags set at the
end-point of a pattern...
```

```
import java.util.regex.Pattern
assert Pattern.compile(/ab(?i)c.def/,
Pattern.DOTALL).flags() ==
```

```
Pattern.DOTALL + Pattern.CASE_INSENSITIVE
assert Pattern.compile(/ab(?i)c.d(?-i)ef/,
Pattern.DOTALL).flags() ==
Pattern.DOTALL
```

```
assert Pattern.compile(/ab(?i:c.d)ef/,
Pattern.DOTALL).flags() ==
Pattern.DOTALL
```

A character class is a set of characters, one of which may be matched. We've already seen the predefined character classes `\s`, `\w`, `\d`, `\S`, `\W`, `\D`. We can also define our own:

```
['bat', 'bet', 'bit', 'bot', 'but'].each{
assert it ==~ /b[aeiou]t/ }
  // [aeiou] matches one of a,e,i,o,u
assert ! ('bnt' ==~ /b[aeiou]t/)

['bat', 'bet', 'bit', 'bot', 'but'].each{
assert ! (it ==~ /b[^aeiou]t/) }
  // [^aeiou] matches anything except
a,e,i,o,u...
['bbt', 'bxt', 'b%t', 'b)t', 'b*t',
'b\n t'].each{ assert it ==~ /b[^aeiou]t/ }
  // ...even newlines

assert 'b' ==~ /[abbbc]/ //duplicate chars
in character class have no effect
assert '&' ==~ /[a&]/ &&
  !('&' ==~ /[a&&z]/) &&
  '&' ==~ /[a&&&]/ &&
  !('&' ==~ /[a&&]/) &&
  '&' ==~ /[a&\&]/ //all legal syntax

[ /[a-j]/: ['a'..'j'],
  //we can specify a range of characters
inside a class using hyphen -
```

```

/[_a-zA-Z]/: ['A'..'Z', '_', 'a'..'z'],
  //we can have many ranges mixed with
single characters
/[_a-zA-Z]/: ['A'..'Z', '_',
*'a'..'z'],
  //same effect as [_a-zA-Z]
/[a-m&&g-z]/: ['g'..'m'],
  //&& is intersection operator
/[a-zA-Z&&[^bc]]/: ['a', 'd'..'z'],
  //^ means 'not' everything in the
character class
/[a-zA-Z&&[^m-p]]/: ['a'..'l', 'q'..'z'],
  //&& with ^ works like subtraction
/[^d\s]/: [0x0..0x7F].collect{ it as
char } - ['\t'..'r', ' ', '0'..'9'],
  //not digit AND not whitespace
/[\D\S]/: [0x0..0x7F].collect{ it as char
},
  //not equal to above, but means: not
digit OR not whitespace
].each{ regex, validVals->
  assert (0x0..0x7F).findAll{ (it as char)

```

```
==~ regex } ==  
    validVals.collect{ it as int }  
}
```

The only meta-characters inside a character class are \, [, ^ (in the first position), ] (not in the first position or after the ^), - (not in the first position, after the ^, or before the ]), and &&. Quote them with a / to get the literal character. The other usual meta-characters are normal characters inside a character class, and do not need to be quoted with a backslash, though can be. Character class precedences are, from highest: literal escapes (eg \s), grouping (eg [abc]), ranges (eg a-g), unions (eg [abc][xyz]), then intersections ([a-z&&[gjpqy]]).

We can use the alternation operator | to give some options:

```
['abc', 'def', 'xyz'].each{ assert it ==~  
/abc|def|xyz/ }  
['abcz', 'aijz', 'axyz'].each{ assert it ==~  
/a(bc|ij|xy)z/ }  
    //we delimit the alternation with  
parentheses  
  
//when using longhand syntax, we can see  
what option was matched, using groups,  
//which we'll meet soon:  
def m= java.util.regex.Pattern.compile(  
/a(bc|ij|xy)z/ ).matcher( 'abcz' )  
m.matches()  
assert m.group(1) == 'bc' //whatever was  
matched between the parens
```

We use ? to indicate optional character/s:

```
['0 days', '1 day', '2 days'].each{ assert
it ==~ /. days?/ }
['Mon', 'Monday'].each{ assert it ==~
/Mon(day)?/ }
```

Use {n} to match a character exactly n times:

```
assert 'aaab' ==~ /a{3}b/
assert 'abcabc' ==~ /(abc){2}/ // {n} can
apply to a multi-character sequence
['ab', 'ba', 'bb', 'aa'].each{ it ==~
/[ab]{2}/ }
// {n} can apply to a character class
['abab', '%&@b'].each{ assert it ==~ /.{3}b/
}
```

We can match a character a variable number of times. Use the \* operator to match any number of a character:

```
['aaab', 'aab', 'ab', 'b'].each{ assert it
==~ /a*b/ }
//even zero occurrences of the character
is matched
['abcabc', 'abc', ''].each{ assert it ==~
/(abc)*/ }
// * can apply to a multi-character
sequence
['abbacb', 'acaba', 'cbbbac', 'c', ''].each{
assert it ==~ /[abc]*/ }
// * can apply to a character class
```

```
['aaab', 'b', 'abab'].each{ assert it =~  
/.*b/ }  
    // * is greedy: in 'abab' .* matches  
    'aba'
```

//Use + to match at least one occurrence of a character:

```
['aaab', 'aab', 'ab'].each{ assert it =~  
/a+b/ }  
assert !( 'b' =~ /a+b/ ) //at least one 'a'  
is required  
assert 'abcabcxz' =~ /(abc)+[xyz]+/  
    // + can apply to character class or  
multi-character sequence
```

//Other variable-length repetition operators:

```
assert 'aaaab' =~ /a{3,}b/ // {n,} matches  
at least n characters  
assert 'aaaab' =~ /a{3,5}b/ // {n1,n2}  
matches between n1 and n2 characters  
assert 'abaxyzxyz' =~ /[ab]{2,}(xyz){2,4}/
```

```
//these also can apply to  
multi-character sequences or character  
classes
```

By using longhand syntax, we see that \* operator is greedy, repeating the preceding token as often as possible, returning the leftmost longest match:

```
def m= java.util.regex.Pattern.compile(  
/(.*)/(.*)/ ).matcher( 'one,two,three' )  
m.matches()  
assert m.group(1) == 'one,two' //what was  
matched between the first parens  
assert m.group(2) == 'three' //what was  
matched between the second parens  
  
assert m.hasGroup() //misc method to check  
whether the pattern has groups  
assert m.groupCount() == 2 //misc method to  
count them
```

Anything between parentheses is a capturing group, whose matched values can be accessed later:

```
//we can access matched values in groups  
outside the pattern using  
//longhand syntax...  
def m= java.util.regex.Pattern.compile(  
/(a*)(b*)/ ).matcher( 'aaabb' )  
m.matches()  
assert m.group(1) == 'aaa' && m.start(1) ==  
0 && m.end(1) == 3  
assert m.group(2) == 'bb' && m.start(2) == 3
```

```
&& m.end(2) == 5
assert m.group(0) == 'aaabb' //group(0) is
the entire string
assert m.group() == 'aaabb' && m.start() ==
0 && m.end() == 5
    //parameters default to 0
```

```
//...or outside the pattern using indexing
(don't forget the first [0] index)...
```

```
m= java.util.regex.Pattern.compile(
/(a*)(b*)/ ).matcher( 'aaabb' )
m.matches()
assert m[0][0] == 'aaabb' //the entire
string
assert m[0][1] == 'aaa' && m.start(1) == 0
&& m.end(1) == 3
assert m[0][2] == 'bb' && m.start(2) == 3 &&
m.end(2) == 5
```

```
//...or within the pattern using \n
notation:
```

```
assert 'aaabb,aaa,bb' ==~ /(a*)(b*),\1,\2/
    // \1 is the first group matched, \2 the
second matched
```

```
assert 'abbcc,abb,bb,cc' ==~
/(a(b*)) (c*),\1,\2,\3/
    //groups numbered by sequence of their
opening parens from left to right
assert 'abcddd,ab,ddd' ==~
/(a(?:b)) (?>c) (d*),\1,\2/
```

**//groups beginning with ?: or ?> aren't  
numbered**

```
assert 'aba,a,b' ==~ /(a(b)?)+,\1,\2/
```

**//second match for \1 has no match for  
\2, so \2 keeps value from first match**

```
assert 'abc,bc' ==~ /a(bc)?,\1/
```

```
assert !( 'a,' ==~ /a(bc)?,\1/ )
```

**//referencing \1 causes entire match to  
fail if it hasn't already matched**

```
assert !( 'a' =~ /([abc]\1)/ )  
    //referencing a group within itself  
causes entire match to fail
```

\1 through \9 in patterns are always interpreted as group references, and a backslash-escaped number greater than 9 is treated as a group reference if at least that many groups exist at that point in the string pattern. Otherwise digits are dropped until either the number is smaller or equal to the existing number of groups or it is one digit. Grouping parentheses and group references cannot be used inside character classes.

Some miscellaneous methods:

```
def m= ( ~/ (a*) |bc/ ).matcher( 'bc' )  
//another longhand syntax  
m.matches()  
assert m.group(1) == null && m.start(1) ==  
-1 && m.end(1) == -1  
                                //if match  
successful but group didn't match anything  
  
def p= java.util.regex.Pattern.compile(  
/ab*c/ )  
assert p.pattern() == /ab*c/ //retrieve the  
definition from a compiled pattern
```

#### Finding Patterns in Strings

As well as matching an entire string to a pattern, we can also find a pattern within a string using =~ syntax:

```
assert 'abcdefg' =~ /cde/ //is 'cde' within
'abcdefg'?
assert ! ( 'abcdefg' =~ /ace/ )
assert java.util.regex.Pattern.compile(
/cde/ ).matcher( 'abcdefg' ).find()

//alternative syntax

assert 'xxx z9g\t\nxxx' =~ /\s\w\d.\t\n/
    //special characters work the same as
with ==~ matching
assert ( 'xxxgOoDbYexxx' =~ /( ?i)goodbye/ )
    //flags also work the same as with ==~
assert 'xxbatxxx' =~ /b[aeiou]t/
    //character classes also work the same
as with ==~
```

There can be more than one occurrence of the pattern:

```

def s= 'horse house'
assert s =~ /ho.se/ //to check for the first
occurrence only
def m= (s =~ /ho.se/)
assert m.size() == 2 && m[0] == 'horse' &&
m[1] == 'house'
    //to retrieve all occurrences

def l= []
s.eachMatch( /ho.se/ ){ l << it[0] }
//alternative syntax, be sure to use it[0]
assert l == ['horse', 'house']
def l2= []
s.eachMatch( /abc/ ){ l2 << it[0] } //no
matches
assert l2 == []
def l3= []
s.eachMatch( /hor./ ){ l3 << it[0] } //one
match only
assert l3 == ['hors']

```

Some longhand syntax, with various methods:

```

import java.util.regex.Pattern
def s= 'hooose horse house'
def m= Pattern.compile(/ho.se/).matcher(s)
assert m.find() && s[m.start()..<m.end()] ==
'hoose'
assert m.find() && s[m.start()..<m.end()] ==
'horse'
assert m.find() && s[m.start()..<m.end()] ==
'house'
assert ! m.find()
assert m.reset() && s[m.start()..<m.end()]
== 'hoose'
    //use reset() to find from beginning
assert m.find() && s[m.start()..<m.end()] ==
'horse'
assert m.find(1) && s[m.start()..<m.end()]
== 'horse'
    //giving a parameter to find() starts
finding from that index
m.setIndex(1)
    //alternatively, calling setIndex()
resets from that index, without finding
    //until find() called

assert m.find() && s[m.start()..<m.end()] ==
'horse'

```

We can group when finding with =~ just as we do when matching with ==~:

```

def m= ( 'mistlemuscle' =~ /m(.)s(.)le/ )

```

```

assert m.size() == 2
assert m.count == 2 //alternative to size()
assert m[0] == ['mistle', 'i', 't']
assert m[0].size() == 3 && m[0][0] ==
'mistle' &&
    m[0][1] == 'i' && m[0][2] == 't'
assert m[1] == ['muscle', 'u', 'c']
assert m[1].size() == 3 && m[1][0] ==
'muscle' &&
    m[1][1] == 'u' && m[1][2] == 'c'

//using the eachMatch() method...
def l= []
'mistlemuscle'.eachMatch( /m(.)s(.)le/ ){ l
<< it }
assert l*.toList() == [['mistle', 'i', 't'],
['muscle', 'u', 'c']]
def l2= []
'mistle'.eachMatch( /m(.)s(.)le/ ){ l2 << it
}
assert l2*.toList() == [['mistle', 'i',
't']]
def l3= []
'practical'.eachMatch( /m(.)s(.)le/ ){ l3 <<
it }
assert l3*.toList() == []

//using longhand notation...
import java.util.regex.Pattern
m= Pattern.compile( /(a+)(b+)/ ).matcher(
'aaabbcccaabbb' )

```

```

m.find()
assert m.group(1) == 'aaa' && m.start(1) ==
0 && m.end(1) == 3 &&
    m.group(2) == 'bb' && m.start(2) == 3
&& m.end(2) == 5 &&
    m.group() == 'aaabb' && m.start() ==
0 && m.end() == 5
m.find()
assert m.group(1) == 'aa' && m.start(1) == 8
&& m.end(1) == 10 &&
    m.group(2) == 'bbb' && m.start(2) ==
10 && m.end(2) == 13 &&
    m.group() == 'aabbb' && m.start() ==
8 && m.end() == 13

```

Calling `collect()` and `each()` require some special tricks to work:

```

def m= ('redeem coffee' =~ /ee/)
assert m.collect{it} == ['ee', 'ee']
    //when calling collect() on a pattern
with no groups...
assert m.collect{it} == []
    //...we must call reset() if we want to
access the found matches again
m.reset()
assert m.collect{it} == ['ee', 'ee']

def l= [] //ditto for each()
m.each{ l << it }
assert l == []
m.reset()

```

```

l= []
m.each{ l << it }
assert l == ['ee', 'ee']

l= [] //ditto for eachWithIndex
m.eachWithIndex{it, i-> l << it+i }
assert l == []
m.reset()
l= []
m.eachWithIndex{it, i-> l << it+i }
assert l == ['ee0', 'ee1']

m= ('play the game\nfollow the rules' =~
/(?m)^(.*?) the (.*?)$/)
    //for a pattern with groups...
l= []
m.each{g0, g1, g2-> l << [g0, g1, g2] }
    //...we must pass the groups separately
to the closure of each()
assert l == [['play the game', 'play',
'game'],
              ['follow the rules', 'follow',
'rules']]

m= ( 'mistlemuscle' =~ /m(.)s(.)le/ )
assert m[1] == ['muscle', 'u', 'c']
assert m.group(0) == 'muscle' && m.group(1)

```

```
== 'u' && m.group(2) == 'c'  
    //only call group() after using  
    subscripting first
```

Aggregate functions we can use are:

```
assert ('tone, true, tame, tape, take, tile,  
time' =~ /t..e/).  
    findAll{ it[1] == 'a' } == ['tame',  
    'tape', 'take']  
assert ('tone, true, tame, tape, take, tile,  
time' =~ /t..e/).  
    find{ it[1] == 'a' } == 'tame'  
assert ('tone, true, tame, tape, take, tile,  
time' =~ /t..e/).  
    findIndexOf{ it[1] == 'a' } == 2 //index  
of 'tame'  
assert ('tone, true, tame, tape, take, tile,  
time' =~ /t..e/).  
    any{ it[1] == 'a' }  
assert ! ('tone, true, tame, tape, take,  
tile, time' =~ /t..e/).  
    every{ it[1] == 'a' }
```

The sequence of text joined by operators such as `| ? * + {}` has no effect on the success of the `==~` matcher, but does affect what's found with the `==~` finder. The first choice of the `|` is found first, and backtracking to the second choice is only tried if necessary. The choice of the `?` is tried first, and backtracking to ignore the choice only tried if necessary. As much as possible of the `* + {}` is found first, and backtracking to find less text only tried if necessary.

```

assert ('abcdefg' =~ /bcd|bcdef/)[0] ==
'bcd'
assert ('abcdefg' =~ /bcdef|bcd/)[0] ==
'bcdef'
    //first choice always tried first

assert ('Friday 13th' =~ /Fri(day)?/)[0][0]
== 'Friday'

assert ('Say "hello" and "goodbye" to the
world!' =~ /".*"/)[0] ==
    '"hello" and "goodbye"'
l= []
'Say "hello" and "goodbye" to the
world!'.eachMatch( /"[^"]*" / ){ l << it }
    //use NOT DOUBLE-QUOTES instead of ANY
CHARACTER
assert l*.toList() == [ '"hello"',
    '"goodbye"' ]

```

Because the ? and \* operators can match nothing, they may not always be intuitive to understand:

```

def m= ('grgggr' =~ /g?/)
def l= []
for( int i in 0..<(m.size() as int) ) l <<
m[i]
assert l == ['g', '', 'g', 'g', 'g', '', '']
// ? also matches the empty space before
each 'r', and the end of string

m= ('grgggr' =~ /g*/)
l= []
for( int i in 0..<(m.size() as int) ) l <<
m[i]
assert l == ['g', '', 'ggg', '', '']
// * also matches the empty space before
each 'r', and the end of string

m= ('grgggr' =~ /g+/)
l= []
for( int i in 0..<(m.size() as int) ) l <<
m[i]
assert l == ['g', 'ggg'] // + repetition is
the most intuitive to use

```

By putting a ? after the operators ? \* + {}, we can make them "lazy" instead of "greedy", that is, as little as possible is found first, and backtracking to find MORE text is tried if necessary:

```

assert ('Friday 13th' =~ /Fri(day)??/)[0][0]
== 'Fri' //instead of 'Friday'

def l= []
'Say "hello" and "goodbye" to the
world!'.eachMatch( /".*?"/ ){ l << it }
assert l*.toList() == [ '"hello"',
[ '"goodbye"' ] ]

```

We've seen some longhand methods such as 'find', 'matches', 'start', and 'end'. There's many more such methods:

```

def s= 'a quick quick dog'
def m= (s =~ /a.*k/)
  //starts at the beginning, but doesn't
try to match the entire string
assert m.lookingAt() &&
s[m.start()..<m.end()] == 'a quick quick'

//replaceFirst...
assert (s =~ /quick/).replaceFirst('fast')
== 'a fast quick dog'
assert (s =~ /qu(ick)/).replaceFirst('kw$1')
== 'a kwick quick dog'
  //can reference groups in pattern using
$
assert (s =~
/qu(ick)/).replaceFirst('kw\\$1') == 'a kw$1
quick dog'
  //include literal $ by writing \$,
escaping \ as \\

```

```
//utility method to create a literal
replacement String for the given String...
import java.util.regex.Matcher
assert Matcher.quoteReplacement( 'kw$1' ) ==
'kw\\$1'
assert (s =~ /qu(ick)/).replaceFirst(
Matcher.quoteReplacement( 'kw$1' ) ) ==
'a kw$1 quick dog'
```

```
//we can mix GStrings and replacement group
refs by mixing single-quoted and
//double-quoted strings...
def ice= 'ice cream'
assert ('some malting beer' =~ /a(lting
).*/).replaceFirst('e$1' + "$ice") ==
'some melting ice cream'
```

```
//replaceAll...
assert (s =~ /quick/).replaceAll('fast') ==
'a fast fast dog'
s= 'a quickly quacking duck'
assert (s =~ /qu(.)ck/).replaceAll('kw$1ck')
== 'a kwickly kwacking duck'
```

```
//another shorthand...
assert 'a quick quick
dog'.replaceFirst(/qu(ick)/, 'kw\\$1') ==
'a kw$1 quick dog'
assert 'a quickly quacking
duck'.replaceAll(/qu(.)ck/, 'kw$1ck') ==
```

```
'a kwickly kwacking duck'
```

```
//'appendReplacement' and 'appendTail'
```

```
should be used together for more
```

```
//complex replacements...
```

```
m= 'one banana two havana three matana four'
```

```
=~ /(.a.)ana/
```

```
def i=0, sb= new StringBuffer()
```

```
while( m.find() ) m.appendReplacement(sb,
```

```
'$1a' + 'na'*i++)
```

```
m.appendTail(sb)  
assert sb.toString() == 'one bana two havana  
three matanana four'
```

Similarly to back-references in patterns, \$1 through \$9 in replacement strings are always interpreted as group references, and a dollar-escaped number greater than 9 is treated as a group reference if at least that many groups exist in the string pattern. Otherwise digits are dropped until either the number is smaller or equal to the existing number of groups or it is one digit.

We've already seen the greedy and lazy operators. There's also possessive operators, which act like greedy operators, except they never backtrack. Whereas choosing greedy or lazy operators affects the efficiency of a match, they don't affect the outcome. However, possessive operators can affect the outcome of a match:

```

//the greedy * operator, with backwards
backtracking...
def m= ( ~/(*),(.*)/ ).matcher(
'one,two,three' )
assert m.matches() && m.group(1) ==
'one,two' && m.group(2) == 'three'

//the lazy *? operator, with forwards
backtracking...
m= ( ~/(*?),(.*)/ ).matcher(
'one,two,three' )
assert m.matches() && m.group(1) == 'one' &&
m.group(2) == 'two,three'

//the possessive *+ operator, with no
backtracking at all, even when doing so
//would cause a match...
assert ! ( ~/(*+),(.*)/ ).matcher(
'one,two,three' ).matches()

//we can qualify other operators with
possessiveness, such as ++, ?+, {m,n}+...
m= ( ~/([abc,]*+),(.*)/ ).matcher(
'abba,and,beegees' )
assert ! m.matches()
    //greedily matches 'abba,a', but doesn't
backtrack to 'abba'

```

Atomic grouping, a more general form of possessiveness, enables everything in the atom group to be considered as one token. No backtracking occurs within the group, only outside of it:

```
assert ! ( 'abbbc' ==~ /a(?>b*)bc/ )
      //after 'bbb' matched, no backtracking
      to 'bb' within atomic group
```

Atomic grouping and possessiveness are handy with nested repetition, allowing much faster match failures.

#### Finding Positions in Strings

We can use ^ and \$ to match the beginning and end of each line using flag m:

```

def s= 'an apple\nthe lime\na banana'
assert ! (s =~ /^a.{7}$/)
    //normally, ^ matches the beginning of
the entire input,
    //and $ matches its end

def m= (s =~ /( ?m)^a.{7}$/)
    //in multi-line mode, ^ matches the
beginning of each line,
    //and $ matches each line's end

assert m.size() == 2 && m[0] == 'an apple'
&& m[1] == 'a banana'
assert m.toString() ==

'java.util.regex.Matcher[pattern=( ?m)^a.{7}$
region=0,26 lastmatch=a banana]'
    //some technical info

assert ((s+' \n') =~ /( ?m)^a.{7}$/) // $
ignores any \n at the end of the string

import java.util.regex.Pattern
m= Pattern.compile(/^a.{7}$/,
Pattern.MULTILINE).matcher(s)
    //alternative to ( ?m) in longhand syntax
assert m.find() && s[m.start()..<m.end()] ==
'an apple'
assert m.find() && s[m.start()..<m.end()] ==
'a banana'
assert ! m.find()

```

At the end of strings with \n at the end, \$ matches twice:

```
m= ( 'nine\nlives' =~ /$/ )
assert m.find() && m.start() == 10 &&
m.end() == 10
    // $ matches at end of string once only
assert ! m.find()
```

```
m= ( 'nine\nlives\n' =~ /$/ )
assert m.find() && m.start() == 10 &&
m.end() == 10
    // $ matches just before \n ...
assert m.find() && m.start() == 11 &&
m.end() == 11
    //...and again, $ matches after the \n
assert ! m.find()
```

```
m= ( 'nine\nlives\n' =~ /(?m)$/ )
assert m.find() && m.start() == 4 && m.end()
== 4
    //in multiline mode, $ matches at end of
each line
assert m.find() && m.start() == 10 &&
m.end() == 10
assert m.find() && m.start() == 11 &&
m.end() == 11
    // $ also always matches after the \n in
multiline mode
assert ! m.find()
```

```
m= ( 'nine\nlives\n' =~ /^/ )
    // ^ matches at beginning of string once
only,
    //even if there's an \n at the end
assert m.find() && m.start() == 0 && m.end()
== 0
assert ! m.find()
```

```
m= ( 'nine\nlives\n' =~ /(m)^/ )
assert m.find() && m.start() == 0 && m.end()
== 0
assert m.find() && m.start() == 5 && m.end()
== 5
    //in multiline mode, ^ matches at
beginning of each line
```

```
assert ! m.find()
    // ^ also never matches after the \n in
multiline mode
```

We can use `\A` `\Z` and `\z` to match the beginning and end of input, even in multiline mode:

```
def s1= 'an apple\na banana'
assert (s1 =~ /\A.{8}\n.{8}\Z/)
    // \A always matches the beginning of
the entire input, and \Z its end
assert (s1 =~ /\A.{8}\n.{8}\z/) // \z also
matches its end

assert (s1 =~ /(?!m)\A.{8}\n.{8}\Z/)
    // ?m flag has no effect on meaning of
\A \Z and \z

def s2= s1 + '\n'
assert (s2 =~ /(?!m)\A.{8}\n.{8}\Z/)
    // \Z ignores an extra \n when matching
the end of input...
assert ! (s2 =~ /(?!m)\A.{8}\n.{8}\z/) //
...but \z is fussy
```

We can match at word boundaries:

```

// \b matches either the preceding or
following character, but not both, is
// a word (matched by \w)
(0x20..0x7F).each{it1->
  (0x20..0x7F).each{it2->
    def s= "${it1 as char}${it2 as char}"
    if( s ==~ /\. \b./ ) assert (s[0] ==~
/\w/) ^ (s[1] ==~ /\w/)

// ^ means xor (exclusive or)
} }

// \B matches where \b doesn't
assert (0x0..0x7F).findAll{ (it as char) ==~
/\b/ && (it as char) ==~ /\B/ }.
size() == 0

```

We can look behind or ahead of a position, ie, find a position based on text that precedes/follows it, but without matching that text itself. We can only use fixed-length strings when looking behind, ie, literal text, character classes, finite repetition ( {length} and ? ), and alternation where each string in it is also of fixed length, because the length of the match must be able to be predetermined:

```

//use (?=) to find the position just in
front of all 'qu'...
assert 'the queen quietly
quacked'.replaceAll( /(?=qu)/, 'we' ) ==
'the wequeen wequietly wequacked'

//use (!) to find all 'c' not followed by
'a'...
assert 'clever cats can count

```

```
mice'.replaceAll( /c(?!a)/, 'k' ) ==  
    'klever cats can kount mike'
```

```
//use (?<=) to find all words ending in  
'-gry'...
```

```
assert 'The angry, hungry boy gried out.'  
    .replaceAll( /\b\w+?(?<=gry)\b/,  
'naughty' ) ==  
    'The naughty, naughty boy gried out.'
```

```
//use (?<!) to find 3-letter words not  
ending with 'e'...
```

```
assert 'The spy saw seven  
spuds.'.replaceAll( /\b\w{3}(?<!e)\b/, 'hid'  
) ==  
    'The hid hid seven spuds.'
```

```
//lookaheads and lookbehinds can contain  
capturing groups...
```

```
assert 'the landlord dared band led not'.
```

```
replaceAll( /\b\w{4,}(?<=(\w{3})d)\b/,
'$1' ) ==
'the lor are ban led not'
```

Matching positions in a string is useful for splitting the string, and for inserting text:

```
assert 'The leaky
cauldron.'.split(/\b/).toList() ==
['', 'The', ' ', 'leaky', ' ', 'cauldron',
'.']
//note that an empty string is prepended
assert 'Hi, my, bye.'.split( /\b(?\w)/
).toList() ==
['', 'Hi, ', 'my, ', 'bye.']
assert 'The leaky
cauldron.'.replaceAll(/\b/, '*') ==
'*The* *leaky* *cauldron*.'
//note that text inserted at beginning
but not at end
```

We can split a string in many ways:

```

def s= 'hi,my,spy,tie,bye,, '
assert s.split( /,/ ).toList() == ['hi',
'my', 'spy', 'tie', 'bye']
assert s.split( /,/, 1 ).toList() ==
['hi,my,spy,tie,bye,, ']
    //extra argument gives max number of
splits
assert s.split( /,/, 2 ).toList() == ['hi',
'my,spy,tie,bye,, ']
assert s.split( /,/, 3 ).toList() == ['hi',
'my', 'spy,tie,bye,, ']
assert s.split( /,/, 0 ).toList() == ['hi',
'my', 'spy', 'tie', 'bye']
    //any number of splits; same as no arg
assert s.split( /,/, -1 ).toList() == ['hi',
'my', 'spy', 'tie', 'bye', '', '']
    //a negative arg doesn't remove trailing
empty strings

assert ( ~/,/ ).split(s).toList() == ['hi',
'my', 'spy', 'tie', 'bye']
    //alternative syntax
assert ( ~/,/ ).split(s, 2).toList() ==
['hi', 'my,spy,tie,bye,, ']

```

#### Restricting a String to a Region for a Pattern

We can set the limit of the part of the input string that will be searched to find a match:

```

import java.util.regex.Pattern

```

```
def m= Pattern.compile( /abc+/ ).matcher(
  'aaabc' )
assert m.find()
m.region(1, 4) //restrict string 'aaabc' to
a region within, ie, 'aab'
assert ! m.find()
assert m.regionStart() == 1 && m.regionEnd()
== 4
assert ! m.region(1, 4).find() //alternative
syntax
```

```
//we can make a region's boundaries
transparent to lookaround and boundary
//matching constructs...
m= Pattern.compile( /abc\b/ ).matcher(
  'aaabcdef' )
m.region(1, 5)
assert m.find() //doesn't consider whether
there's a word boundary (\b) after
                //'aabc' in full string
assert ! m.hasTransparentBounds()
m.region(1, 5)
m.useTransparentBounds(true)
assert ! m.find() //doesn't find anything
because the \b doesn't match
assert m.hasTransparentBounds()
assert ! m.region(1,
5).useTransparentBounds(true).find()
//alternative syntax
```

```
//we can decide whether to match anchors
```

```
such as ^ and $ at the boundaries of
//the region...
m= Pattern.compile( /^abc$/ ).matcher(
'aaabcdef' )
m.region(2, 5)
assert m.find()
assert m.hasAnchoringBounds() //match such
anchors by default
m.region(2, 5)
m.useAnchoringBounds(false)
assert ! m.find() //the ^ and $ no longer
match
assert ! m.region(2,
5).useAnchoringBounds(false).find()
//alternative syntax
```

## JN2015-Files

To see the OS-dependent characters used for formatting filenames (here, when running on Windows):

```
assert File.separator == '\\\' &&
File.separatorChar == '\\\' as char
    //used for formatting file names
assert File.pathSeparator == ';' &&
File.pathSeparatorChar == ';' as char
```

Instances of File are immutable representations of objects in the file system, that may or may not exist. To see different formats of a filename (here, when running within D:\Groovy\Scripts directory):

```
def f= new File('File.txt') //relative file
name
assert f.name == 'File.txt'
```

```
assert ! f.isAbsolute()
assert f.path == 'File.txt'
assert f.parent == null
assert f.absolutePath ==
'D:\\Groovy\\Scripts\\File.txt' //returns a
string
assert f.absoluteFile.toString() ==
'D:\\Groovy\\Scripts\\File.txt'
    //returns a File object instead of
string
assert f.canonicalPath ==
'D:\\Groovy\\Scripts\\File.txt'
assert f.canonicalFile.toString() ==
'D:\\Groovy\\Scripts\\File.txt'
    //returns a File object instead of
string
assert f.toURI().toString() ==
'file:/D:/Groovy/Scripts/File.txt'
    //toURI() returns a URI object

f= new File('D:/Groovy/Scripts/File.txt')
//absolute file name
assert f.name == 'File.txt'
assert f.isAbsolute()
assert f.path ==
'D:\\Groovy\\Scripts\\File.txt'
assert f.parent == 'D:\\Groovy\\Scripts'
assert f.parentFile.toString() ==
'D:\\Groovy\\Scripts'
    //returns a File object instead of
string
```

```
assert f.absolutePath ==
'D:\\Groovy\\Scripts\\File.txt'
assert f.canonicalPath ==
'D:\\Groovy\\Scripts\\File.txt'

f= new File('../File.txt')
assert f.name == 'File.txt'
assert ! f.isAbsolute()
assert f.path == '..\\File.txt'
assert f.parent == '..'
assert f.absolutePath ==
'D:\\Groovy\\Scripts\\..\\File.txt'
assert f.canonicalPath ==
'D:\\Groovy\\File.txt'
```

```
f= new File('') //current directory
assert f.name == ''
assert ! f.isAbsolute()
assert f.path == ''
assert f.parent == null
assert f.absolutePath ==
'D:\\Groovy\\Scripts'
assert f.canonicalPath ==
'D:\\Groovy\\Scripts'
```

```
assert new File('File.txt') == new
File('File.txt')
    //compares two filenames' lexical names
assert new File('File.txt').compareTo(new
File('File.txt')) == 0
    //equivalent method name
```

```
assert new File('File.txt') != new
```

```
File('../Scripts/File.txt')
    //lexical names different (although
files are the same)
```

None of the above example's files were created. Files are only created by some event:

```
def f1= new File('File1.txt')
f1 << 'abcdefg'
    //file created by writing to it; file
appended to if it already exists
assert f1.length() == 7 && f1.size() == 7
assert f1.isFile() && ! f1.isDirectory() &&
! f1.isHidden()

def f2= new File('D:/Groovy/Scripts',
'File2.txt')
    //we can optionally supply the parent,
either as a string...
f2= new File(new File('D:/Groovy/Scripts'),
'File2.txt')
    //...or as a File object
assert ! f2.exists()
f2.createNewFile() //if it doesn't already
exist
assert f2.exists()

def d1= new File('Directory1')
d1.mkdir() //make directory, if it doesn't
already exist
def d2= new File('Directory2/SubDir1')
d2.mkdirs()
```

```
//make directory, including necessary
but nonexistent parent directories

println f1.getFreeSpace()
    //the number of unallocated bytes in the
partition this abstract file is in
println f1.getUsableSpace()
    //the number of bytes available to this
virtual machine in the partition
    //this abstract file is in
println f1.getTotalSpace() //the size of the
partition this abstract file is in

//We can set file permissions:

assert f2.setWritable(true, false) &&
f2.canWrite()
    //set writable permission for every user
assert f2.setWritable(true) && f2.canWrite()
    //set writable permission on file for
owner only
assert f2.setWritable(false, false) && !
f2.canWrite()
    //unset writable permission for every
user
assert f2.setWritable(false) && !
f2.canWrite()
    //unset writable permission on file for
owner only
f2.writable= true //property format for
```

```
owner only
assert f2.canWrite()

assert f2.setReadOnly() && ! f2.canWrite()
assert f2.setExecutable(true, false) &&
f2.canExecute()
    //set executable permission for every
user
assert f2.setExecutable(true) &&
f2.canExecute()
    //set executable permission on file for
owner only
f2.executable= true //property format for
owner only
assert f2.canExecute()
assert ! f2.setExecutable(false)
    //returns false because command
unsuccessful: can't make file
    //nonexecutable on Windows, though can
on other systems

assert f2.setReadable(true, false) &&
f2.canRead()
    //set readable permission for every user
assert f2.setReadable(true) && f2.canRead()
    //set readable permission on file for
owner only
f2.readable= true //property format for
owner only
assert f2.canRead()
assert ! f2.setReadable(false)
```

```
//can't make file nonreadable on Windows

//We can retrieve a list of files from a
directory:

assert new
File('D:/Groovy/Scripts').list().toList() ==
    ['Script.bat', 'File1.txt', 'File2.txt',
'Directory1', 'Directory2']
    //list() returns an array of strings

assert new
File('Directory2').list().toList() ==
['SubDir1']
assert new File('').list() == null
    //list() returns null if directory not
explicitly specified

assert new File('D:/Groovy/Scripts').list(
    [accept:{d, f-> f ==~ /.?*?1.* / } ] as
FilenameFilter
    ).toList() == ['File1.txt', 'Directory1']
    //filter taking dir (File) and file
(String) arguments, returns boolean

assert new File('D:/Groovy/Scripts').list(
    {d, f-> f ==~ /.?*?1.* / } as
FilenameFilter
    ).toList() == ['File1.txt', 'Directory1']
//shorter syntax
```

```
assert new
File('D:/Groovy/Scripts').listFiles().toList
()* .name ==
    ['Script.bat', 'File1.txt', 'File2.txt',
    'Directory1', 'Directory2']
    //listFiles() returns array of File
objects
```

```
assert new
File('Directory2').listFiles().toList()* .toS
tring() ==
    ['Directory2\\SubDir1']
```

```
assert new
File('D:/Groovy/Scripts').listFiles(
    {dir, file-> file ==~ /.*?.txt/ } as
FilenameFilter
    ).toList()* .name == [ 'File1.txt',
    'File2.txt' ]
```

```
assert new
File('D:/Groovy/Scripts').listFiles(
    [accept:{file-> file ==~ /.*?.txt/ }]
as FileFilter
    ).toList()* .name == [ 'File1.txt',
    'File2.txt' ]
    //use a filter taking one argument only,
returning boolean
```

```
//Renaming and deleting files:
```

```
f2.renameTo( new File('RenamedFile2.txt') )
assert f2.name == 'File2.txt' //because File
object is immutable
assert new File('RenamedFile2.txt').exists()
```

```
[new File('RenamedFile2.txt'), new
File('Directory1'), new File('Directory2')].
  each{ it.delete() } //delete files
```

```
assert ! new
```

```
File('RenamedFile2.txt').exists()
```

```
assert ! new File('Directory1').exists()
```

```
assert new File('Directory2').exists()
```

```
    //because each sub-directory must be
deleted separately
```

```
assert new
```

```
File('Directory2/SubDir1').delete()
```

```
//returns true if file deleted OK
```

```
assert new File('Directory2').delete()
```

```
assert ! new File('Directory2').exists()
```

```
new File('File1.txt').deleteOnExit()
```

```
assert new File('File1.txt').exists() //but
will be deleted when VM exits
```

```
def mod= new
```

```
File('File1.txt').lastModified()
```

```
assert new
```

```
File('File1.txt').setLastModified(mod -
```

```
60000)
```

```
    //60 seconds previously, returns true if  
successful
```

```
new File('File1.txt').lastModified= mod -  
120000
```

```
//property syntax for setting only
assert new File('File1.txt').lastModified()
== mod - 120000
```

To perform general file manipulation in a file system, we can retrieve all the topmost directories:

```
println
File.listRoots().toList().toString()
//listRoots() returns an array of File
objects
```

To create a temporary file, with given prefix (of at least 3 chars) and suffix:

```
File.createTempFile('Tem', '.txt')
//created in directory for temporary
files
File.createTempFile('Tem', '.txt', new
File('D:\\Groovy\\Scripts'))
//eg, created
D:/Groovy/Scripts/Tem59217.txt
```

We can read and write to files in various ways, as in this example:

```
def f1= new File('File1.txt') <<
'abcdefghijklmnop:qrstuvwxyz\n'
//create and write to the file
f1.leftShift('123:456:7890\n') //equivalent
method name
new File('File2.txt').createNewFile()
[new File('Directory1'), new
```

```
File('Directory2/SubDir1')] .each{
it.mkdirs() }
```

```
def list= []
new File('D:\\Groovy\\Scripts').eachFile{
list<< it.name }
    //eachFile() returns a list of File
objects
assert list ==
    ['Script.bat', 'File1.txt', 'File2.txt',
'Directory1', 'Directory2']
```

```
list= []
new
File('D:\\Groovy\\Scripts').eachFileMatch(~/
File.*?.txt/){ list<< it.name }
    //a regular expression, or any caseable
expression
assert list == ['File1.txt', 'File2.txt']
```

```
list= []
new
File('D:\\Groovy\\Scripts').eachFileRecurse{
list<< it.name }
assert list == ['Script.bat', 'File1.txt',
'File2.txt',
                'Directory1', 'Directory2',
'SubDir1']
```

```
list= []
new File('D:\\Groovy\\Scripts').eachDir{
```

```
list<< it.name }
assert list == ['Directory1', 'Directory2']
```

```
list= []
f1.eachLine{ list<< it }
assert list == [
  'abcdefg:hijklmnop:qrstuv:xyz',
  '123:456:7890' ]
```

```
list= f1.readlines()
assert list == [
  'abcdefg:hijklmnop:qrstuv:xyz',
  '123:456:7890' ]
```

```
list= []
f1.splitEachLine(':'){ list<< it } //splits
each line into a list
assert list == [
  ['abcdefg', 'hijklmnop', 'qrstuv',
  'xyz'],
  ['123', '456', '7890'],
]
```

```
def f2= new File('File2.txt')
f2.write('abcdefg\n') //can only write
strings
assert f2.getText() == 'abcdefg\n'
f2.append('hijklmnop,')
f2.append(42) //can append any object
assert f2.getText() == '''abcdefg
hijklmnop,42'''
```

```
f2.write('', 'unicode') //overwrites
existing contents
assert f2.getText('unicode') == ''
f2.append('', 'unicode') //also appends
unicode marker 0xFEFF
assert f2.getText('unicode') == '' + (0xFEFF
as char) + ''
```

```
[ new File('File1.txt'),
  new File('File2.txt'),
  new File('Directory1'),
  new File('Directory2/SubDir1'),
```

```
new File('Directory2'),
].each{ it.delete() } //delete files used by
this example
```

## JN2025-Streams

We can create streams of data from files, network resources, memory locations, etc, both input and output. To initially demonstrate the use of streams, we'll use streams around a file, both byte and Character streams. The methods introduced in these example can be used for any stream.

InputStreams and OutputStreams are streams of bytes:

```
def fos= new
FileOutputStream('TestFile.txt')

//These methods are available for all output
streams, not just FileOutputStream:
[ 21, 34, 43, 79 ].each{ fos.write(it) }
    //write out the lowest-order 8 bits of
the supplied integer
fos.flush()
fos.write([69, 32, 22] as byte[])
fos.write([10, 11, 12, 13, 88, 89] as
byte[], 3, 2)
    //write 2 bytes from array starting at
index 3
fos.close()
try{ fos.write(77); assert 0 }catch(e){
assert e instanceof IOException }
    //no writing after file closed

//check the byte contents of the file with a
```

**File utility method:**

```
assert new
```

```
File('TestFile.txt').readBytes().toList() ==  
    [ 21, 34, 43, 79, 69, 32, 22, 13, 88 ]
```

```
def fis= new FileInputStream('TestFile.txt')
```

```
//These methods are available for all input  
streams, not just FileInputStream:
```

```
assert fis.available() == 9
```

```
    //an estimate of bytes left for reading  
or skipping in the input stream
```

```
assert fis.read() == 21 //actually, the next  
byte is returned as an integer
```

```
fis.skip(2) //skip over, here, 2 bytes of  
data from the stream
```

```
assert fis.available() == 6
```

```
def ba2= new byte[3]
```

```
fis.read(ba2)
```

```
assert ba2.toList() == [79, 69, 32]
```

```
def ba3= new byte[6]
```

```
assert fis.read(ba3, 3, 2) == 2 //fill ba3  
with 2 elements from index 3,
```

```
                //return num
```

```
of elements copied, here, 2
```

```
assert ba3.toList() == [0, 0, 0, 22, 13, 0]
```

```
assert fis.read(ba3) == 1 //return num of  
elements copied, here, 1
```

```
assert ba3.toList() == [88, 0, 0, 22, 13, 0]
```

```
assert fis.read(ba3) == -1 //return -1 if
```

already at end-of-stream

```
//true if this input stream support the
mark() and reset() methods...
if( fis.markSupported() ){
    fis.reset()
        //reset reading to beginning of stream
if mark() hasn't ever been called
    assert fis.read() == 21
    fis.mark(0) //mark this position in the
stream; argument has no meaning here
    fis.read(new byte[4])
    fis.reset() //reset reading to where the
last mark() method was called
    assert fis.read() == 34
}
fis.close()
try{ fis.read(); assert 0 }catch(e){ assert
e instanceof IOException }
```

```
new File('TestFile.txt').delete() // delete  
the file used by this example
```

Readers and Writers are streams of Characters:

```
def fw= new FileWriter('TestFile.txt')  
  
//These methods are available for all  
writers, not just for FileWriter:  
[ 'a', 'b' ].each{ fw.write(it as char) }  
//write out the supplied character  
[ 'cd', 'efg' ].each{ fw.write(it) } //write  
out the supplied string  
fw.flush()  
fw.write(['h', 'i', 'j'] as char[])  
fw.write(['h', 'i', 'j', 'k', 'l', 'm'] as  
char[], 3, 2)  
    //write 2 chars from array starting at  
index 3  
fw.write('klmnopq', 2, 4) //write 4 chars  
from string starting at index 2  
fw.append('q' as char). //these Java 5.0  
methods allow chaining  
    append('rstuv').  
    append('vwxyz', 2, 6)  
        //use subsequence from index 2 to  
index 6 of supplied string  
fw.close()  
try{ fw.write('z'); assert 0 }catch(e){  
assert e instanceof IOException }
```

```
//no writing after file closed

assert new File('TestFile.txt').readLines()
== [ 'abcdefghijklmnopqrstuvwxyz' ]

def fr= new FileReader('TestFile.txt')

//These methods are available for all
readers, not just for FileReader:
if(fr.ready()){
    assert fr.read() == 'a'
    fr.skip(2) //skip over, here, 2 chars
    def ca2= new char[3]
    fr.read(ca2)
    assert ca2.toList()*.toString() == ['d',
'e', 'f']
    def ca3= new char[6]

    assert fr.read(ca3, 3, 2) == 2 //fill ca3
with 2 elements from index 3,
//return
num of elements copied, here, 2
    assert ca3.toList()*.toString() == ['\0',
'\0', '\0', 'g', 'h', '\0']
//similar
to InputStream method
    fr.skip(20)
    assert fr.read(ca3) == -1 //return -1 if
already at end-of-stream
```

```
//true if this input stream support the
mark() and reset() methods...
if( fr.markSupported() ){
    fr.reset()
        //reset reading to beginning of
stream if mark() hasn't ever been called
    assert fr.read() == 'a' as char
    fr.mark(0) //mark this position in the
stream; argument has no meaning here
    fr.read(new char[4])
    fr.reset() //reset reading to where the
last mark() method was called
    assert fr.read() == 'b' as char
}
fr.close()
try{ fr.read(); assert 0 }catch(e){ assert
e instanceof IOException }
}
```

```
new File('TestFile.txt').delete() //delete  
the file used by this example
```

#### Closing Streams

When we write to an output stream or writer such as `FileWriter`, we should always `close()` it in some way:

```
//here, because we don't close() the  
FileWriter, if there's an IOException,  
//some written data may be lost...  
def fw= new FileWriter('TestFile1.txt')  
try{  
    fw.write('abc\r\ndefg')  
    throw new IOException('') //simulate error  
on write() in previous line  
}catch(e){ }  
assert new  
File('TestFile1.txt').readLines().toList()  
== []  
    //nothing written because wasn't closed  
or flushed  
new File('TestFile1.txt').delete()  
assert new File('TestFile1.txt').exists()  
//not deleted because wasn't closed  
  
//here, we close() the FileWriter in a  
'finally' block, not losing any written  
//data...  
def fw2= new FileWriter('TestFile2.txt')  
try{  
    try{
```

```
        fw2.write('abc\r\ndefg')
        throw new IOException('') //simulate
error on write() in previous line
    }finally{
        fw2.close() //or flush() file so no data
will be lost when exception thrown
    }
}catch(e){ }
assert new File('TestFile2.txt').readLines()
== ['abc', 'defg']
    //contents written OK
new File('TestFile2.txt').delete()
assert ! new File('TestFile2.txt').exists()
//file deleted OK

//using withWriter() always closes the File,
whatever is thrown inside
//closure...
try{
    new File('TestFile3.txt').withWriter(){
w->
        w.write('abc\r\ndefg')
        throw new IOException('') //simulate
error on write() in previous line
    }
}catch(e){ }
new File('TestFile3.txt').delete()
```

```
assert ! new File('TestFile3.txt').exists()
    //deleted OK because withWriter() closed
the file
```

We can choose from many such methods to read and write characters to streams, where the stream is always closed automatically. Here's some methods which use a Reader and/or Writer. Although these examples use Files, all these methods work for other streamed resources also.

```
new File('TestFile1.txt').withWriter{ w->
    w<< 'abc' << 'def' //operator syntax
    w.leftShift('ghi').leftShift('jkl')
//equivalent method name
}

//file overwritten because it already
exists...

new
File('TestFile1.txt').withWriter('unicode'){
w->
    w<< 'abcdefghij'
}

new
File('TestFile1.txt').withWriterAppend('unic
ode'){ w->
    w<< 'klmnop' //although appending, unicode
marker 0xFEFF also added
}

//here, we'll use concatenation format for
string because it's easier to read
def fw= new FileWriter('TestFile1.txt')
```

```
fw.withWriter{ w->
  ['ab,cd\n' + 'efg\n' + 'hi,jk\n' + 'l',
  'mn,op'].each{
    w<< it
  }
}

new File('TestFile1.txt').withReader{ r->
  assert r.read() == 'a'
}

def list= []
new File('TestFile1.txt').eachLine{
  list<< it
}
assert list == ['ab,cd', 'efg', 'hi,jk',
  'lmn,op']

assert new File('TestFile1.txt').readLines()
==
  ['ab,cd', 'efg', 'hi,jk', 'lmn,op']

assert new File('TestFile1.txt').text ==
  'ab,cd\n' + 'efg\n' + 'hi,jk\n' +
  'lmn,op' //property

//filter lines from file, and write to
writer...
def fw2= new FileWriter('TestFile2.txt')
new File('TestFile1.txt').filterLine(fw2){
  line->
```

```

    ! line.contains('g')
}
assert new File('TestFile2.txt').text ==
    'ab,cd\r\n' + 'hi,jk\r\n' + 'lmn,op\r\n'
    // \n was changed to \r\n for Windows

def fw2a= new FileWriter('TestFile2.txt')
new
FileReader('TestFile1.txt').filterLine(fw2a)
{ line->
    ! line.contains('g')
}
assert new File('TestFile2.txt').text ==
    'ab,cd\r\n' + 'hi,jk\r\n' + 'lmn,op\r\n'

def fr2= new FileReader('TestFile2.txt')
assert [fr2.readLine(), fr2.readLine()] ==
['ab,cd', null]
    //known bug: only returns correctly on
first call
fr2.close()

new FileReader('TestFile2.txt').withReader{
r->
    def ca= new char[25]
    r.read(ca)
    assert ca.toList().join('').trim() ==
'ab,cd\r\n' + 'hi,jk\r\n' + 'lmn,op'
}

```

```
def list2= []
new
FileReader('TestFile2.txt').splitEachLine(',
'){ line->
  list2<< line
}
assert list2 == [ ['ab', 'cd'], ['hi',
'jk'], ['lmn', 'op'] ]

def fw2b= new FileWriter('TestFile2.txt')
new
FileReader('TestFile1.txt').transformLine(fw
2b){ line->
  if( line.contains(',') ) line += ',z'
  line
}
assert new File('TestFile2.txt').text ==
  'ab,cd,z\r\n' + 'efg\r\n' +
  'hi,jk,z\r\n' + 'lmn,op,z\r\n'
def fw2c= new FileWriter('TestFile2.txt')
new
FileReader('TestFile1.txt').transformLine(fw
2c){ line->
  if( line.contains(',') ) line += ',z'
  line
}
assert new File('TestFile2.txt').text ==
  'ab,cd,z\r\n' + 'efg\r\n' +
  'hi,jk,z\r\n' + 'lmn,op,z\r\n'

def fw2d= new FileWriter('TestFile2.txt')
```

```
new
FileReader('TestFile1.txt').transformChar(fw
2d){ ch->
  if(ch == ',') ch= '***'
  ch
}
assert new File('TestFile2.txt').text ==
  'ab***cd\n' + 'efg\n' + 'hi***jk\n' +
'lmn***op' // \n not converted
```

```
[new File('TestFile1.txt'), new
File('TestFile2.txt')].each{ it.delete() }
//delete files created by this example
```

Some methods which use an input and/or output stream which, although using Files in the examples, all work for other streamed resources also:

```
new File('TestFile1.txt').withOutputStream{
os->
  os<< ([95, 96] as byte[]) //operator
syntax for byte arrays
  os.leftShift( [97, 98, 99] as byte[] )
//equivalent method name
}
```

```
assert new
File('TestFile1.txt').readBytes().toList()
== [95, 96, 97, 98, 99]
```

```
def list= []
new File('TestFile1.txt').eachByte(){ b->
  list<< b
}
assert list == [95, 96, 97, 98, 99]
```

```
new
FileOutputStream('TestFile1.txt').withStream
{ os->
  os.write([100, 101, 102, 103] as byte[])
}
```

```
def list2= []
new
FileInputStream('TestFile1.txt').eachByte(){
b->
  list2<< b
}
assert list2 == [100, 101, 102, 103]
```

```
new File('TestFile1.txt').withInputStream{
is->
  def ba= new byte[5]
  is.read(ba)
  assert ba == [100, 101, 102, 103, 0]
}
```

```
new
FileInputStream('TestFile1.txt').withStream{
s->
  def ba= new byte[5]
  s.read(ba)
  assert ba == [100, 101, 102, 103, 0]
}
```

```
assert new
FileInputStream('TestFile1.txt').text ==
'defg'
```

```
assert new
FileInputStream('TestFile1.txt').getText('un
icode') == ''
```

```
new
FileInputStream('TestFile1.txt').withReader{
r->
  assert r.read() == 'd'
}
```

```
new
FileOutputStream('TestFile2.txt').withWriter
('unicode'){ w->
  w<< ''
}
assert new
FileInputStream('TestFile2.txt').getText('un
icode') == ''
```

```
new
FileOutputStream('TestFile2.txt').withWriter
{ w->
  w<< new FileInputStream('TestFile1.txt')
  //send contents of input stream
directly to output stream
  w<< 2.495 << '\n' //send an object to
output stream as string, returning
  //a writer, then send
another object to that writer
  w<< [3,4,5]
  //send another object to output
stream as string, returning a writer
}
assert new
FileInputStream('TestFile2.txt').text ==
```

```
'defg2.495\n' + '[3, 4, 5]'
```

```
def list3= []  
new  
FileInputStream('TestFile2.txt').eachLine{  
  line->  
    list3<< line  
}  
assert list3 == ['defg2.495', '[3, 4, 5]']
```

```
new  
FileInputStream('TestFile2.txt').readLine()  
== 'defg2.495'
```

```
new  
FileInputStream('TestFile2.txt').readLines()  
== ['defg2.495', '[3, 4, 5]']
```

```
def fw3= new FileWriter('TestFile3.txt')  
new  
FileInputStream('TestFile2.txt').filterLine(  
  fw3){ line->  
    line.contains('g')  
}  
assert new File('TestFile3.txt').readLines()  
== ['defg2.495']
```

```
[ new File('TestFile1.txt'),
```

```
new File('TestFile2.txt'),  
new File('TestFile3.txt')].each{  
it.delete() }
```

Although the examples above are for files, they're all available for streams, readers, and writers around all other resources also.

#### Resource-specific Streams

When we met the `FileInputStream`, `FileOutputStream`, `FileReader`, and `FileWriter` in the above examples, we constructed them with a single String. We can also construct them with a file, and add an 'append' flag:

```
def fos= new FileOutputStream(new  
File('TestFile.txt'), true)  
  
//appends to the file  
fos= new FileOutputStream(new  
File('TestFile.txt'), false) //overwrites  
the file  
fos= new FileOutputStream(new  
File('TestFile.txt')) //overwrites  
the file  
fos= new FileOutputStream('TestFile.txt',  
true) //appends to the file  
fos= new FileOutputStream('TestFile.txt',  
false) //overwrites the file  
fos= new FileOutputStream('TestFile.txt')  
//overwrites the file  
  
def fis= new FileInputStream(new  
File('TestFile.txt'))  
fis= new FileInputStream('TestFile.txt')
```

```
def fw= new FileWriter(new
File('TestFile.txt'), true) //appends to the
file
fw= new FileWriter(new File('TestFile.txt'),
true) //overwrites the file
fw= new FileWriter(new File('TestFile.txt'))
//overwrites the file
fw= new FileWriter('TestFile.txt', true)
//appends to the file
fw= new FileWriter('TestFile.txt', false)
//overwrites the file
fw= new FileWriter('TestFile.txt')
//overwrites the file
```

```
def fr= new FileReader(new  
File('TestFile.txt'))  
fr= new FileReader('TestFile.txt')
```

There are many other streams, readers, and writers that wrap around specific resources. `ByteArrayInputStream` and `ByteArrayOutputStream` wrap around an array of bytes:

```

def bais= new ByteArrayInputStream( [33, 34,
35] as byte[] )
[33, 34, 35, -1].each{ assert bais.read() ==
it }
def bais2=
    new ByteArrayInputStream( [33, 34, 35,
36, 37, 38, 39] as byte[], 2, 4 )
[35, 36, 37, 38, -1].each{ assert
bais2.read() == it }

def baos= new ByteArrayOutputStream()
baos.write([100, 101, 102, 103] as byte[])
assert baos.size() == 4
assert baos.toByteArray().toList() == [100,
101, 102, 103]

def baos2= new ByteArrayOutputStream(10)
    //we can specify initial size of
internal buffer
baos.writeTo( baos2 ) //we can writeTo any
OutputStream
assert baos2.toByteArray().toList() == [100,
101, 102, 103]
assert baos2.toString() == 'defg'
assert baos2.toString('unicode') == ''
baos2.reset()
assert baos2.toByteArray().toList() == []

```

CharArrayReader and CharArrayWriter wrap around an array of chars:

```

def car= new CharArrayReader( ['a', 'b',
    'c'] as char[] )
['a', 'b', 'c', -1].each{ assert car.read()
    == it }
def car2=
    new CharArrayReader( ['a', 'b', 'c',
        'd', 'e', 'f', 'g'] as char[], 2, 4 )
['c', 'd', 'e', 'f', -1].each{ assert
    car2.read() == it }

def caw= new CharArrayWriter()
caw.write(['a', 'b', 'c', 'd'] as char[])
assert caw.size() == 4
assert caw.toCharArray().toList() == ['a',
    'b', 'c', 'd'].collect{ it as char }

def caw2= new CharArrayWriter(10)
    //we can specify initial size of
internal buffer
caw.writeTo( caw2 ) //we can writeTo any
Writer
assert caw2.toCharArray().toList() == ['a',
    'b', 'c', 'd'].collect{ it as char }
assert caw2.toString() == 'abcd'
caw2.reset()
assert caw2.toCharArray().toList() == []

```

StringReader and StringWriter wrap around a StringBuffer:

```
def sr= new StringReader( 'abcde' )  
['a', 'b', 'c', 'd', 'e', -1].each{ assert  
sr.read() == it }
```

```
def sw= new StringWriter()  
sw= new StringWriter(10) //we can specify  
initial size of StringBuffer  
sw.write( 'abcde' )  
assert sw.buffer.toString() == 'abcde'  
assert sw.toString() == 'abcde'
```

InputStreamReader and OutputStreamWriter are a reader and writer pair that forms the bridge between byte streams and character streams. An InputStreamReader reads bytes from an InputStream and converts them to characters using a character-encoding, either the default or one specified by name. Similarly, an OutputStreamWriter converts characters to bytes using a character-encoding and then writes those bytes to an OutputStream. In this example, we use a FileInputStream and FileOutputStream, but any InputStream or OutputStream could be used:

```

def wtr= new OutputStreamWriter(new
FileOutputStream('TheOutput.txt'))
wtr<< 'abc'
wtr.close()

def rdr= new InputStreamReader(new
FileInputStream('TheOutput.txt'))
def list= []
rdr.eachLine{ list<< it }
assert list == ['abc']

println System.getProperty("file.encoding")
    //to see the default file encoding used

wtr= new OutputStreamWriter(new
FileOutputStream('TheOutput.txt'),
'unicode')
wtr<< 'def'
println wtr.encoding //perhaps, 'UTF-16', as
'unicode' above is an alias
wtr.close()

rdr= new InputStreamReader(new
FileInputStream('TheOutput.txt'), 'unicode')
println rdr.encoding
list= []
rdr.eachLine{ list<< it }
assert list == ['def']

```

The buffered streams, reader, and writer wrap around another, buffering the data read or written so as to provide for

the efficient processing of bytes, characters, arrays, and lines. It's very useful for streams, readers, and writers whose input/output operations are costly, such as files.

```
def bos= new BufferedOutputStream(new
FileOutputStream('TheOutput.txt'))
println bos.buf.size() //see the size of the
default buffer
bos= new BufferedOutputStream(new
FileOutputStream('TheOutput.txt'), 16384)
    //set the buffer size
assert bos.buf.size() == 16384
bos= new
File('TheOutput.txt').newOutputStream()
    //returns a buffered output stream

def bis= new BufferedInputStream(new
FileInputStream('TheOutput.txt'))
bis= new BufferedInputStream(new
FileInputStream('TheOutput.txt'), 16384)
    //set the buffer size
bis= new
File('TheOutput.txt').newInputStream()
    //returns a buffered input stream

def bwtr= new BufferedWriter(new
FileWriter('TheOutput.txt'))
bwtr= new BufferedWriter(new
FileWriter('TheOutput.txt'), 16384)
    //set the buffer size
bwtr= new File('TheOutput.txt').newWriter()
//returns a buffered writer
```

```
bwtr= new
File('TheOutput.txt').newWriter('unicode')
bwtr= new
File('TheOutput.txt').newWriter(true)
//appends to the file
bwtr= new
File('TheOutput.txt').newWriter('unicode',
true) //appends to the file

def brdr= new BufferedReader(new
FileReader('TheOutput.txt'))
brdr= new BufferedReader(new
FileReader('TheOutput.txt'), 16384)
    //set the buffer size
brdr= new File('TheOutput.txt').newReader()
//returns a buffered reader
brdr= new
File('TheOutput.txt').newReader('unicode')
brdr= new
FileInputStream('TheOutput.txt').newReader()

def file= new File('TheOutput.txt')
def wtr= file.newWriter()
wtr.writeLine('abc')
wtr.writeLine('def')
wtr.newLine() //writes blank line
wtr.close()
def rdr= file.newReader()
assert rdr.readLine() == 'abc' //doesn't
```

```
return end-of-line characters
assert rdr.text == 'def' + '\r\n' + '\r\n'
//returns end-of-line characters
```

A `SequenceInputStream` joins two other streams together:

```
def f1= new File('TheOutput1.txt'), f2= new
File('TheOutput2.txt')
f1<< 'abcde'; f2<< 'fghij'
def is1= new FileInputStream(f1), is2= new
FileInputStream(f2)
def sis= new SequenceInputStream(is1, is2)
assert sis.text == 'abcdefghij'
```

`SequenceInputStream` can also join three or more streams together using a `Vector`. See the upcoming tutorial on multi-threading for more on `Vectors`:

```
def f1= new File('TheOutput1.txt'),
    f2= new File('TheOutput2.txt'),
    f3= new File('TheOutput3.txt')
f1<< 'abc'; f2<< 'def'; f3<< 'ghij'
def list=[ new FileInputStream(f1),
           new FileInputStream(f2),
           new FileInputStream(f3) ]
def sis= new SequenceInputStream(new
Vector(list).elements())
assert sis.text == 'abcdefghij'
```

A line-number reader keeps track of line numbers:

```
def w= new File('TheOutput.txt').newWriter()
w.writeLine('abc'); w.writeLine('defg');
w.close()
```

```
def lnr= new LineNumberReader(new
FileReader('TheOutput.txt'))
lnr= new LineNumberReader(new
FileReader('TheOutput.txt'), 16384)
    //set the buffer size
assert lnr.lineNumber == 0
assert lnr.readLine() == 'abc'
assert lnr.lineNumber == 1
lnr.lineNumber= 4
assert lnr.readLine() == 'defg'
assert lnr.lineNumber == 5
```

A pushback input stream allows read input to be pushed back on:

```
def ba= [7, 8, 9, 10, 11, 12, 13] as byte[]
def pis= new PushbackInputStream(new
ByteArrayInputStream(ba))
pis= new PushbackInputStream(new
ByteArrayInputStream(ba), 1024)
    //or specify buffer size
def ba2= new byte[3]
pis.read(ba2)
assert ba2.toList() == [7, 8, 9]

pis.unread(2)
pis.read(ba2)
assert ba2.toList() == [2, 10, 11]

pis.unread([3, 4, 5, 6] as byte[])
pis.read(ba2)
assert ba2.toList() == [3, 4, 5]
pis.read(ba2)
assert ba2.toList() == [6, 12, 13]
```

A pushback reader provides a similar facility for characters:

```
def ca= ['g', 'h', 'i', 'j', 'k', 'l', 'm']
as char[]
def prdr= new PushbackReader(new
CharArrayReader(ca))
prdr= new PushbackReader(new
CharArrayReader(ca), 1024)
    //or specify buffer size
def ca2= new char[3]
prdr.read(ca2)
assert ca2.toList() == ['g', 'h',
'i'].collect{it as char}

prdr.unread('b' as int)
prdr.read(ca2)
assert ca2.toList() == ['b', 'j',
'k'].collect{it as char}

prdr.unread(['c', 'd', 'e', 'f'] as char[])
prdr.read(ca2)
assert ca2.toList() == ['c', 'd',
'e'].collect{it as char}
prdr.read(ca2)
assert ca2.toList() == ['f', 'l',
'm'].collect{it as char}

prdr.unread(['a', 'b', 'c', 'd', 'e', 'f',
'g'] as char[], 1, 4)
    //offset 1, length 4 of array
prdr.read(ca2)
assert ca2.toList() == ['b', 'c',
'd'].collect{it as char}
```

A `DataOutputStream` writes out Groovy structures as bytes, and a `DataInputStream` reads such bytes in as Groovy structures:

```
def baos= new ByteArrayOutputStream(30)
def dos= new DataOutputStream(baos)
assert dos.size() == 0

def bais= new ByteArrayInputStream( baos.buf
)
def dis= new DataInputStream(bais)

dos.writeBoolean( true )
assert baos.toByteArray().toList() == [1]
//writes boolean as a 1-byte value
assert dis.readBoolean() == true

dos.writeByte( 200 ) //converted to -56, a
1-byte value
assert baos.toByteArray().toList() == [1,
-56]
    //'true', followed by '200 as byte'
assert dis.readByte() == -56
dis.reset() //resets input stream
dis.skipBytes(1) //we can skip bytes
assert dis.readUnsignedByte() == 200
baos.reset() //flushes backing stream
dis.reset()

dos.writeBytes('abcdefg') //writes string as
a sequence of bytes
```

```
assert baos.toByteArray() as List == [97,
98, 99, 100, 101, 102, 103]
dis.reset()
def ba= new byte[5]
dis.readFully(ba) //readFully() is converse
of writeBytes()
assert ba as List == [97, 98, 99, 100, 101]
dis.reset()
```

```
ba= new byte[5]
dis.readFully(ba, 1, 2) //offset 1 and
length 2 of ba
assert ba as List == [0, 97, 98, 0, 0]
baos.reset(); dis.reset()
```

```
dos.writeChar('a' as int) //writes char as
2-byte value, high byte first
assert baos.toByteArray() as List == [0, 97]
assert dis.readChar() == 'a'
baos.reset(); dis.reset()
```

```
dos.writeChars('ab') //writes string as a
sequence of characters
assert baos.toByteArray() as List == [0, 97,
0, 98]
baos.reset(); dis.reset() //DataInputStream
has no readChars() method
```

```
dos.writeShort(5000) //writes a short as two
bytes, high byte first
assert baos.toByteArray() as List == [19,
```

```
-120] && 20*256 - 120 == 5000
assert dis.readShort() == 5000
dis.reset()
dis.readUnsignedShort() == 5000 //similar to
readUnsignedByte()
baos.reset(); dis.reset()
```

```
dos.writeInt(5000) //writes an integer as
four bytes, high byte first
assert baos.toByteArray() as List == [0, 0,
19, -120]
assert dis.readInt() == 5000
baos.reset(); dis.reset()
```

```
dos.writeLong(5000) //writes a long as eight
bytes, high byte first
assert baos.toByteArray() as List == [0, 0,
0, 0, 0, 0, 19, -120]
assert dis.readLong() == 5000
baos.reset(); dis.reset()
```

```
dos.writeDouble(123.456)
    //calls Double.doubleToLongBits(),
writes as 8 bytes, high first
println baos.toByteArray() as List
assert dis.readDouble() == 123.456d
baos.reset(); dis.reset()
```

```
dos.writeFloat(123.456f)
    //calls Float.floatToIntBits(), writes
as 4 bytes, high first
```

```
println baos.toByteArray() as List
assert dis.readFloat() == 123.456f
baos.reset(); dis.reset()
```

```
dos.writeUTF('abc')
    //writes using "modified UTF-8 encoding
in a machine-independent manner"
assert baos.toByteArray() as List == [0, 3,
97, 98, 99]
    //UTF-8 adds 0, 3 at beginning
assert dis.readUTF() == 'abc'
dis.reset()
```

```
assert DataInputStream.readUTF(dis) == 'abc'  
    //a static method to perform the same  
action
```

We'll meet more different types of streams, readers, and writers in the tutorials on Inheritance, Networking, Multi-threading, and others coming up.

ObjectInputStream and ObjectOutputStream sugar

There are also helper methods for Object input/output classes as this example shows:

```
@Immutable class Point implements  
Serializable { int x, y }  
  
def file = new File('points.dat')  
def square = [ new Point(10, 10),  
               new Point(20, 10),  
               new Point(20, 20),  
               new Point(10, 20) ]  
file.withObjectOutputStream { oos ->  
    oos.writeObject(square)  
}  
file.withObjectInputStream(getClass().classL  
oader){ ois ->  
    def saved = ois.readObject( )  
    assert square == saved  
}
```

JN2515-Closures

Blocks

We can embed a sequence of statements inside "try", called a "block". Defined variables are only visible within that block, not outside:

```
def a = 'good morning'
try{
  def b = 'greetings', c = 'nice day'
      //'def' keyword applies to
both 'b' and 'c'

  assert a == 'good morning'
  assert b == 'greetings'
}
assert a == 'good morning'
//println b //a compile error if
uncommented: b not visible here
```

Using the "def" keyword is optional because we are inside a script:

```
def c = 5
assert c == 5
d = 6
assert d == 6 //def keyword optional because
we're within a script context
assert binding.variables.c == null
assert binding.variables.d == 6
      //when def not used, variable
becomes part of binding.variables
```

But variables without "def" are visible outside the block:

```
try{
  h = 9
  assert binding.variables.h == 9
}
assert h == 9
assert binding.variables.h == 9
```

We can't define a variable (using "def") with the same name as another already visible (ie, another "in scope"):

```
def a = 'island'
//def a = 'snake' //a compile error if
uncommented: a already defined
try{
  //def a = 'jewel' //a compile error if
uncommented: a already defined
}
```

We can nest blocks:

```
def a = 123
try{
  try{
    try{
      assert a == 123
    }
  }
}
```

Closures

We can take a sequence of statements that refers to its external context and assign it to a variable, then execute it later. It's technically called a "closable block", commonly called a "closure":

```
def a = 'coffee'
def c = {
  def b = 'tea'
  a + ' and ' + b //a refers to the variable
a outside the closure,
                //and is remembered by the
closure
}
assert c() == 'coffee and tea' //short for
c.call()
```

The closure assigned to the variable (here, c) will remember its context (here, including a) even if that context is not in scope when the closure is called:

```
def c
try{
  def a = 'sugar'
  c = { a } //a closure always returns its
only value
}
assert c() == 'sugar'
def d = c //we can also assign the closure
to another variable
assert d() == 'sugar'
```

A closure always returns a value, the result of its last statement:

```
giveSeven = { 7 }  
assert giveSeven() == 7 //value of last  
statement is returned  
  
giveNull = { def a }  
assert giveNull() == null //null returned if  
last statement has no value
```

By putting a closure within another, we can create two instances of it:

```
c = { def e = { 'milk' }; e }  
d = c  
assert c == d  
v1 = c()  
v2 = c()  
assert v1 != v2
```

#### Closure Parameters

We can put parameters at the beginning of a closure definition, and pass values in when we call the closure:

```
def toTriple = {n -> n * 3}  
assert toTriple.call( 5 ) == 15
```

We can also pass information out using the parameters:

```

def f = { list, value -> list << value }
x = []
f(x, 1)
f(x, 2,) //trailing comma in argument list
OK
f(x, 3)
assert x == [1, 2, 3]

```

One parameter is always available, called "it", if no explicit parameters are named:

```

c = { it*3 }
assert c( 'run' ) == 'runrunrun'

```

If parameters aren't specified, "it" will still be implicitly defined, but be null:

```

//c = { def it = 789 }
      //a compile error when
uncommented: 'it' already implicitly defined
c = { value1 -> def it = 789; [value1, it] }
      //works OK because no 'it' among
parameters
assert c( 456 ) == [456, 789]
c = {-> def it = 789; it } //zero
parameters, not even 'it', so works OK
assert c() == 789

```

Parameters can't have the same name as another variable in scope, except for the implicit parameter 'it':

```

def name= 'cup'
//def c={ name-> println (name) } //a
compile error when uncommented:
//current
scope already contains name 'name'
c= { def d= { 2 * it }; 3 * d(it) }
    //'it' refers to
immediately-surrounding closure's parameter
in each case
assert c(5) == 30

```

If there's already a variable called 'it' in scope, we can access it using owner.it:

```

it= 2
c= { assert it == 3; assert owner.it == 2 }
c(3)

```

We can pass one closure into another as a parameter:

```

toTriple = {n -> n * 3}
runTwice = { a, c -> c( c(a) )}
assert runTwice( 5, toTriple ) == 45

```

We can return a closure from another:

```

def times= { x -> { y -> x * y }}
assert times(3)(4) == 12

```

There's a shortcut syntax when explicitly defining a closure within another closure call, where that closure is the last or only parameter:

```

def runTwice = { a, c -> c(c(a)) }
assert runTwice( 5, {it * 3} ) == 45 //usual
syntax
assert runTwice( 5 ){it * 3} == 45
    //when closure is last param, can put it
after the param list

def runTwiceAndConcat = { c -> c() + c() }
assert runTwiceAndConcat( { 'plate' } ) ==
'plateplate' //usual syntax
assert runTwiceAndConcat(){ 'bowl' } ==
'bowlbowl' //shortcut form
assert runTwiceAndConcat{ 'mug' } ==
'mugmug'
    //can skip parens altogether if closure
is only param

def runTwoClosures = { a, c1, c2 ->
c1(c2(a)) }
    //when more than one closure as last
params
assert runTwoClosures( 5, {it*3}, {it*4} )
== 60 //usual syntax
assert runTwoClosures( 5 ){it*3}{it*4} == 60
//shortcut form

```

Arguments in a closure call can be named. They are interpreted as the keys in a map passed in as the first parameter:

```

def f= {m, i, j-> i + j + m.x + m.y }
assert f(6, x:4, y:3, 7) == 20

def g= {m, i, j, k, c-> c(i + j + k, m.x +
m.y) }
assert g(y:5, 1, 2, x:6, 3){a,b-> a * b } ==
66

```

We can enquire the number of parameters for a closure, both from inside and outside the closure:

```

c= {x,y,z-> getMaximumNumberOfParameters() }
assert c.getMaximumNumberOfParameters() == 3
assert c(4,5,6) == 3

```

A closure may have its last parameter/s assigned default value/s:

```

def e = { a, b, c=3, d='a' -> "${a+b+c}$d" }
assert e( 7, 4 ) == '14a'
assert e( 9, 8, 7 ) == '24a' //override
default value of 'c'

```

A closure can take a varying number of arguments by prefixing its last parameter with `Object[]`, and accessing them using `'each'`:

```

def c = { arg, Object[] extras ->
  def list= []
  list<< arg
  extras.each{ list<< it }
  list
}
assert c( 1 )           == [ 1 ]
assert c( 1, 2 )       == [ 1, 2 ]
assert c( 1, 2, 3 )    == [ 1, 2, 3 ]
assert c( 1, 2, 3, 4 ) == [ 1, 2, 3, 4 ]

```

We can also prefix the last parameter of a closure with `Closure[]` to pass in a varying number of other closures, even using the shortcut syntax:

```

def apply = { a, Closure[] cc ->
  (cc as List).inject(a){ flo, it-> it(flo)
}
      //apply the closures nestedly to
the initial value
}
assert apply(7){it*3}{it+1}{it*2}.toString()
== '44'

```

When we call a closure with a list argument, if there's no closure defined with a list parameter, the arguments are passed in as separate parameters:

```

def c= {a, b, c-> a + b + c}
def list=[1,2,3]
assert c(list) == 6

```

A closure may be copied with its first parameter/s fixed to a constant value/s, using curry:

```
def concat = { p1, p2, p3 -> "$p1 $p2 $p3" }
def concatAfterFly = concat.curry( 'fly' )
assert concatAfterFly( 'drive', 'cycle' ) ==
'fly drive cycle'
def concatAfterFlySwim =
concatAfterFly.curry( 'swim' )
assert concatAfterFlySwim( 'walk' ) == 'fly
swim walk'
```

In closures, we can use currying and parameter-count-varying together:

```
def c = { arg, Object[] extras -> arg + ', '
+ extras.join(', ' ) }
def d = c.curry( 1 ) //curry first param
only
assert d( 2, 3, 4 ) == '1, 2, 3, 4'
def e = c.curry( 1, 3 ) //curry part of
Object[] also
assert e( 5 ) == '1, 3, 5'
def f = e.curry( 5, 7, 9, 11 ) //currying
continues on Object
assert f( 13, 15 ) == '1, 3, 5, 7, 9, 11,
13, 15'
```

We can make closures recursive:

```
def gcd //predefine closure name
gcd={ m,n-> m%n==0? n: gcd(n,m%n) }
assert gcd( 28, 35 ) == 7
```

We can even make a recursion of anonymous closures (thanks to 'call' method available for each closure)

```
def results = [];
{ a, b ->
  results << a
  a<10 && call(b, a+b)
}(1,1)
assert results == [1, 1, 2, 3, 5, 8, 13] //
Fibonacci numbers
```

## Functions

A function is similar to a closure, though a function can't access defined variables in its surrounding context:

```

a = 32 //def keyword not used for this one
def c = 'there', d = 'yonder'
def f(){
    assert a == 32 //outer 'a' visible because
    'def' keyword wasn't used with it
    def c = 'here'
    //compiles OK because other defined c
invisible inside function definition
    //println d //a compile error when
uncommented: d not accessable
    c
}
assert f() == 'here' //syntax to invoke a
function

```

The def keyword is compulsory when defining functions:

```

def f(){
    a = 1
    c = { 'here, again' }
    c()
}
assert f() == 'here, again'
//g(){ println 'there, again' }
    //a compile error when uncommented: def
keyword required

```

We use a special syntax to assign a function to another variable when using the original definition name:

```
def f(){ 77 } //define function using name
'f'
assert f() == 77
def g = this.&f //special syntax to assign
function to another variable
assert g() == 77
def h = g //don't use special syntax here
assert h() == 77
f = 'something else' //this 'f' is a
VARIABLE, not the function NAME
assert f() == 77 //the function name can't
be reassigned
```

Unlike blocks and closures, we can't nest functions:

```

def f(){
  //def g1(){ println 'there' }
  //a compile error when uncommented:
  can't nest functions
  'here'
}
assert f() == 'here'
try{
  //def g2(){ println 'yonder' }
  //a compile error when uncommented:
  can't nest functions
}
c = {
  //def g3(){ println 'outer space' }
  //a compile error when uncommented:
  can't nest functions
}
def h(){
  try{ def c = { 'here, again' } }
  //we can have blocks and closures
  within functions
}

```

#### Function Parameters

A function can have parameters, with which we can pass information both in and out:

```
def foo( list, value ){
  list << value
}
x = []
foo(x, 1)
foo(x, 2)
assert x == [1, 2]
```

We can have more than one function of the same name if they each have different numbers of (untyped) parameters.

```
def foo(value){ 'v1' }
def foo(list, value){ 'v2' }
assert foo(9) == 'v1'
assert foo([], 1) == 'v2'
```

A function returns a value, unless prefixed by void instead of def, when it always returns null:

```

def f1(){ 7 }
assert f1() == 7 //value of last statement
is returned

def f2(){ return 8; 3 }
assert f2() == 8 //return explicitly using
return

void f3(){ 10 }
assert f3() == null //null always returned

//void f4(){ return 9 }
//a compile error when uncommented:
can't use 'return' in a void function

```

When there's a method and closure with the same name and parameters, the method is chosen instead of the closure:

```

def c(){'method c'}
def c= {-> 'closure c'}
assert c() == 'method c'

def d(i){'method d'}
def d= {'closure d'}
assert d(9) == 'method d'

```

Some Similarities with Closures

We can use the shortcut invocation syntax for closure parameters:

```
def f(Closure c){ c() }  
assert f{ 'heehee' } == 'heehee'
```

A function may have its last parameter/s assigned default value/s:

```
def dd( a, b=2 ) { "$a, $b" }  
assert dd( 7, 4 ) == '7, 4'  
assert dd( 9 ) == '9, 2'
```

Arguments in a function call can be named, interpreted as keys in a map passed in as first parameter:

```
def f(m, i, j){ i + j + m.x + m.y }  
assert f(6, x:4, y:3, 7) == 20  
  
def g(m, i, j, k, c){ c(i + j + k, m.x +  
m.y) }  
assert g(y:5, 1, 2, x:6, 3){a,b-> a * b } ==  
66
```

A function can take a varying number of arguments by prefixing its last argument by Object[], and accessing them using each:

```

def c( arg, Object[] extras ){
  def list= []
  list<< arg
  extras.each{ list<< it }
  list
}
assert c( 1 ) == [ 1 ]
assert c( 1, 2, 3, 4 ) == [ 1, 2, 3, 4 ]

```

When we call a function with a list argument, if there's none defined with a list parameter, the arguments are passed in separately:

```

def x(a, b, c){a + b + c}
def list=[1,2,3]
assert x(list) == 6

```

We can call a function recursively by referencing its own name:

```

def gcd( m, n ){ if( m%n == 0 )return n;
gcd(n,m%n) }
assert gcd( 28, 35 ) == 7

```

## JN2525-Classes

### Accessing Private Variables

Closures and functions can't remember any information defined within themselves between invocations. If we want a closure to remember a variable between invocations, one only it has access to, we can nest the definitions inside a block:

```

def c
try{
  def a= new Random() //only closure c can
see this variable; it is private to c
  c= { a.nextInt(100) }
}
100.times{ println c() }
try{ a; assert 0 }catch(e) //'a'
inaccessible here
{ assert e instanceof
MissingPropertyException }

```

We can have more than one closure accessing this private variable:

```

def counterInit, counterIncr, counterDecr,
counterShow
  //common beginning of names to show
common private variable/s
try{
  def count
  counterInit= { count= it }
  counterIncr= { count++ }
  counterDecr= { count-- }
  counterShow= { count }
}
counterInit(0)
counterIncr(); counterIncr(); counterDecr();
counterIncr()
assert counterShow() == 2

```

We can also put all closures accessing common private variables in a map to show they're related:

```
def counter= [:]
try{
  def count= 0
  counter.incr= { count++; counter.show() }
  counter.decr= { count--; counter.show() }
  counter.show= { count }
}

counter.incr()
assert counter.show() == 1
```

Expando

We can access private variables with an Expando instead. An expando allows us to assign closures to Expando names:

```

def counter= new Expando()
try{
  def count= 0
  counter.incr= { count++; show() }
                //no need to qualify closure call
with expando name
  counter.decr= { count--; show() }
  counter.show= { timesShown++; count }
  counter.timesShown= 0
                //we can associate any value, not
just closures, to expando keys
}
counter.incr(); counter.incr();
counter.decr(); counter.incr()
assert counter.show() == 2

```

An expando can also be used when common private variables aren't used:

```

def language= new Expando()
language.name= "Groovy"
language.numLetters= { name.size() }

assert language.numLetters() == 6
language.name= "Ruby"
assert language.numLetters() == 4
language.name= "PHP"
assert language.numLetters() == 3

```

Like individual closures, closures in expandos see all external variables all the way to the outermost block. This is not always helpful for large programs as it can limit our choice of names:

```

def a= 7
try{
  //... .. lots of lines and blocks in
  between ... ..
  def exp= new Expando()
  exp.c= {
    //def a= 2 //does not compile if
uncommented: a is already defined
    //... ..
  }
}

```

For single-argument closures, both standalone and within expandos, we can use the implicit parameter as a map for all variables to ensure they're all valid, though the syntax is not very elegant:

```

def a= 7
try{
  def c= {
    it= [it: it]
    it.a= 2
    it.it + it.a
  }
  assert c(3) == 5
}

```

There is a better way to ensure a chosen variable name will not "shadow" another from the same scope.

#### Static Classes

Just as we can use functions instead of closures to hide names from the surrounding context, so also we can use static classes instead of expandos to hide such external names. We use the static keyword to qualify the individual definitions in a class definition:

```

def a= 7
def a= 7
class Counter{
    //variable within a class is called a
    field...
    static public count= 0
        //count has 'public' keyword, meaning
    it's visible from outside class

    //function within a class is called a
    method...
    static incr(){
        count++
        //variables defined within class
    visible from everywhere else inside class
    }
    static decr(){
        //println a //compile error if
    uncommented:
        //a is outside the class and
    not visible
        count--
    }
}
Counter.incr(); Counter.incr();
Counter.decr(); 5.times{ Counter.incr() }
assert Counter.count == 6

```

Methods act quite similar to standalone functions. They can take parameters:

```

class Counter{
    static private count = 0
        //qualified with private, meaning not
visible from outside class
    static incr( n ){ count += n }
    static decr( count ){ this.count -= count
}
        //params can have same name as a
field; 'this.' prefix accesses field
    static show(){ count }
}
Counter.incr(2); Counter.incr(7);
Counter.decr(4); Counter.incr(6)
assert Counter.show() == 11

```

We can have more than one method of the same name if they each have different numbers of parameters.

```

class Counter{
    static private count = 0
    static incr(){ count++ }
    static incr( n ){ count += n }
    static decr(){ count-- }
    static decr( n ){ count -= n }
    static show(){ count }
}
Counter.incr(17); Counter.incr();
Counter.decr(4)
assert Counter.show() == 14

```

Methods are also similar to other aspects of functions:

```

class U{
  static a(x, Closure c){ c(x) }
  static b( a, b=2 ){ "$a, $b" } //last
argument/s assigned default values
  static c( arg, Object[] extras ){ arg +
extras.inject(0){ flo, it-> flo+it } }
  static gcd( m, n ){ if( m%n == 0 )return
n; gcd(n,m%n) }

//recursion by calling own name
}
assert U.a(7){ it*it } == 49 //shorthand
passing of closures as parameters
assert U.b(7, 4) == '7, 4'
assert U.b(9) == '9, 2'
assert U.c(1,2,3,4,5) == 15 //varying number
of arguments using Object[]
assert U.gcd( 28, 35 ) == 7

```

We can assign each method of a static class to a variable and access it directly similar to how we can with functions:

```

class U{
    static private a= 11
    static f(n){ a*n }
}
assert U.f(4) == 44
def g= U.&f //special syntax to assign
method to variable
assert g(4) == 44
def h = g //don't use special syntax here
assert h(4) == 44

```

When there's no accessibility keyword like 'public' or 'private' in front of a field within a static class, it becomes a property, meaning two extra methods are created:

```

class Counter{
    static count = 0
    //property because no accessibility
keyword (eg 'public', 'private')
    static incr( n ){ count += n }
    static decr( n ){ count -= n }
}
Counter.incr(7); Counter.decr(4)
assert Counter.count == 3
assert Counter.getCount() == 3 //extra
method for property, called a 'getter'
Counter.setCount(34) //extra
method for property, called a 'setter'
assert Counter.getCount() == 34

```

When we access the property value using normal syntax, the 'getter' or 'setter' is also called:

```

class Counter{
    static count= 0 //'count' is a property

    //we can define our own logic for the
getter and/or setter...
    static setCount(n){ count= n*2 } //set the
value to twice what's supplied
    static getCount(){ 'count: '+ count }
                        //return the value as a
String with 'count: ' prepended
}
Counter.setCount(23) //our own 'setCount'
method is called here
assert Counter.getCount() == 'count: 46'
                        //our own 'getCount'
method is called here
assert Counter.count == 'count: 46'
                        //our own 'getCount'
method is also called here
Counter.count= 7
assert Counter.count == 'count: 14'
                        //our own 'setCount' method
was also called in previous line

```

To run some code, called a static initializer, the first time the static class is accessed. We can have more than one static initializer in a class.

```
class Counter{
    static count = 0
    static{ println 'Counter first accessed' }
//static initializer
    static incr( n ){ count += n }
    static decr( n ){ count -= n }
}
println 'incrementing...'
Counter.incr(7) //'Counter first accessed'
printed here
println 'decrementing...'
Counter.decr(4) //nothing printed
```

#### Instantiable Classes

We can write instantiable classes, templates from which we can construct many instances, called objects or class instances. We don't use the static keyword before the definitions within the class:

```
class Counter{
  def count = 0 //must use def inside
  classes if no other keyword before name
  def incr( n ){ count += n }
  def decr( n ){ count -= n }
}
def c1= new Counter() //create a new object
from class
c1.incr(2); c1.incr(7); c1.decr(4);
c1.incr(6)
assert c1.count == 11

def c2= new Counter() //create another new
object from class
c2.incr(5); c2.decr(2)
assert c2.count == 3
```

We can run some code the first time each object instance is constructed. First, the instance initializer/s are run. Next run is the constructor with the same number of arguments as in the calling code.

```

class Counter{
  def count
  { println 'Counter created' }
  //instance initializer shown by using
standalone curlies
  Counter(){ count= 0 }
  //instance constructor shown by using
class name
  Counter(n){ count= n }
  //another constructor with a different
number of arguments
  def incr( n ){ count += n }
  def decr( n ){ count -= n }
}
c = new Counter() //'Counter created'
printed
c.incr(17); c.decr(2)
assert c.count == 15
d = new Counter(2) //'Counter created'
printed again
d.incr(12); d.decr(10); d.incr(3)
assert d.count == 7

```

If we don't define any constructors, we can pass values directly to fields within a class by adding them to the constructor call:

```
class Dog{  
  def sit  
  def number  
  def train(){ ([sit()] * number).join(' ' )  
}  
}  
def d= new Dog( number:3, sit: {'Down boy!'}  
)  
assert d.train() == 'Down boy! Down boy!  
Down boy!'
```

Methods, properties, and fields on instantiable classes act similarly to those on static classes:

```

class U{
  private timesCalled= 0 //qualified with
visibility, therefore a field
  def count = 0 //a property
  def a(x){ x }
  def a(x, Closure c){ c(x) } //more than
one method of the same name but
//each having
different numbers of parameters

  def b( a, b=2 ){ "$a, $b" } //last
argument/s assigned default values
  def c( arg, Object[] extras ){ arg +
extras.inject(0){ flo, it-> flo+it } }
  def gcd( m, n ){ if( m%n == 0 )return n;
gcd(n,m%n) }
//recursion by
calling own name
}
def u=new U()
assert u.a(7){ it*it } == 49 //shorthand
passing of closures as parameters
assert u.b(7, 4) == '7, 4'
assert u.b(9) == '9, 2'
assert u.c(1,2,3,4,5) == 15 //varying number
of arguments using Object[]
assert u.gcd( 28, 35 ) == 7
u.setCount(91)
assert u.getCount() == 91

```

A class can have both static and instantiable parts by using the static keyword on the definitions that are static and not using it on those that are instantiable:

```
class Dice{
  //here is the static portion of the
class...
  static private count //doesn't need a
value
  static{ println 'First use'; count = 0 }
  static showCount(){ return count }

  //and here is the instantiable portion...
  def lastThrow
  Dice(){ println 'Instance created';
count++ }

  //static portion can be used by
instantiable portion, but not vice versa
  def throww(){
    lastThrow =
1+Math.round(6*Math.random()) //random
integer from 1 to 6
    return lastThrow
  }
}
d1 = new Dice() //'First use' then 'Instance
created' printed
d2 = new Dice() //'Instance created' printed
println "Dice 1:
${(1..20).collect{d1.throww()}}"
println "Dice 2:"
```

```
$(1..20).collect{d2.throww()}"  
println "Dice 1 last throw: $d1.lastThrow,
```

```
dice 2 last throw: $d2.lastThrow"  
println "Number of dice in play:"  
${Dice.showCount()}"
```

A class can have more than one constructor:

```
class A{
  def list= []
  A(){
    list<< "A constructed"
  }
  A(int i){
    this()
    //a constructor can call another
constructor if it's the first statement
    list<< "A constructed with $i"
  }
  A(String s){
    this(5)
    list<< "A constructed with '$s'"
  }
}
def a1= new A()
assert a1.list == [ "A constructed" ]

def a2= new A(7)
assert a2.list.collect{it as String} == [
  "A constructed",
  "A constructed with 7",
]

def a3= new A('bird')
assert a3.list.collect{it as String} == [
  "A constructed",
  "A constructed with 5",
  "A constructed with 'bird'",
]
```

## Categories

When a class has a category method, that is, a static method where the first parameter acts like an instance of the class, we can use an alternative 'category' syntax to call that method:

```
class View{
  def zoom= 1
  def produce(str){ str*zoom }
  static swap(self, that){ //first parameter
acts like instance of the class
  def a= self.zoom
  self.zoom= that.zoom
  that.zoom= a
}
}
def v1= new View(zoom: 5), v2= new
View(zoom: 4)
View.swap( v1, v2 ) //usual syntax
assert v1.zoom == 4 && v2.zoom == 5
use(View){ v1.swap( v2 ) } //alternative
syntax
assert v1.zoom == 5 && v2.zoom == 4
assert v1.produce('a') == 'aaaaa'
```

We can also use category syntax when the category method/s are in a different class:

```

class View{
    static timesCalled= 0 //unrelated static
definition
    def zoom= 1
    def produce(str){ timesCalled++; str*zoom
}
}
class Extra{
    static swap(self, that){ //first parameter
acts like instance of View class
    def a= self.zoom
    self.zoom= that.zoom
    that.zoom= a
    }
}
def v1= new View(zoom: 5), v2= new
View(zoom: 4)
use(Extra){ v1.swap( v2 ) }
    //alternative syntax with category
method in different class
assert v1.zoom == 4 && v2.zoom == 5
assert v1.produce('a') == 'aaaa'

```

Many supplied library classes in Groovy have category methods that can be called using category syntax. (However, most category methods on Numbers, Characters, and Booleans do not work with category syntax in Groovy-1.0)

```
assert String.format('Hello, %1$s.', 42) ==
'Hello, 42.'
use(String){
  assert 'Hello, %1$s.'.format(42) ==
'Hello, 42.'
}
```

Far more common are supplied library classes having category methods in another utility class, eg, List having utilities in Collections:

```
def list= ['a', 7, 'b', 9, 7, 7, 2.4, 7]
Collections.replaceAll( list, 7, 55 )
//normal syntax
assert list == ['a', 55, 'b', 9, 55, 55,
2.4, 55]

list= ['a', 7, 'b', 9, 7, 7, 2.4, 7]
use(Collections){
  list.replaceAll(7, 55) //category syntax
}
assert list == ['a', 55, 'b', 9, 55, 55,
2.4, 55]
```

We can call category methods inside other category methods:

```

class Extras{
    static f(self, n){ "Hello, $n" }
}
class Extras2{
    static g(self, n){
        Extras.f(self, n)
    }
    static h(self, n){
        def ret
            use(Extras){ ret= self.f(n) } //call
Extras.f() as a category method
        ret
    }
}
assert Extras.f(new Extras(), 4) == 'Hello,
4'
assert Extras2.g(new Extras2(), 5) ==
'Hello, 5'
assert Extras2.h(new Extras2(), 6) ==
'Hello, 6'

class A{ }
def a= new A()
use(Extras){
    assert a.f(14) == 'Hello, 14'
}
use(Extras2){
    assert a.g(15) == 'Hello, 15'
    assert a.h(16) == 'Hello, 16' //call
category method within another
}

```

But we can't call category methods inside another category method from the same class:

```
class Extras{
    static f(self, n){ "Hello, $n" }
    static g(self, n){ f(self, n) }
    static h1(self, n){ f(n) } //calling f
without first parameter only valid
                                //when called
within a category method
    static h2(self, n){
        def ret
        use(Extras){
            ret= self.f(n)
        } //class as category within itself only
valid if method wasn't called

//using category syntax
    ret
}
}
assert Extras.f(new Extras(), 4) == 'Hello,
4'
assert Extras.g(new Extras(), 5) == 'Hello,
5'
try{ Extras.h1(new Extras(), 6); assert 0 }
catch(e){ assert e instanceof
MissingMethodException }
assert Extras.h2(new Extras(), 7) == 'Hello,
7'
```

```
class A{ }
def a= new A()
use(Extras){
  assert a.f(14) == 'Hello, 14'
  assert a.g(15) == 'Hello, 15'
  assert a.h1(16) == 'Hello, 16'
  try{ a.h2(17); assert 0 }
```

```
catch(e){ assert e instanceof  
GroovyRuntimeException }  
}
```

A lot of entities in Groovy are classes, not just the explicit ones we've just learnt about. Numbers, lists, sets, maps, strings, patterns, scripts, closures, functions, and expandos are all implemented under the hood as classes. Classes are the building block of Groovy.

### **JN2535-Control**

A Groovy script is a sequence of statements:

```
def a= 1  
assert "a is $a" == 'a is 1'  
def b= 2; assert "b is $b" == 'b is 2'  
    //if two statements on one line,  
separate by semicolons  
def c= 3; ; def d= 4 //empty statement in  
between
```

When defining classes, we can provide 'asType' methods to convert the class into another using the 'as' operator. Classes we've seen in previous tutorials that convert to another using 'as' (eg, Integer, BigDecimal, String) use the 'asType' method under the hood:

```

class A{
  def x
  Object asType(Class c){
    if(c == B) return new B(x:x*3)
  }
}
class B{
  def x
}

```

```

def a= new A(x:3)

```

```

def b1= a.asType(B)
assert b1.class == B && b1.x == 9

```

```

def b2= a as B //more common, shortcut
syntax for asType()
assert b2.class == B && b2.x == 9

```

We can use 'as' to convert a list into a class instance using the list elements as constructor arguments:

```

class A{
  int x,y
  A(x,y){ this.x=x; this.y=y }
  String toString(){ "x: $x; y: $y" }
}
def a= [1,2] as A
assert a.class == A && a.toString() == 'x:
1; y: 2'

```

## **Conditional Statements**

The if and if-else statements let us choose subsequent statements to execute based on a condition:

```

def x= 7
if( x > 4 ){ println 'x is greater than 4' }
//if-statement (no 'else' clause)

if( x > 4 ) println 'x is greater than 4'

//curlies optional if only one statement
//if-else statement...
if( x > 4 ){
    println 'x is greater than 4'
}else{
    println 'x is less than or equal to 4'
}

if( x > 8 ) println 'x is greater than 8'
//again,
curlies optional if only one statement
else println 'x is less than or equal to 8'

//an 'else' clause always belongs to
def result
if( x > 4 )
    if( x > 8 ) result= 'x is greater than 8'
    else result= 'x is less than or equal to
8'

//a single 'else' with two 'if'
clauses belongs to the innermost
assert result == 'x is less than or equal to
8'

```

The meaning of the 'in' operator depends whether its context in the code is conditional or iterative. When in a conditional context, the 'isCase' method of the target is invoked:

```
class A{
  boolean isCase(Object o){
    if(o == 'A') return true
    else return false
  }
}

def a= new A()

assert a.isCase('A')
assert 'A' in a //more common, shortcut
syntax for isCase()

assert ! (a.isCase('Z'))
assert ! ('Z' in a) //more common, shortcut
syntax for isCase()
```

The switch statement inspects an expression and resumes execution from the first matching case-expression, ie, regex matched, list or set or range contained in, class an instance of, or object equal to:

```
def values= [
  'abc': 'abc',
  'xyz': 'list',
  18: 'range',
  31: BigInteger,
  'dream': 'something beginning with dr',
  1.23: 'none',
]
values.each{
```

```

def result
  switch( it.key ){
    case 'abc': //if switched expression
matches case-expression, execute all
                //statements until 'break'
    result= 'abc'
    break
    case [4, 5, 6, 'xyz']:
    result= 'list'
    break
    case 'xyz': //this case is never chosen
because 'xyz' is matched by
                //previous case, then
'break' executed
    result= 'xyz'
    break
    case 12..30:
    result= 'range'
    break
    case Integer:
    result= Integer //because this case
doesn't have a 'break', result
                //overwritten by
BigInteger in next line
    case BigInteger:
    result= BigInteger
    break
    case ~/dr.*/:
    result= 'something beginning with dr'
    break
    case {it instanceof Integer && it>30}:

```

```
//use Closure
    result= 'result is > 30'
    break
default:
    result= 'none'
```

```
}  
  assert result == it.value  
}
```

When we supply our own values in the case-expression, the 'isCase' method is invoked to determine whether or not the switch-expression is matched. If there's no 'isCase' method, the 'equals' method is used to test for equality:

```

class A{
    boolean isCase(Object switchValue){
//'isCase' method used for case-expression
        if(switchValue == 'Hi') return true
        else return false
    }
}
switch( 'Hi' ){
    case new A():
        assert true
        break
    default:
        assert false
}

class B{
    boolean equals(Object switchValue){
//'equals' method used for case-expression
        this.class == switchValue.getClass()
    }
}
switch( new B() ){
    case new B():
        assert true
        break
    default:
        assert false
}

```

The while statement lets us iterate through a block of code:

```
def x= 0, y= 0
while( x < 5 ){
    x++
    y += x
}
assert x == 5 && y == 15

while( x < 10 ) x++ //curlies optional if
only one statement
assert x == 10

while( x < 15 ){ //we can break out of a
while-loop using 'break'
    x++
    if( x == 12 ) break
}
assert x == 12

while( x != 15 && x != 18 ){
    //we can jump to the next iteration of a
while-loop using 'continue'
    x++
    if( x == 15 ){
        x++
        continue
    }
}
assert x == 18
```

We've already seen the 'each' and other related method calls, which emulate the while-statement at a higher level of abstraction, but with some restrictions: the loop variable isn't available outside the loop, no guarantees are made about the order of iteration through the collection, and the 'break', 'continue', and 'return' commands aren't available:

```
int n=1
while( n <= 5 ){
    def y= 0
    (1..n).each{ //loop variable, here 'it',
not available outside loop
        y += it
        //if( y > 8 ) break
            //a syntax error when uncommented:
'break' command not available here
    }
    assert y == (n+1)*(n/2.0) //another way to
add the numbers 1 to n
    n++
}
```

Other method calls that loop are 'times', 'upto', 'downto', and 'step'. Like 'each', they don't allow 'break', 'continue', and 'return' commands, but do make guarantees about the order of iteration:

```

def a= 2
3.times{
  a= a*a
}
assert a == 256

def list= []
1.upto(5){
  list<< it
}
assert list == [1, 2, 3, 4, 5]

list= []
5.3.downto(2.1){ // 'upto', 'downto', and
  'step' also work with decimal numbers
  list<< it
}
assert list == [5.3, 4.3, 3.3, 2.3]

list= []
1.step(9.5, 2.5){
  list<< it
}
assert list == [1, 3.5, 6, 8.5]

```

We can label any statement with a name. Labelling a while loop lets any arbitrarily deep nested statement break out of or continue on from it:

```

yonder: def d= 4
there: {

```

```

def e= 5
here: if( e == 5 ){
    def f= 6
    there: def g= 7 //label can repeat a
previously-used outer label
    }
}
there: def h= 8
        //label can repeat a previously-used
label at same syntactic level

def i=0, j=0
outer: while( i<5 ){ //labelling a while
loop is especially useful...
    j= 0
    i++
    while( j<5 ){
        j++
        if( i==3 && j==2 ) break outer
            //...because we can break out of
a specified labelled while loop
    }
}
assert i==3 && j==2

def outer= 0, inner= 0
outer: while( outer != 5 && outer != 8 ){
        //label can
have same name as any variables
    inner= 0
    outer++

```

```
while( inner<5 ){
    inner++
    if( outer==5 ){
        outer++
        continue outer
        //we can also continue on from
a specified labelled while loop
    }
```

```
    }  
  }  
  assert outer==8
```

#### For-Statements

For-statements are complex yet powerful iterative statements with many possible formats. When 'in' is used in the iterative context of a for-statement, the 'iterator' method of the target is invoked. The 'iterator' method must return an Iterator, defining at least the 'hasNext' and 'next' methods:

```
class CountToFive{  
  def n= 0  
  def iterator(){  
    [ hasNext: { n<5 },  
      next: { n++ },  
    ] as Iterator  
  }  
}  
  
def list= []  
def counter= new CountToFive()  
for(int i in counter){  
  list<< i  
}  
assert list == [0, 1, 2, 3, 4]
```

The for-statement works with many kinds of objects (eg, Collection, array, Map, String, regex, File, Reader, InputStream, etc):

```
def list= []  
for( e in [0, 1, 2, 3, 4] ){ //iterate over  
a list  
  list<< e
```

```
}
assert list == [0, 1, 2, 3, 4]

list= []
for( i in 1..9 ){ //iterate over a range
    list<< i
}
assert list == [1, 2, 3, 4, 5, 6, 7, 8, 9]

list= []
for( e in (3..6).toArray() ){ //over an
array
    list<< e
}
assert list == [3, 4, 5, 6]

list= []
for( e in ['abc':1, 'def':2, 'xyz':3] ){
//over a map
    list<< e.value
}
assert list as Set == [1, 2, 3] as Set

list= []
for( v in [1:'a', 2:'b', 3:'c'].values() ){
//over values in a map
    list<< v
}
assert list as Set == ['a', 'b', 'c'] as Set

list = []
```

```
for( c in "abc" ){ //over the chars in a  
string
```

```
list<< c
}
assert list == ['a', 'b', 'c']
```

We can use 'break' and 'continue' within a for-loop using 'in':

```
def list = []
for( c in 'abc' ){
  list<< c
  if( c == 'b' ) break
}
assert list == ['a', 'b']

list = []
for( c in 'abc' ){
  if( c == 'b' ) continue
  list<< c
}
assert list == ['a', 'c']
```

'each' methods can also be considered as emulating for-loops at a higher level of abstraction, without the guarantees about the order of iteration, and the 'break', 'continue', and 'return' commands being unavailable:

```

def list= []
['a', 'b', 'c'].each{
  list<< it
}
assert list == ['a', 'b', 'c']

//instead of...
list= []
for( item in ['a', 'b', 'c'] ){
  list<< item
}
assert list == ['a', 'b', 'c']

```

Another format for the for-statement is the initializer-condition-incrementer format:

```

def list= []
for(def i=0; i<5; i++){
  //first value an initializer, second a
  condition, third an incrementer
  list<< i
}
assert list == [0, 1, 2, 3, 4]

//equivalent while-statement...
list= []
try{
  def i=0 //initializer
  while( i<5 ){ //condition
    list<< i
    i++ //incrementer
  }
}

```

```
    }  
  }  
  assert list == [0, 1, 2, 3, 4]
```

```
//for-statement with 'break'
```

```
list= []  
for(def i=0; i<5; i++){  
  list<< i  
  if( i == 2 ) break  
}  
assert list == [0, 1, 2]
```

```
//equivalent while-statement with 'break'
```

```
list= []  
try{  
  def i=0  
  while( i<5 ){  
    list<< i  
    if( i == 2 ) break  
    i++  
  }  
}  
assert list == [0, 1, 2]
```

```
//for-statement with 'continue'
```

```
list= []  
for(def i=0; i<5; i++){  
  if( i == 2 ){ continue }  
  list<< i  
}  
assert list == [0, 1, 3, 4]
```

```
//equivalent while-statement with 'continue'  
list= []  
try{  
  def i=0  
  while( i<5){  
    if( i == 2 ){ i++; continue }  
    list<< i  
    i++  
  }  
}
```

```
    }  
  }  
  assert list == [0, 1, 3, 4]
```

We can have more than one initializer, and more than one incrementer:

```
//two initializers and two incrementers...  
def list= []  
for(def i=0; def j=10; i<5; i++; j++){ //the  
middle expression is the condition  
  list<< i + j  
}  
assert list == [10, 12, 14, 16, 18]
```

```
//three initializers and three  
incrementers...  
list= []  
for(def i=0; def j=10; def k=20; i<3; i++;  
j++; k++){  
  list<< i + j + k  
}  
assert list == [30, 33, 36]
```

```
//when there's an even number of  
expressions, the condition is just before  
//the middle...  
list= []  
try{  
  def i=0  
  for(def j=10; i<5; i++; j++){  
    list<< i + j
```

```
    }  
  }  
  assert list == [10, 12, 14, 16, 18]  
  
  //we can force in more initializers than  
  incrementers by using  
  //'null' statements...  
  list= []  
  for(def i=0; def j=10; i<5; i++; null ){
```

```

    list<< i + j
  }
  assert list == [10, 11, 12, 13, 14]

```

#### Operator Overloading

The precedence hierarchy of the operators, some of which we haven't looked at yet, is, from highest to lowest:

```

$(scope escape)
new ()(parentheses)
[](subscripting) ()(method call)
{}(closable block) [](list/map)
. ?. *. (dots)
~ ! $ ()(cast type)
**(power)
++(pre/post) --(pre/post) +(unary)
-(unary)
* / %
+(binary) -(binary)
<< >> >>> .. ..<
< <= > >= instanceof in as
== != <=>
&
^
|
&&
||
?:
= **= *= /= %= += -= <<= >>= >>>= &= ^= |=

```

We've seen how the 'as' operator is mapped to the asType() method, and how the 'in' operator is mapped to the isCase() and iterator() methods. Many more operators have equivalent method names. We've seen how [] subscripting has equivalent methods getAt() and putAt() in the HashMap class. They are also equivalent when we

define such methods on our own classes:

```
class A{  
    int key  
    def value  
    def getAt(int n){ if(key == n) return  
value }  
    void putAt(int n, def o){ key= n; value= o  
}  
}  
def a= new A()  
a[1]= 'abc' //calls putAt()  
assert a[1] == 'abc' //calls getAt()  
assert a[2] == null
```

We've also seen how various operators have equivalent method names in the numerical classes, such as Integer, BigDecimal, float, etc. They, too, are also equivalent when we define such methods on our own classes:

```
class OddNumber{ //only gives odd results to  
operations, adding 1 if necessary  
    int value  
    OddNumber(int n){ value= (n%2)? n: n+1 }  
  
    def power(int n){ value**n }  
    def multiply(int n){ def i= value*n;  
(i%2)? i: i+1 }  
    def div(int n){ int i= value/n; (i%2)? i:  
i+1 }  
    def mod(int n){ int i= value - div(n)*n;  
(i%2)? i: i+1 }  
    def plus(int n){ int i= value + n; (i%2)?
```

```

i: i+1 }
  def minus(int n){ int i= value - n; (i%2)?
i: i+1 }

  def and(int n){ n == value }
  def or(int n){ n == value || (n ==
value-1) }
  def xor(int n) {n == value-1 }

  def leftShift(int n){ value= (n%2)? n: n+1
}
  def rightShift(int n){ (value * 10**n) + 1
}
  def rightShiftUnsigned(int n){ (value *
10**(n*2)) + 1 }

  def next(){ new OddNumber(value + 2) }
  def previous(){ new OddNumber(value - 2) }
}
def e= new OddNumber(6)
assert e.value == 7

assert e**3 == 343 //calls power()
assert e*4 == 29 //calls multiply()
assert e/3 == 3 //calls div()
assert e%3 == -1 //calls mod()
assert e+5 == 13 //calls plus()
assert e-1 == 7 //calls minus()

assert e & 7 //calls and()
assert e | 6 && e | 7 //calls or()

```

```
assert e ^ 6 //calls xor()
```

```
e<< 2 //calls leftShift()
```

```
assert e.value == 3
```

```
assert e>>2 == 301 //calls rightShift()
```

```
assert e>>>2 == 30001 //calls  
rightShiftUnsigned()
```

```
assert (e++).value == 3 //calls next()
```

```
assert e.value == 5
```

```
assert (++e).value == 7
```

```
assert e.value == 7
```

```
assert (e--).value == 7 //calls previous()
```

```
assert e.value == 5
assert (--e).value == 3
assert e.value == 3
```

## JN3015-Types

We can restrict the types of values a variable may hold by specifying some restricting class instead of 'def':

```
import
org.codehaus.groovy.runtime.typehandling.GroovyCastException

def v= 3 //variable v can hold any value
v= 'helicopter'
v= false
v= new StringBuffer()
v= null

int i= 15 //variable i can only hold integer values
i= 'A'
assert i == 65 //'A' casted to its integer value

//unable to cast boolean value to integer
try{ i= false; assert 0 }catch(e){ assert e
in GroovyCastException }

Closure c= {it * 3} //variable c can only hold Closures
try{ c= false; assert 0 }catch(e){ assert e
```

```
in GroovyCastException }
    //unable to cast boolean value to
Closure

StringBuffer s= new StringBuffer('morning')
    //variable s can only hold
StringBuffers
try{ s= { it * 5 }; assert 0 }catch(e){
```

```
assert e in GroovyCastException }
    //unable to cast Closure value to
StringBuffer
```

When we assign values not of a variable's type to the variable, sometimes it may be 'cast' to the type, other times an exception is thrown:

```
import
org.codehaus.groovy.runtime.typehandling.GroovyCastException

int i
i= 45L; assert i == 45i
i= 45.1f; assert i == 45i
try{ i= '42'; assert 0 }catch(e){assert e in GroovyCastException}
try{ i= false; assert 0 }catch(e){assert e in GroovyCastException}

//long similar to int

byte by
by= 200i; assert by == -56

//short similar to byte

float f
f= 123i; assert f == 123.0f
try{ f= '42.1'; assert 0 }catch(e){assert e in GroovyCastException}
```

```
//double similar to float
```

```
BigInteger bi
```

```
bi= 42L; assert bi == 42g
```

```
try{ bi= '421'; assert 0 }catch(e){assert e  
in GroovyCastException}
```

```
BigDecimal bd
```

```
bd= 42.1f; assert bd == 42.1g
```

```
try{ bd= '4.21'; assert 0 }catch(e){assert e  
in GroovyCastException}
```

```
boolean b
```

```
b= 0; assert ! b
```

```
b= 1i; assert b
```

```
b= 1g; assert b
```

```
b= 1.1g; assert b
```

```
b= 1.1f; assert b
```

```
b= ''; assert ! b
```

```
b= 'a'; assert b
```

```
b= 'abc'; assert b
```

```
b= null; assert ! b
```

```
char c
```

```
c= 'a'; assert c == ('a' as char)
```

```
try{ c= 'abc'; assert 0 }catch(e){assert e  
in GroovyCastException}
```

```
String s
```

```
s= 42i; assert s == '42'
```

```
s= 42L; assert s == '42'
```

```
s= 42g; assert s == '42'  
s= 42.1g; assert s == '42.1'  
s= 42.100g; assert s == '42.100'  
s= 42.1f; assert s == '42.1'
```

```
StringBuffer sb  
try{ sb= 'abc'; assert 0 }catch(e){ assert e  
in GroovyCastException }
```

We can statically type Closure parameters. The casting is more restrictive than for assigning to variables:

```
import
org.codehaus.groovy.runtime.typehandling.GroovyCastException

int i
def toTriple= {int n -> n * 3}
i= 5
assert toTriple(5) == 15

//a float is cast to an integer when
assigning to a variable, but not when
//passing as a parameter...
i= 5.0f
try{ toTriple(5.0f); assert 0 }
catch(e){assert e.class in
MissingMethodException}

//a String can't cast to an integer, either
when assigning to a variable or
//passing as a parameter...
try{ i= 'abc'; assert 0 }
catch(e){assert e.class in
GroovyCastException}
try{ toTriple('abc'); assert 0 }
catch(e){assert e.class in
MissingMethodException}
```

We can also statically type the variable-numbered parameters in a closure:

```

def c = { int[] args ->
  args.toList().inject(0){ flo, it-> flo +
it }
}
assert c( 5 ) == 5
assert c( 4, 2, 3 ) == 9
try{ c( 2, 'abc' ); assert 0 }catch(e){
assert e in MissingMethodException }

```

We can statically type function parameters:

```

def f(String s, int i){ ([s]*i).join(',') }
assert f('abc', 3) == 'abc,abc,abc'

def f(int n, int i){ "$n * $i" } //another
function f defined with same
//number of
but different types of parameters
assert f(4, 5) == '4 * 5'
assert f('a', 5) == 'a,a,a,a,a'
//correct function selected based on
parameter types...
try{ f(4, 'x'); assert 0 }catch(e){ assert e
in MissingMethodException }
//...or no method selected

```

We can statically type the return type from a function. Casting a returned value of a different type follows the same rules as for assigning to variables:

```
String f(){ 'abc' }  
assert f() == 'abc'
```

```
int g(){ 2.4f }  
assert g() == 2i
```

We can statically type method parameters just like we do with function parameters, including selecting a method based on its parameter types, for both static methods and instance methods:

```

//static methods...
class A{
    static f(String s, int i){
([s]*i).join(',') }
    static f(int n, int i){ "$n * $i" }
//another method f defined with same
//number
of but different types of parameters
}
assert A.f('abc', 3) == 'abc,abc,abc'
assert A.f(4, 5) == '4 * 5'
assert A.f('a', 5) == 'a,a,a,a,a'
    //correct method selected based on
parameter types...
try{ A.f(4, 'x'); assert 0 }catch(e){ assert
e in MissingMethodException }
    //...or no method selected

//instance methods...
class Counter{
    def count = 0
    def incr( String n ){ count += new
Integer(n) }
    def incr( int n ){ count += n }
}
def c= new Counter(count: 5)
c.incr(3)
c.incr('4')
try{ c.incr(2.5); assert 0 }catch(e){ assert
e in MissingMethodException }
assert c.count == 12

```

We can statically type the return type from a method, just as we can from a function, both static and instance methods:

```
class A{  
    static String f(){ 'abc' }  
    static int g(){ 2.4f }  
    byte h(){ 200i }  
}  
assert A.f() == 'abc'  
assert A.g() == 2i  
assert new A().h() == -56
```

Property getters and setters can accept and return any statically-typed value:

```
class Counter{  
    def count= 0  
    void setCount(int n){ count= n*2 } //set  
the value to twice what's supplied  
    String getCount(){ 'count: '+ count }  
    //return the value as a String with  
'count: ' prepended  
}  
def c= new Counter()  
c.count= 23  
assert c.count == 'count: 46'
```

A list can be cast to a class using that class's constructor:

```
class A{
    int x,y
    A(x,y){ this.x=x; this.y=y } //2-arg
    constructor
    String toString(){ "x: $x; y: $y" }
}
A a
a= [1,2] //2-element list causes 2-arg
constructor of A to be called
assert a.class == A && a.toString() == 'x:
1; y: 2'
```

#### Statically-Typed Arrays

We can statically type an Object array variable:

```

Object[] oa= new Object[2]
assert oa.class in Object[] && oa.size() ==
2 && oa[0,1] == [null, null]

oa= 7 //if we assign another scalar value,
it's wrapped into an array
assert oa.class in Object[] && oa.size() ==
1 && oa[0] == 7

oa= [3, 5] //if we assign another collection
value, it's cast to an array
assert oa.class in Object[] && oa.size() ==
2 && oa[0,1] == [3, 5]

def map= ['a':4, 'b':8, 'c':12]
oa= map
assert oa.class in Object[] && oa.size() ==
3

//it's cast to an array of MapEntry
oa.each{ assert it.key in map.keySet() &&
it.value == map[it.key] }

```

We can statically type a variable not only as an array, but as a certain type of array:

```

int[] ia
ia= 7.5
assert ia.class in int[] && ia.size() == 1
&& ia[0] == 7i
    //assigned value above cast to an
integer array

try{ ia= ['abc', 'def']; assert 0 }catch(e){
assert e in ClassCastException }
    //can't cast Strings to Integers

```

We can instead statically type each array element:

```

def a= new int[3]
assert a[0] == 0 && a[1] == 0 && a[2] == 0
//default value is 0

a[0]= 7.5
assert a[0] == 7i //assigned value in above
line was cast to an integer

try{ a[1]= 'abc'; assert 0 }catch(e){ assert
e in ClassCastException }

//can't cast String to an Integer

```

Statically typing both the variable and each element allows both array assignments and element assignments to be cast or disallowed:

```
int[] ia= new int[3]
ia[0]= 7.5
assert ia[0] == 7i
ia= 7.5
assert ia.class in int[] && ia.size() == 1
&& ia[0] == 7i
```

A multidimensional array type casts its assigned value in various ways:

```
//a scalar value is cascadingly wrapped by
arrays...
Object[][] ia
ia= 7.5
assert ia in Object[][] && ia.size() == 1 &&
       ia[0] in Object[] && ia[0].size() ==
1 &&
       ia[0][0] == 7.5

//a one-dimensional vector value is
array-wrapped at the innermost level...
ia= ['a', 'b', 'c']
assert ia in Object[][] && ia.size() == 3 &&
       ia[0] in Object[] && ia[0].size() ==
1 &&
       ia[0][0] == 'a' && ia[1][0] == 'b' &&
ia[2][0] == 'c'
```

#### Interfaces

Groovy enables a construct known as an interface, which classes can implement. We can test for implemented interfaces with the 'in' operator:

```
class A{} //a standard class definition,  
though without any fields,  
           //properties, or methods  
def a= new A()  
assert a in A  
  
interface X{}  
class B implements X{} //a class can  
implement an interface  
def b= new B()  
assert b in B && b in X  
    //'in' tests for the class and for  
interfaces implemented  
assert ! (a in X)  
  
interface Y{}  
interface Z{}  
class C implements X, Y, Z{} //a class can  
implement more than one interface  
def c= new C()  
assert c in C && c in X && c in Y && c in Z
```

Interfaces can contain method declarations. Each declared method must be defined in implementing classes:

```

interface X{
    String sayPies(int i)
}

class A implements X{
    String sayPies(int n){ "There are $n
pies!" } //sayPies(int) in X defined
    String sayBirds(int n){ "There are $n
birds!" }
}
def a= new A()
assert a.sayPies(24) == 'There are 24 pies!'

//class B implements X{}
    //a compile error when uncommented:
sayPies(int) must be implemented

//these each give a compile error when
uncommented...
//class C implements X{ String sayPies(float
n){ "$n" } } //wrong parameter type
//class D implements X{ Object sayPies(int
n){ "$n" } } //wrong return type

```

An interface can also be composed of other interfaces, using the 'extends' keyword:

```
interface X{
    def x1(int i)
    def x2()
}
interface Y{
    def x1(int i)
    def y()
}
interface Z extends X, Y{ }
    //it's OK if a method, here x1(int), is
in more than one interface

class A implements Z{
    def x1(int i){ i }
    def x2(){ 2 }
    def y(){ 3 }
}
assert new A().x1( 1 ) == 1
```

We can implement an interface with map syntax:

```

interface X{
  int echo(int i)
  def sayTarts(int i)
  String sayPies(int i)
}
def a= [
  echo: {n-> n},
  sayTarts: {n-> "There are $n tarts!"},
  sayPies: {n-> "There are $n pies!" as
String},
  //explicit cast from GString to String
required here
] as X

assert a.echo(12) == 12
assert a.sayTarts(18) == 'There are 18
tarts!'
assert a.sayPies(24) == 'There are 24 pies!'

//when interface has only one method, we
don't need a map, but can assign and
//cast the closure directly...
interface Y{
  def sayCakes(int i)
}
def b= {n-> "There are $n cakes!"} as Y

assert b.sayCakes(36) == 'There are 36
cakes!'

```

Interfaces can also have fields, but their values can't be changed:

```
interface X{
    int status= 1 //constant field on
interface
    int changeCounter()
}
class A implements X{
    int counter= 1 //updateable field on class
itself
    int changeCounter(){ counter++ }
    int changeStatus(){ status++ }
}
def a= new A()
a.changeCounter() //field 'counter' can be
changed...
try{ a.changeStatus(); assert 0 }catch(e){
assert e in IllegalAccessException }
    //...but field 'status' can't
```

#### Static Typing with Interfaces

We can use an interface, instead of a class, to statically type a variable, field, parameter, etc:

```
import  
org.codehaus.groovy.runtime.typehandling.GroovyCastException  
  
interface X{}  
class A implements X{}  
class B{}  
X a  
a= new A()  
try{ a= new B(); assert 0 }catch(e){ assert  
e in GroovyCastException }
```

Groovy supplies many interfaces we can use to statically type variables. Some have no methods, eg, `Serializable`, while others have one or more:

```

class A implements Serializable{}
    //Serializable interface marks class A
via the 'in' operator
assert A in Serializable

//class B implements Closeable{}
    //compile error when uncommented: method
close() must be defined

class C implements Closeable{
    void close(){}
    //Closeable interface signifies that
this close() method is present
}
def c= new C()
if( c in Closeable ) c.close()

```

We've met the Comparator interface in the tutorial on Collections, and the Iterator interface in the tutorial on Control Structures.

Many Groovy classes we've met implement interfaces, which we can use to statically type variables:

```

import
org.codehaus.groovy.runtime.typehandling.GroovyCastException

List list1= new ArrayList(),
    list2= [],
    list3= new LinkedList()
assert list1 in ArrayList &&
    list2 in ArrayList &&

```

**list3 in LinkedList**

```
Set set1= new HashSet(),  
    set2= list1,  
    set3= list3,  
    set4= new TreeSet()  
assert [set1, set2, set3].every{ it in  
HashSet } &&  
    set4 in TreeSet
```

```
SortedSet ss1= new TreeSet(),  
    ss2  
try{ ss2= new HashSet(); assert 0 }catch(e){  
assert e in GroovyCastException }
```

```
Map map1= new HashMap(),  
    map2= new TreeMap(),  
    map3= [:],  
    map4= new LinkedHashMap()  
assert map1 in HashMap &&  
    map2 in TreeMap &&  
    [map3, map4].every{ it in  
LinkedHashMap }
```

```
SortedMap sm1= new TreeMap(),
```

```
sm2
```

```
try{ sm2= new HashMap(); assert 0 }catch(e){  
assert e in GroovyCastException }
```

### JN3025-Inheritance

Groovy enables one class to extend another, just as interfaces can, though classes extend at most one class. We can test for extended classes with the 'in' operator, just like with implemented interfaces:

```
class A{}  
class B extends A{}  
def b= new B()  
assert b in B && b in A  
  
class A1{}  
class A2{}  
//class C extends A1, A2{}  
    //compile error when uncommented: a  
class can extend at most one class
```

Public instance fields, properties, and methods defined on an extended class are also available on the extending class:

```

class A{
    public int prev //field
    int signature //property
    String sayPies(int n){ "There are ${prev=
signature= n} pies!" } //method
}
class B extends A{
    String sayBirds(int n){ "There are $n
birds!" }
}
def b= new B()
assert b.sayBirds(17) == 'There are 17
birds!'
assert b.sayPies(11) == 'There are 11 pies!'
    //method sayPies(int) from A acts as
part of B
assert b.prev == 11 //field 'prev' from A
acts as part of B
b.signature= 19
assert b.signature == 19 //property
'signature' from A acts as part of B
assert b.getSignature() == 19

```

We can use the 'private' and 'protected' modifiers to restrict the visibility of instance methods, properties, and fields:

```

class A{
    //private methods, properties, and fields
are not visible outside the class,
    //even in inheriting classes...
    private int prevPies

```

```
private String sayPies(int n){ "There are
${prevPies= n} pies!" }
```

```
//protected methods, properties, and
fields are visible in inheriting
//classes (and within the same package)...
```

```
protected int prevBeans
protected String sayBeans(int n){ "There
are ${prevBeans= n} beans!" }
}
```

```
class B extends A{
```

```
def testAccesses(){
    assert sayPies(23) == 'There are 23
pies!'
```

```
        //Groovy bug: this private
method shouldn't be visible here
```

```
    try{ prevPies; assert 0 }catch(e){
assert e in MissingPropertyException }
```

```
        //A's private field
'prevPies' not visible here
```

```
    assert sayBeans(29) == 'There are 29
beans!'
```

```
        //A's protected method
visible here in an inheriting class
```

```
    assert prevBeans == 29
```

```
        //A's protected field visible
here in an inheriting class
```

```
    }
```

```
}
```

```
def b= new B()
```

```
assert b.sayPies(11) == 'There are 11 pies!'
        //Groovy bug: this private
method shouldn't be visible here
try{ b.prevPies; assert 0 }catch(e){ assert
e in MissingPropertyException }
        //A's private field
'prevPies' not visible here
```

```
assert b.sayBeans(14) == 'There are 14
beans!'
        //this protected method is visible here
in the same package it's defined in
assert b.prevBeans == 14
        //this protected field is visible here
```

in the same package it's defined in

```
b.testAccesses()
```

Public static fields, properties, and methods are inherited by extending classes:

```
class A{  
    static public int numBananas //field  
    static signature //property  
    static String sayBananas(int n){ //method  
        "There are ${numBananas= signature= n}  
bananas!"  
    }  
}  
class B extends A{  
  
    assert A.sayBananas(23) == 'There are 23  
bananas!' //method call  
    assert A.numBananas == 23 //field access  
    assert A.signature == 23 //property accesses  
    assert A.getSignature() == 23  
  
    assert B.sayBananas(23) == 'There are 23  
bananas!' //method call  
  
    assert B.numBananas == 23 //field access  
    assert B.signature == 23 //property access  
    B.getSignature() == 23 //property access  
    using method syntax
```

We can make static methods, properties, and fields private or protected:

```

class A{
    static private int numBananas= 0
    static private String sayBananas(int n){
        "There are ${numBananas= n} bananas!"
    }
    static protected int numApples= 0
    static protected String sayApples(int n){
        "There are ${numApples= n} apples!"
    }
}
class B extends A{
    static testAccesses(){
        assert sayBananas(37) == 'There are 37
bananas!'
        //numBananas //compile error when
uncommented:
        //A's private field not
visible here

        assert sayApples(29) == 'There are 29
apples!'
        numApples //A's protected field visible
here in an inheriting class
    }
}
assert B.sayBananas(31) == 'There are 31
bananas!'
try{ B.numBananas; assert 0 }catch(e){
assert e in MissingPropertyException }
assert B.sayApples(23) == 'There are 23

```

apples!'

```
assert B.numApples == 23
```

```
B.testAccesses()
```

We can define what's called an "abstract class", a class with only some methods defined, the others being only declarations just like in interfaces. An abstract class and each method declaration in it must be modified with the keyword 'abstract':

```
interface X{  
    def x()  
}  
interface Y{  
    def y()  
}  
abstract class A{  
    def a(){ println 1 } //method definition  
    abstract b()          //declaration of  
method only  
}  
class B extends A implements X, Y{  
    def x(){ println 2 }  
    def y(){ println 3 }  
    def b(){ println 4 } //declared method  
from abstract class A defined here  
}
```

Whether a method is static or not is part of its definition, not its declaration. So interface and abstract methods may not be declared static.

```

interface X{
    def x()
    //static x1() //error when uncommented:
interface methods can not be static
}
interface Y{
    def y()
}
abstract class A{
    static a(){ println 1 }
    abstract b()
    abstract c()
    //abstract static c1()
    //error when uncommented: abstract
methods can not be static
}
class B extends A implements X, Y{
    static x(){ println 2 }
    def y(){ println 3 }
    static b(){ println 4 }
    def c(){ println 5 }
}


```

At the other end from abstract classes and methods are "final classes" and "final methods". A final class may not be extended; a final method may not be overridden:

```

class A{
  final a(){ 11 }
  def b(){ 12 }
}
final class B extends A{
  //def a(){ 15 } //compile error when
uncommented: can not override final A.a()
  def b(){ 16 }
}
//class C extends B{} //compile error when
uncommented: can not extend final C

```

#### Constructors

Just as a class's constructor can call another constructor at the beginning of its code, so also it can call a constructor on the superclass at the beginning of its code:

```

class A{
  def list= []
  A(){
    list<< "A constructed"
  }
  A(int i){
    this()
    list<< "A constructed with $i"
  }
}
class B extends A{
  B(){
    list<< "B constructed"
  }
}

```

```
B(String s){
    super(5) //a constructor can call its
superclass's constructor if it's
        //the first statement
    list<< "B constructed with '$s'"
}
}

def b1= new B('kea')
assert b1.list.collect{it as String} == [
    "A constructed",
    "A constructed with 5",
    "B constructed with 'kea'",
]
def b2= new B()
assert b2.list == [
    "A constructed",
    //default parameterless constructor
```

```
called if super() not called
    "B constructed",
]
```

#### Using Classes by Extending Them

Some classes supplied with Groovy are intended to be extended to be used. For example, `FilterInputStream`, `FilterOutputStream`, `FilterReader`, and `FilterWriter`:

```
//When not extended, FilterOutputStream
simply passes its method calls to the
//wrapped stream...
try{
    def fos= new FilterOutputStream(new
FileOutputStream('abc.txt'))
    fos.write(33i)
    fos.write([34,35,36] as byte[])
    fos.write([34,35,36,37,38,39,40] as
byte[], 3, 2)
    fos.close()
    def fis= new FilterInputStream(new
FileInputStream('abc.txt'))
    def ba= new byte[6]
    fis.read(ba)
    assert ba.toList() == [33,34,35,36,37,38]
}

//We can extend FilterOutputStream to
provide the logic for the filter...
class EvenNumberOutputStream extends
FilterOutputStream{
    EvenNumberOutputStream(OutputStream out){
```

```

    super(out)
}
def write(int i){
    if(i%2 == 0) super.write(i) //call
method of same name in the super-class
}
def write(byte[] ba){
    super.write( ba.toList().findAll{ it%2
== 0 } as byte[] )
}
def write(byte[] ba, int start, int size){
    this.write( ba[start..<(start+size)] as
byte[] )
        //another way to call method of same
name in same class definition
}
}
try{ //...then call the methods...
    def fos= new EvenNumberOutputStream(new
FileOutputStream('abc.txt'))
    fos.write(33i)
    fos.write([34,35,36] as byte[])
    fos.write([34,35,36,37,38,39,40] as
byte[], 3, 2)
    fos.close()
    def fis= new FilterInputStream(new
FileInputStream('abc.txt'))
    def ba= new byte[6]

```

```
fis.read(ba)
assert ba.toList() == [34,36,38,0,0,0]
}
```

We can similarly extend `FilterInputStream`, `FilterReader`, and `FilterWriter`.

#### The Object Hierarchy

All classes are arranged in a hierarchy with `java.lang.Object` as the root. Here are those we've met so far; those labelled as such are abstract and final classes:

```
java.lang.Object
  java.lang.Boolean (final)
  java.lang.Character (final)
  java.lang.Number (abstract)
    java.lang.Integer (final)
    java.lang.Long (final)
    java.math.BigInteger
    java.math.BigDecimal
    java.lang.Short (final)
    java.lang.Byte (final)
    java.lang.Float (final)
    java.lang.Double (final)
  java.math.MathContext (final)
  java.util.Random
  java.util.Date
  java.util.TimeZone (abstract)
    java.util.SimpleTimeZone
  java.util.Calendar (abstract)
    java.util.GregorianCalendar
  groovy.time.BaseDuration (abstract)
    groovy.time.Duration
    groovy.time.TimeDuration
```

**groovy.time.DatumDependentDuration**

**groovy.time.TimeDatumDependentDuration**

**java.util.AbstractCollection (abstract)**

**java.util.ArrayList (abstract)**

**java.util.ArrayList**

**groovy.lang.Sequence**

**groovy.lang.IntRange**

**groovy.lang.ObjectRange**

**java.util.AbstractSet (abstract)**

**java.util.HashSet**

**java.util.TreeSet**

**java.util.AbstractMap (abstract)**

**java.HashMap**

**java.util.LinkedHashMap**

**groovy.lang.SpreadMap**

**java.TreeMap**

**java.util.Collections**

**java.lang.String (final)**

**java.lang.StringBuffer (final)**

**java.util.regex.Pattern (final)**

**java.util.regex.Matcher (final)**

**groovy.lang.GroovyObjectSupport**

**(abstract)**

**groovy.lang.Binding**

**groovy.lang.Closure (abstract)**

**groovy.lang.GString (abstract)**

**groovy.util.Expando**

**java.text.Format (abstract)**

**java.text.NumberFormat (abstract)**

**java.text.DecimalFormat**

```
    java.text.DateFormat (abstract)
        java.text.SimpleDateFormat
java.text.DecimalFormatSymbols
java.text.DateFormatSymbols
java.io.File
java.io.InputStream (abstract)
    java.io.ByteArrayInputStream
    java.io.FileInputStream
    java.io.FilterInputStream
        java.io.BufferedInputStream
        java.io.DataInputStream
        java.io.LineNumberInputStream
        java.io.PushbackInputStream
    java.io.SequenceInputStream
    java.io.StringBufferInputStream
java.io.OutputStream (abstract)
    java.io.ByteArrayOutputStream
    java.io.FileOutputStream
    java.io.FilterOutputStream
        java.io.BufferedOutputStream
        java.io.DataOutputStream
        java.io.PrintStream
java.io.Reader (abstract)
    java.io.BufferedReader
        java.io.LineNumberReader
    java.io.CharArrayReader
    java.io.FilterReader (abstract)
        java.io.PushbackReader
    java.io.InputStreamReader
        java.io.FileReader
    java.io.StringReader
```

```
java.io.Writer (abstract)
  java.io.BufferedWriter
  java.io.CharArrayWriter
  java.io.FilterWriter (abstract)
  java.io.OutputStreamWriter
```

```
java.io.FileWriter
java.io.PrintWriter
java.io.StringWriter
```

### JN3035-Exceptions

Exceptions and Errors are together known as Throwables. The Throwables are positioned like so in the Object hierarchy:

```
java.lang.Object
  java.lang.Throwable
    java.lang.Error
    java.lang.Exception
```

Errors are fatalities that we would normally want to cause a program failure, while Exceptions are events that we would normally want to handle in our program. An example of using them with a try-catch statement, a 'try' clause followed by a 'catch' clause:

```
//assert 1 == 0 //AssertionError when
uncommented

//try{ assert 1 == 0 }catch(e){}
    //AssertionError when uncommented:
Exceptions, not Errors, are caught here

try{
    assert 1 == 0
}catch(Error e){}
    //by specifying Error, prevents bad
assertion from causing program failure

try{
    assert 1 == 0
}catch(Throwable e){} //specifying Throwable
also prevents program failure

//try{ assert 1 == 0 }catch(Object o){}
    //compile error when uncommented:
    //only Throwables and its subclasses may
be caught
```

A common idiom for asserting for exceptions is:

```
try{
    'moo'.toLong() //this will generate an
exception
    assert false //asserting that this point
should never be reached
}catch(e){
    assert e in NumberFormatException
}
```

Some common exceptions associated with Groovy:

```
assert new java.lang.ArithmeticException()
assert new
java.lang.ArrayIndexOutOfBoundsException()
assert new java.lang.NullPointerException()
assert new java.io.IOException()
```

We can put code within a 'finally' clause following a matching 'try' clause, so that regardless of whether the code in the 'try' clause throws an exception, the code in the finally clause will always execute:

```
def z
try{
  def i= 7, j= 0
  try{
    def k= i / j
    assert false //never reached due to
Exception in previous line
  }finally{
    z= 'reached here' //always executed even
if Exception thrown
  }
}catch(e){
  assert e in ArithmeticException
  assert z == 'reached here'
}
```

We can attach more than one 'catch' clause to a 'try' clause, and attach a 'finally' clause also:

```
class E1 extends Exception{} //we can define
our own exceptions
class E2 extends Exception{}
class E3 extends Exception{}

try{
    def z

    //multi-catch try-block with
finally-clause...
    try{
        throw new E2()
        assert false
    }catch(E1 e){
        assert false
    }catch(E2 e){
        z= 'reached here'
        throw new E3() //uncaught exception
because only one catch clause executed
    }catch(E3 e){
        assert false //never reached
    }finally{
        assert z == 'reached here'
        throw new E1()
        assert false
    }

}

}catch(E1 e){} //catches exception thrown in
embedded finally clause
```

With the multi catch block (since Groovy 2.0), we're able to define several exceptions to be catch and treated by the same catch block.

```
try {
    /* ... */
} catch (IOException | NullPointerException e)
{
    /* one block to handle 2 exceptions */
}
```

An exception will ripple up through the nested blocks, executing only code in 'finally' clauses, until caught, or the thread terminates.

```
class MyException extends Exception{}
def z
try{
    try{
        throw new MyException()
        assert false
    }
} catch (e) {
    assert e in MyException
    z= 'been here'
}
assert z == 'been here'
```

Exceptions will also ripple through function and method invocations

```
class MyException extends Exception{}

def z= []
```

```
def met(){
  throw new MyException()
}
try{ met(); assert false }
catch(e){assert e in MyException; z <<
'function'}
```

```
class M{
  def m(){ throw new MyException() }
}
try{ new M().m(); assert false }
catch(e){assert e in MyException; z <<
'method' }
```

```
def c= { throw new MyException() }
try{ c(); assert false }
catch(e){assert e in MyException; z <<
'closure'}
```

```
assert z == ['function', 'method',
'closure']
```

```
//Method embedded in closure...
def z2
def d= { new M().m(); assert false }
try{ d(); assert false }
```

```
catch(e){assert e in MyException; z2=
'closure d'}
assert z2 == 'closure d'
```

We can mark a function or method indicating what type of Exception it might throw. This is a useful documentation feature:

```
class MyException extends Exception{
def z= []

def met() throws MyException{ // 'function
met() may throw MyException'
    throw new MyException()
}
try{ met(); assert false }
catch(e){assert e in MyException; z <<
'function' }

class M{
    def m() throws MyException{ // 'method m()
of class M may throw MyException'
        throw new MyException()
    }
}
try{ new M().m(); assert false }
catch(e){assert e in MyException; z <<
'method' }

assert z == ['function', 'method']
```

We can use the ProxyMetaClass to intercept methods in a class within a selected block for the current thread.

#### Interceptors with ProxyMetaClass

By using ProxyMetaClass, we can attach an interceptor to a class for a block of code. The Groovy-supplied Interceptor interface has three methods. The beforeInvoke() method specifies code to be executed before the intercepted method, the doInvoke() indicates whether to execute the intercepted method, and afterInvoke() executes after the intercepted method finishes, or after a false-returning doInvoke(). The result parameter passed to afterInvoke() is the result of executing the method, or what was returned from beforeInvoke() if the intercepted method wasn't executed. What afterInvoke() returns is returned from the method call in the main flow of the program.

```
class MyClass{
    public MyClass(String s){ println
"constructing $s" }
    public String sayHello(String name){
        println "saying hello to $name"
        "Hello " + name //return this value
    }
}

class MyInterceptor implements Interceptor{
    Object beforeInvoke(Object object, String
methodName, Object[] arguments){
        println " BEFORE $object .$methodName
$argsuments"
        if( methodName == 'sayHello' )
arguments[0] += ' and family'

//we can change the arguments
        null //value returned here isn't
actually used anywhere else
    }
    boolean doInvoke(){ true } //whether or
not to invoke the intercepted
```

```
                //method with  
beforeInvoke's copy of arguments
```

```
    Object afterInvoke(Object object, String  
methodName, Object[] arguments,  
                        Object result){  
        println "    AFTER $object .$methodName  
$arguments: $result"  
        if( methodName == 'sayHello' ) result=  
(result as String) + ' and in-laws'  
  
//we can change the returned value  
    result  
    }  
}
```

```
def proxy= ProxyMetaClass.getInstance(  
MyClass )
```

```
//create proxy metaclass for MyClass  
proxy.interceptor= new MyInterceptor()  
                //attach new  
interceptor to MyClass's proxy metaclass  
proxy.use{  
    def invoice= new MyClass('trade')  
    println invoice.sayHello('Ms Pearl')  
}
```

```
/*example output:
```

```
    BEFORE class MyClass .ctor {"trade"}  
constructing trade
```

```
AFTER class MyClass .ctor {"trade"}:
MyClass@1d63e39
  BEFORE MyClass@1d63e39 .sayHello {"Ms
  Pearl"}
saying hello to Ms Pearl and family
  AFTER MyClass@1d63e39 .sayHello {"Ms Pearl
```

```
and family"}: Hello Ms Pearl and family
Hello Ms Pearl and family and in-laws
*/
```

We can invoke a different method instead of the one called:

```
class MyClass{
    public String sayHello(String name){
        println "saying hello to $name"
        return "Hello " + name
    }
    public String sayGoodbye(String name){
        println "saying goodbye to $name"
        return "Goodbye " + name
    }
}

class MyInterceptor implements Interceptor{
    def toInvoke= true
        //so we can change whether or
not to invoke the original method
    def resultFromSayGoodBye

    Object beforeInvoke(Object object, String
methodName, Object[] arguments){
        if( object instanceof MyClass &&
methodName == 'sayHello' ){
            resultFromSayGoodBye=
object.sayGoodbye(arguments[0])

        //so we can invoke a different method
```

```
        toInvoke= false //don't invoke
sayHello
    }
}
boolean doInvoke(){ toInvoke }
```

```
Object afterInvoke(Object object, String
methodName, Object[] arguments,
                    Object result){
    if( object instanceof MyClass &&
methodName == 'sayHello' ){
        toInvoke= true
        result= resultFromSayGoodBye
    }
    result
}
}
```

```
//a utility to match up class, interceptor,
and code...
```

```
def useInterceptor= { Class theClass, Class
theInterceptor, Closure theCode->
```

```
    def proxy= ProxyMetaClass.getInstance(
theClass )
```

```
    def interceptor=
```

```
theInterceptor.newInstance()
```

```
        //must use dynamic
```

```
constructor here because class not yet known
```

```
    proxy.interceptor= interceptor
```

```
    proxy.use( theCode )
```

```
}
```

```
useInterceptor( MyClass, MyInterceptor ){  
    println new MyClass().sayHello('Ms Pearl')  
}
```

**/\*output:**

```
saying goodbye to Ms Pearl  
Goodbye Ms Pearl  
*/
```

We can even use interceptors on predefined Java classes:



```

class MyInterceptor implements Interceptor{
    Object beforeInvoke(Object object, String
methodName, Object[] arguments){
        null
    }
    boolean doInvoke(){ true }
    Object afterInvoke(Object object, String
methodName, Object[] arguments,
                        Object result){
        if( object instanceof ArrayList &&
methodName == 'size' ){
            result = (result as Integer) + 10
//add 10 to size of ArrayLists
        }
        result
    }
}

def useInterceptor= { Class theClass, Class
theInterceptor, Closure theCode->
    def proxy= ProxyMetaClass.getInstance(
theClass )
    def interceptor=
theInterceptor.newInstance()
    proxy.interceptor= interceptor
    proxy.use( theCode )
}

useInterceptor( ArrayList, MyInterceptor ){
    assert ['a', 'b', 'c'].size() == 13
}

```

We can prevent methods being intercepted inside the interceptor by using special & notation:

```
class MyInterceptor implements Interceptor{
    Object beforeInvoke( Object object, String
methodName, Object[] arguments ){
        null
    }
    boolean doInvoke(){ true }

    Object afterInvoke( Object object, String
methodName, Object[] arguments,
                        Object result ){
        if( object instanceof ArrayList &&
methodName == 'size' ){
            result = (result as Integer) +
[1,2,3,4,5,6,7,8,9,10].&size()
                // & before method name
prevents re-interception of method
        }
        result
    }
}

def useInterceptor= { Class theClass, Class
theInterceptor, Closure theCode->
    def proxy= ProxyMetaClass.getInstance(
theClass )
    def interceptor=
theInterceptor.newInstance()
    proxy.interceptor= interceptor
```

```
proxy.use( theCode )  
}
```

```
useInterceptor( ArrayList, MyInterceptor ){
    assert ['a', 'b', 'c'].size() == 13
}
```

Like categories, interceptors are only valid for a certain block in the current thread. We can also combine categories with interceptors in various ways, also only valid in the current thread:

```
class MyCategory{
    static String categorize( String s ){
        "categorized: $s" }
}

class StringInterceptor implements
Interceptor{
    Object beforeInvoke(Object object, String
methodName, Object[] arguments){
        if( object instanceof String )
            use(MyCategory){
                assert object.&categorize() ==
"categorized: $object"
            }
        null
    }
    boolean doInvoke(){ true }
    Object afterInvoke(Object object, String
methodName, Object[] arguments,
                        Object result){
        if( object instanceof String )
            result= "intercepted: $result"
        result
    }
}
```

```

}

def useInterceptor= { Class theClass, Class
theInterceptor, Closure theCode->
  def proxy= ProxyMetaClass.getInstance(
theClass )
  def interceptor=
theInterceptor.newInstance()
  proxy.interceptor= interceptor
  proxy.use( theCode )
}

useInterceptor( String, StringInterceptor ){
  assert new String('silver').toString() ==
'intercepted: silver'

  use(MyCategory){
    assert new String('golden').categorize()
==
'intercepted: categorized: golden'
  }

  Thread.start{ //no interception in spawned
thread...
    use(MyCategory){
      assert new
String('bronze').categorize() ==
'categorized: bronze'

```

```
    }  
  }  
}
```

#### Unintercepted Interceptors

The special & notation for bypassing interceptors handles simple code, but for more complex code we often need our own UninterceptedInterceptor:

```
abstract class UninterceptedInterceptor  
implements Interceptor{  
    def proxy= null //we need to know the  
proxy...  
  
    abstract Object doBefore( Object object,  
String methodName,  
                                Object[]  
arguments )  
  
    public Object beforeInvoke( Object object,  
String methodName,  
                                Object[]  
arguments ){  
  
        proxy.interceptor= null //...so we can  
turn off interception...  
        def result  
        try{  
            result= doBefore(object, methodName,  
arguments)  
        }catch(Exception e){  
            throw e  
        }  
    }  
}
```

```

        }finally{
            proxy.interceptor= this //...and turn
interception back on
        }
        result
    }
    abstract boolean doInvoke()

    abstract Object doAfter( Object object,
String methodName, Object[] arguments,
                            Object result )

    public Object afterInvoke( Object object,
String methodName,
                                Object[]
arguments, Object result ){
        proxy.interceptor= null //turn off
interception
        try{
            result= doAfter(object, methodName,
arguments, result)
        }catch(Exception e){
            throw e
        }finally{
            proxy.interceptor= this //turn
interception back on
        }
        result
    }
}

```

```

class MyInterceptor extends
UninterceptedInterceptor{
    Object doBefore( Object object, String
methodName, Object[] arguments ){
        null
    }
    boolean doInvoke(){ true }

    Object doAfter( Object object, String
methodName, Object[] arguments,
                    Object result ){
        if( object instanceof ArrayList &&
methodName == 'size' ){
            result = (result as Integer) +
[1,2,3,4,5,6,7,8,9,10].size()
                //call ArrayList size()
method here without stack overflow
        }
        result
    }
}

```

```

def useInterceptor= { Class theClass, Class
theInterceptor, Closure theCode->
    def proxy= ProxyMetaClass.getInstance(
theClass )
    def interceptor=
theInterceptor.newInstance()
    proxy.interceptor= interceptor
    interceptor.proxy= proxy
        //we must now store a proxy

```

```
reference in the interceptor  
    proxy.use( theCode )  
}
```

```
useInterceptor( ArrayList, MyInterceptor ){
    assert ['a', 'b', 'c'].size() == 13
}
```

#### Intercepting many classes in one block

Often, we want to intercept more than one class in one block. This example is of an aliasing interceptor, which disables some English-language names for selected classes, and replaces them with Spanish-language names. We re-use the `UninterceptedInterceptor` class and `useInterceptor` utility from the previous example.

```
import
org.codehaus.groovy.runtime.InvokerHelper

abstract class AliasInterceptor extends
UninterceptedInterceptor{
    protected aliases= [:]

    private toReturn= null, toThrow= false,
toInvoke= false

    Object doBefore( Object object, String
methodName, Object[] arguments ){
        if( methodName in aliases.keySet() )
            toReturn= InvokerHelper.invokeMethod(
object, aliases[methodName],
arguments )

        //use Spanish names instead
        else if( methodName in aliases.values()
) toThrow= true

        //disable the English names
```

```

    else toInvoke= true //run other methods
unchanged
    null
}
Object doAfter( Object object, String
methodName, Object[] arguments,
                Object result ){
    if( toReturn != null ){
        result= toReturn
        toReturn= null
    }else if( toThrow ){
        toThrow= false
        throw new MissingMethodException(
methodName, object.getClass(),

arguments )
    }else toInvoke= false
    result
}
boolean doInvoke(){ toInvoke }
}

```

```

class ArrayListAliasInterceptor extends
AliasInterceptor{
    {aliases.putAll( [tamano:'size',
todos:'each' ] )} //Spanish aliases
}

```

```

class HashMapAliasInterceptor extends
AliasInterceptor{
    {aliases.putAll( [tamano:'size',

```

```
todos:'each' ] )}  
}
```

```
class LinkedHashMapAliasInterceptor extends  
AliasInterceptor{
```

```
{aliases.putAll( [tamano:'size',
todos:'each' ] )}
}
```

We call the code like so:

```
def useAliasing= { Closure c->
  useInterceptor(ArrayList,
ArrayListAliasInterceptor){
  useInterceptor(HashMap,
HashMapAliasInterceptor){
  useInterceptor(LinkedHashMap,
LinkedHashMapAliasInterceptor){
    c()
  }
}
}
}

useAliasing{
  def a= [1, 3, 5, 7, 9]
  println 'size: '+ a.tamano()
  //Spanish 'tamano' is
an alias for the 'size' method
  try{ println a.size(); assert 0 }
  catch(e){ assert e instanceof
MissingMethodException }
  //English 'size'
method disabled
  a.todos{ println 'item: '+ it }
  println ''
}
```

```
def b= [a:1, c:3, e:5, g:7]
println 'size: '+ b.tamano()
try{ println b.size(); assert 0 }
catch(e){ assert e instanceof
MissingMethodException }
b.todos{ println 'item: '+ it }
println ''

def c= new LinkedHashMap( [e:5, g:7, i:9]
)
println 'size: '+ c.tamano()
try{ println c.size(); assert 0 }
catch(e){ assert e instanceof
```

```

MissingMethodException }
  c.todos{ println 'item: '+ it }
}

```

We can put the cascadingly indented code into a list to make it neater by defining a utility category method on the List class.

```

class Extras{
  static closureInject(List self, Closure
base){
  def z= []
  self.eachWithIndex{ it, i-> z<< {-> it(
z[i+1] )} }
  z<< base
  z[0]()
}
}

use(Extras){
  [ {c-> useInterceptor(ArrayList,
ArrayListAliasInterceptor){ c() }},
  {c-> useInterceptor(HashMap,
HashMapAliasInterceptor){ c() }},
  {c-> useInterceptor(LinkedHashMap,
LinkedHashMapAliasInterceptor){ c() }},

].closureInject{
  def a= [1, 3, 5, 7, 9],
  b= [a:1, c:3, e:5, g:7],
  c= new LinkedHashMap( [e:5, g:7,
i:9] )
}
}

```

```
println 'size: '+ a.tamano()  
try{ println a.size(); assert 0 }  
catch(e){ assert e instanceof  
MissingMethodException }  
a.todos{ println 'item: '+ it }  
println ''
```

```
println 'size: '+ b.tamano()  
try{ println b.size(); assert 0 }  
catch(e){ assert e instanceof  
MissingMethodException }  
b.todos{ println 'item: '+ it }  
println ''
```

```
println 'size: '+ c.tamano()  
try{ println c.size(); assert 0 }  
catch(e){ assert e instanceof  
MissingMethodException }
```

```
        c.todos{ println 'item: '+ it }
    }
}
```

#### Our own ProxyMetaClass

We can define our own proxy meta-classes. One case for which we'd do so is to implement our own style of interceptors, here, an around-interceptor:

```
import
org.codehaus.groovy.runtime.InvokerHelper

public class MyProxyMetaClass extends
MetaClassImpl{
    protected adaptee= null
    def interceptor= null
    MyProxyMetaClass(MetaClassRegistry
registry, Class theClass,
                    MetaClass adaptee){
        super(registry, theClass);
    this.adaptee = adaptee
    }
    static getInstance(Class theClass){
        def metaRegistry =
InvokerHelper.getInstance().getMetaRegistry(
)
        new MyProxyMetaClass(metaRegistry,
theClass,
metaRegistry.getMetaClass(theClass) )
    }
    void use(Closure closure){
```

```

        registry.setMetaClass(theClass,
this)
        try{ closure.call() }
        finally{
registry.setMetaClass(theClass, adaptee) }
    }
    void use(GroovyObject object, Closure
closure){
        object.setMetaClass(this)
        try{ closure.call() }
        finally{
object.setMetaClass(adaptee) }
    }
    Object invokeMethod(final Object object,
final String methodName,
                        final Object[]
arguments){
        doCall(object, methodName,
arguments,
                {
adaptee.invokeMethod(object, methodName,
arguments) } )
    }
    Object invokeStaticMethod(final Object
object, final String methodName,
                                final Object[]
arguments){
        doCall(object, methodName,
arguments,
                {
adaptee.invokeStaticMethod(object,

```

```

methodName, arguments) } )
    }
    Object invokeConstructor(final Object[]
arguments){
        doCall(theClass, "ctor", arguments,
            {
adaptee.invokeConstructor(arguments) } )
        }
    Object invokeConstructorAt(final Class
at, final Object[] arguments){
        doCall(theClass, "ctor", arguments,
            {
adaptee.invokeConstructorAt(at, arguments) }
        )
    }
    private Object doCall(Object object,
String methodName, Object[] arguments,
        Closure
howToInvoke){
        if (null == interceptor){ return
howToInvoke.call() }
        interceptor.aroundInvoke(object,
methodName, arguments, howToInvoke)
    }
}

interface AroundInterceptor{
    Object aroundInvoke(Object object, String

```

```
methodName, Object[] arguments,  
                Closure proceed)  
}
```

We can then run our code:

```
class MyInterceptor implements  
AroundInterceptor{  
    Object aroundInvoke(Object object, String  
methodName, Object[] arguments,  
                        Closure proceed){  
        println " BEFORE $object .$methodName  
$arguments"  
        def result= proceed()  
        println " AFTER $object .$methodName  
$arguments: $result"  
        result  
    }  
}  
  
class MyClass{  
    void sayHi(){ System.out.println 'hi' }  
}  
  
def interceptor= new MyInterceptor()  
def proxy= MyProxyMetaClass.getInstance(  
MyClass )  
proxy.use{  
    proxy.interceptor= interceptor  
    new MyClass().sayHi()  
}
```

```
/*outputs:  
  BEFORE class MyClass .ctor {}  
  AFTER class MyClass .ctor {}:  
MyClass@1f5d386  
  BEFORE MyClass@1f5d386 .sayHi {}
```

```
hi
  AFTER MyClass@1f5d386 .sayHi {}: null
*/
```

Using many Interceptors with our own ProxyMetaClass

We can only use one interceptor with the ProxyMetaClass supplied by Groovy, so we need to provide our own when attaching more than one interceptor to a class:

```
import
org.codehaus.groovy.runtime.InvokerHelper

public class MultiInterceptorProxyMetaClass
extends MetaClassImpl{
  protected adaptee= null
  def interceptors= [] //reference a list of
interceptors, instead of just one

  MultiInterceptorProxyMetaClass(
MetaClassRegistry registry, Class theClass,
MetaClass
adaptee ){
  super(registry, theClass)
  this.adaptee = adaptee
  if( null == adaptee )
    throw new IllegalArgumentException(
"adaptee must not be null" )
  }
  static getInstance(Class theClass){
  def metaRegistry=
InvokerHelper.getInstance().getMetaRegistry(
)
```



```

        doCall(object, methodName, arguments,
            {
    adaptee.invokeStaticMethod(object,
methodName, arguments) } )
    }
    Object invokeConstructor(final Object[]
arguments){
        doCall(theClass, "ctor", arguments,
            {
    adaptee.invokeConstructor(arguments) } )
    }
    public Object invokeConstructorAt(final
Class at, final Object[] arguments){
        doCall(theClass, "ctor", arguments,
            { adaptee.invokeConstructorAt(at,
arguments) } )
    }
    private Object doCall( Object object,
String methodName, Object[] arguments,
                        Closure howToInvoke
){
        if( interceptors == [] ){ return
howToInvoke.call() }
        def result
        interceptors.each{ //different logic to
cater for all the interceptors

            result= it.beforeInvoke(object,
methodName, arguments)
            if( it.doInvoke() ){ result=
howToInvoke.call() }

```

```
        it.afterInvoke(object, methodName,  
arguments, result)  
    }
```

```
        result
    }
}
```

Using a MultiInterceptorProxyMetaClass for the Observer pattern

A common design pattern is the Observer pattern. Using interceptors, we can abstract the observation code into its own class, the ObserverProtocol, which can be used by subclasses. It enables us to add and remove observing objects for an observed object. We use method interception to decouple the observing and observed objects from the observation relationship itself.

```
abstract class ObserverProtocol implements
Interceptor{
    private perSubjectObservers

    protected getObservers( subject ){
        if( perSubjectObservers == null )
perSubjectObservers= [:]
        def observers= perSubjectObservers[
subject ]
        if( observers == null ){
            observers= []
            perSubjectObservers[ subject ]=
observers
        }
        observers
    }

    public void addObserver( subject, observer
){
        getObservers(subject) << observer
    }
    public void removeObserver( subject,
```

```
observer ){  
    getObservers (subject) .remove(observer)  
}
```

```
abstract Object beforeInvoke( Object  
object, String methodName,  
                                Object[]  
arguments )
```

```
abstract boolean doInvoke()
```

```
abstract Object afterInvoke( Object  
object, String methodName,
```

**Object[]**

```
arguments, Object result )  
}
```

We can extend this ObserverProtocol with domain-specific observers. The example is a Groovy rewrite of one first implemented in AspectJ by Jan Hannemann and Gregor Kiczales.

```
public class Screen{ //class to be observed  
  def name  
  public Screen( String s ){  
    this.name= s  
  }  
  public void display( String s ){  
    println(this.name + ": " + s)  
  }  
}
```

```
public class Point{ //class to be observed  
  def x, y, color  
  public Point( int x, int y, Color color ){  
    this.x=x  
    this.y=y  
    this.color=color  
  }  
}
```

```
class ColorObserver extends  
ObserverProtocol{  
  Object beforeInvoke( Object object, String  
methodName, Object[] arguments ){  
    null
```

```

    }
    boolean doInvoke(){ true }
    Object afterInvoke( Object object, String
methodName, Object[] arguments,
                        Object result ){
        if( object instanceof Point &&
methodName == 'setColor' ){
            getObservers(object).each{
                it.display("Screen updated (point
subject changed color).")
            }
        }
        result
    }
}

```

```

class CoordinateObserver extends
ObserverProtocol{
    Object beforeInvoke( Object object, String
methodName, Object[] arguments ){
        null
    }
    boolean doInvoke(){ true }
    Object afterInvoke( Object object, String
methodName, Object[] arguments,
                        Object result ){
        if( object instanceof Point && ['setX',
'setY'].contains(methodName) ){
            getObservers(object).each{
                it.display("Screen updated (point
subject changed coordinates).")
            }
        }
    }
}

```

```
    }  
  }  
  result  
}  
}
```

```
class ScreenObserver extends  
ObserverProtocol{  
  Object beforeInvoke( Object object, String  
methodName, Object[] arguments ){  
    null  
  }  
  boolean doInvoke(){ true }  
  Object afterInvoke( Object object, String  
methodName, Object[] arguments,  
                      Object result ){  
    if( object instanceof Screen &&  
methodName == 'display' ){  
      getObservers(object).each{  
        it.display("Screen updated (screen  
subject changed message).")  
      }  
    }  
  }  
}
```

```
    result
  }
}
```

Now we run the program. It first creates five Screen objects (s1, s2, s3, s4, and s5) and one point object, then sets up some observing relationships (namely, s1 and s2 will observe color changes to the point, s3 and s4 will observe coordinate changes to the point, and s5 will observe s2's and s4's display method), and finally, make changes to the point, first, the color, then its x-coordinate. The color change triggers s1 and s2 to each print an appropriate message. s2's message triggers its observer s5 to print a message. The coordinate change triggers s3 and s4 to print a message. s4's message also triggers the observer s5.

```
import java.awt.Color

def colorObserver= new ColorObserver()
def coordinateObserver= new
CoordinateObserver()
def screenObserver= new ScreenObserver()

def pointProxy=
MultiInterceptorProxyMetaClass.getInstance(
Point )
pointProxy.interceptors << colorObserver <<
coordinateObserver

//multi-interception used here
pointProxy.use{

  def screenProxy=
MultiInterceptorProxyMetaClass.getInstance(
Screen )
  screenProxy.interceptors << screenObserver
  screenProxy.use{
```

```
println("Creating Screen s1,s2,s3,s4,s5
and Point p")
def s1= new Screen('s1'),
    s2= new Screen('s2'),
    s3= new Screen('s3'),
    s4= new Screen('s4'),
    s5= new Screen('s5')
def p= new Point(5, 5, Color.blue)

println("Creating observing
relationships:")
println(" - s1 and s2 observe color
changes to p")
println(" - s3 and s4 observe coordinate
changes to p")
println(" - s5 observes s2's and s4's
display() method")

colorObserver.addObserver(p, s1)
colorObserver.addObserver(p, s2)
coordinateObserver.addObserver(p, s3)
coordinateObserver.addObserver(p, s4)
screenObserver.addObserver(s2, s5)
screenObserver.addObserver(s4, s5)

println("Changing p's color:")
p.setColor(Color.red)

println("Changing p's x-coordinate:")
p.setX(4)
```

```
    println("done.")
  }
}
```

**/\*output:**

**Creating Screen s1,s2,s3,s4,s5 and Point p**

**Creating observing relationships:**

- s1 and s2 observe color changes to p**
- s3 and s4 observe coordinate changes to p**
- s5 observes s2's and s4's display() method**

**Changing p's color:**

**s1: Screen updated (point subject changed color).**

**s2: Screen updated (point subject changed color).**

**s5: Screen updated (screen subject changed message).**

**Changing p's x-coordinate:**

**s3: Screen updated (point subject changed coordinates).**

**s4: Screen updated (point subject changed coordinates).**

**s5: Screen updated (screen subject changed**

```
message) .
done .
*/
```

Using a MultiInterceptorProxyMetaClass and UninterceptedFriendlyInterceptor for the Decorator pattern

We can use more than one unintercepted interceptor with a proxy meta-class. A good example where this is necessary is the Decorator pattern. We re-use the MultiInterceptorProxyMetaClass from previous examples, but must write a special unintercepted interceptor, which we call an UninterceptedFriendlyInterceptor, that can be used as one of many with the MultiInterceptorProxyMetaClass.

```
abstract class
UninterceptedFriendlyInterceptor implements
Interceptor{
    def proxy= null

    abstract Object doBefore( Object object,
String methodName,
                                Object[]
arguments )

    public Object beforeInvoke(Object object,
String methodName,
                                Object[]
arguments){
        def theInterceptors= proxy.interceptors
        proxy.interceptors= null
        def result
        try{
            result= doBefore(object, methodName,
arguments)
        }catch(Exception e){
            throw e
        }
    }
}
```

```

    }finally{
        proxy.interceptors= theInterceptors
    }
    result
}
abstract boolean doInvoke()

    abstract Object doAfter( Object object,
String methodName,
                                Object[]
arguments, Object result )

    public Object afterInvoke(Object object,
String methodName,
                                Object[]
arguments, Object result){
        def theInterceptors= proxy.interceptors
        proxy.interceptors= null
        try{
            result= doAfter(object, methodName,
arguments, result)
        }catch(Exception e){
            throw e
        }finally{
            proxy.interceptors= theInterceptors
        }
    }

```

```
        result
    }
}
```

For our example Decorator pattern, we'll code an `OutputStreamWriter` that prints extra if necessary. We use decorators extended from the `UninterceptedFriendlyInterceptor`. Firstly, a `NewlineDecorator` that uses a line-width policy to perhaps place the output on a new line. And second, a very simple `WhitespaceDecorator` that ensures there's some whitespace between any two consecutive items output. Each has only very simple logic for this example.

```
abstract class PrintDecorator extends
UninterceptedFriendlyInterceptor{
    abstract Object doBefore( Object object,
String methodName,
                                Object[]
arguments )

    abstract Object doAfter( Object object,
String methodName, Object[] arguments,
                                Object result )

    //only execute the intercepted method if
it's the last class in the chain of
    //decorators around the method...
    boolean doInvoke(){ proxy.interceptors[-1]
== this }
}

class NewlineDecorator extends
PrintDecorator{
    int lineSizeSoFar= 0
```

```

    Object doBefore( Object object, String
methodName, Object[] arguments ){
        if( methodName == 'leftShift' &&
arguments[0] instanceof String ){
            if( lineSizeSoFar +
arguments[0].size() > 30){
                arguments[0]= '\r\n' + arguments[0]
                lineSizeSoFar= 0
            }else{
                lineSizeSoFar += arguments[0].size()
            }
        }
    }
}

```

```

    Object doAfter( Object object, String
methodName, Object[] arguments,
                    Object result ){
        result
    }
}

```

```

class WhitespaceDecorator extends
PrintDecorator{
    def prevOutput= ' '

```

```

    Object doBefore( Object object, String
methodName, Object[] arguments ){
        if( methodName == 'leftShift' &&
arguments[0] instanceof String ){
            if( prevOutput[-1] != ' ' &&
prevOutput[-1] != '\n' ){

```

```
        arguments[0] = ' ' + arguments[0]
    }
}
}
```

```
Object doAfter( Object object, String
methodName, Object[] arguments,
                Object result ){
    if( methodName == 'leftShift' &&
arguments[0] instanceof String ){
        prevOutput= arguments[0]
    }
}
```

```
    result  
  }  
}
```

After the classes, interceptors, and code block are matched up, the printing logic and the `OutputStreamWriter` are both unaware that the output is being decorated. Each decorator will perhaps modify the output, then pass it along to the next decorator to do the same. The distinct items of output sent to the `OutputStreamWriter` are separated by spaces, whether or not a space was in the output string in the program, and the output fits within a certain width.

```

oswProxy=
MultiInterceptorProxyMetaClass.getInstance(
OutputStreamWriter )
[ new NewlineDecorator(),
  new WhitespaceDecorator(), //the order of
these decorators is important
].each{
  it.proxy= oswProxy
  oswProxy.interceptors << it
}
oswProxy.use{
  def wtr= new OutputStreamWriter(
                    new FileOutputStream( new
File('TheOutput.txt') ) )
  wtr<< "Singing in the Rain" <<
        "hello " <<
        "climate  " <<
        "hotrod" <<
        "far out and spacy" <<
        'Clementine, darling'
  wtr.close()
}

/*output file:
Singing in the Rain hello
climate  hotrod far out and spacy
Clementine, darling
*/

```

Groovy gives us a wide variety of choices for meta-programming. We've looked at Categories and Interceptors, which change the behavior of objects within a selected block and current thread only, in other tutorials. In this tutorial, we'll learn about more ways of meta-programming in Groovy.

#### Intercepting Method Calls and Property Accesses

We can add a special method called 'invokeMethod' to a class definition that executes calls to undefined methods:

```
class MyClass{  
    def hello(){ 'invoked hello directly' }  
    def invokeMethod(String name, Object  
args){  
        return "unknown method  
$name(${args.join(', ')})"  
    }  
}  
def mine= new MyClass()  
assert mine.hello() == 'invoked hello  
directly'  
assert mine.foo("Mark", 19) == 'unknown  
method foo(Mark, 19)'
```

If our class implements GroovyInterceptable, invokeMethod is called for all method invocations whether they exist or not:

```
class MyClass implements
GroovyInterceptable{
    def hello(){ 'invoked hello() directly' }
    def invokeMethod(String name, Object
args){
        "invoked method $name(${args.join(',
')}})"
    }
}
def mine= new MyClass()
assert mine.hello() == 'invoked method
hello()'
assert mine.foo('Mark', 19) == 'invoked
method foo(Mark, 19)'

assert mine.&hello() == 'invoked hello()
directly'
    //we can still invoke a method directly
using .& syntax
```

We can get and set properties using special method names:

```

class MyClass{
    def greeting= 'accessed greeting directly'
    Object getProperty(String property){
        "read from property $property"
    }
    void setProperty(String property, Object
newValue){
        throw new Exception("wrote to property
$property")
    }
}
def mine= new MyClass()
assert mine.greeting == 'read from property
greeting'
try{
    mine.greeting= 'hi'
}catch(e){ assert e.message == 'wrote to
property greeting' }

assert mine.@greeting == 'accessed greeting
directly'
    //we can access a property directly
using .@ syntax

```

When there's a field of some name, referring to that name still considers it to be a property unless the syntax `.@` is used:

```

class MyClass{
    public greeting= 'accessed field greeting
(directly)' //field, not property
    Object getProperty(String property){
        "read from property $property"
    }
}
def mine= new MyClass()
assert mine.greeting == 'read from property
greeting'
assert mine.@greeting == 'accessed field
greeting (directly)'

```

We can call methods and access properties directly, both statically and dynamically, from within the class using various syntaxes:

```

class MyClass implements
GroovyInterceptable{
    def greeting= 'accessed greeting'
    def id= 'White: '

    Object getProperty(String property){
        try{
            return this.@id + //access field
directly
                'indirectly ' +
                this.@"$property" //access
field directly and dynamically
        }catch(e){
            return "no such property $property"

```

```

    }
}

def hello(Object[] args){ "invoked hello
with (${args.join(', ')})" }
def id(){ 'Green: ' }

def invokeMethod(String name, Object
args){
  try{
    return this.&id() + //call method
directly
        'indirectly ' +
        this.&"$name"(args) //call
method directly and dynamically
  }catch(e){
    return "no such method $name"
  }
}
}

def mine= new MyClass()
assert mine.greeting == 'White: indirectly
accessed greeting'
assert mine.farewell == 'no such property
farewell'

assert mine.hello(1, 'b', 3) == 'Green:

```

```
indirectly invoked hello with (1, b, 3)'
assert mine.foo('Mark', 19) == 'no such
method foo'
```

If we add such 'invokeMethod', 'getProperty', or 'setProperty' methods to an object using Expando or Category syntax, they act just like normal methods. Not many supplied classes have 'invokeMethod' and such defined. For such cases, we need to use MetaClasses.

#### MetaClasses

We've seen how classes behave with the default MetaClass:

```
class A{
  def bark(){ 'A: invoked bark()' }
  def invokeMethod(String name, Object
args){
  "A: missing $name(${args.join(', ')} )"
  }
}
def a= new A()
assert a.bark() == 'A: invoked bark()'
assert a.bleet() == 'A: missing bleet()'
```

We can create our own MetaClass which wraps around the existing one. DelegatingMetaClass provides the infrastructure for this, so we only need extend it with our own logic. We can do so on an instance-by-instance basis:

```
public class MyMetaClass extends
DelegatingMetaClass{
  MyMetaClass(Class theClass){
    super(theClass)
  }
  Object invokeMethod(Object object, String
methodName, Object[] arguments){
    "MyMetaClass:"
```

```
    ${super.invokeMethod(object, methodName,  
arguments)}"  
    }  
}
```

```
public class MyOtherMetaClass extends  
DelegatingMetaClass{  
    MyOtherMetaClass(Class theClass){  
        super(theClass)  
    }  
    Object invokeMethod(Object object, String  
methodName, Object[] arguments){  
        "MyOtherMetaClass:  
    ${super.invokeMethod(object, methodName,  
arguments)}"  
    }  
}
```

```
class A{  
    def bark(){ 'A: invoked bark()' }  
    def invokeMethod(String name, Object  
args){  
        "A: missing $name(${args.join(', ')})"  
    }  
}
```

```
def amc= new MyMetaClass(A)  
amc.initialize()  
def a= new A()  
a.metaClass= amc  
    //using metaClass property on an
```

instance affects only that instance...

```
def amc2= new MyOtherMetaClass(A)
amc2.initialize()
def a2= new A()
a2.metaClass= amc2
```

```
assert a.bark() == 'MyMetaClass: A: invoked
bark()'
assert a2.bark() == 'MyOtherMetaClass: A:
invoked bark()'
```

```
Thread.start{
//...even in a new thread
  assert a.bark() == 'MyMetaClass: A:
invoked bark()'
  assert a2.bark() == 'MyOtherMetaClass: A:
invoked bark()'
}
```

```
assert new A().bark() == 'A: invoked bark()'
//new instances don't have new MetaClass
```

```
assert a.bleet() == 'A: missing bleet()'
      //MetaClass invokeMethod() NOT called
here
```

Or we can do so on a class-wide basis:

```
public class MyMetaClass extends
DelegatingMetaClass{
  MyMetaClass(Class theClass){
    super(theClass)
  }
  Object invokeMethod(Object object, String
methodName, Object[] arguments){
    "MyMetaClass:
${super.invokeMethod(object, methodName,
arguments)}"
  }
}

class A{
  def bark(){ 'A: invoked bark()' }
  def invokeMethod(String name, Object
args){
    "A: missing $name(${args.join(', ')})"
  }
}

def amc= new MyMetaClass(A)
amc.initialize()

def a= new A()
```

```
import
org.codehaus.groovy.runtime.InvokerHelper
InvokerHelper.instance.metaRegistry.setMetaC
lass(A, amc)
    //all newly-created instances of A after
this call will be affected

assert a.bark() == 'A: invoked bark()'
//created before so old MetaClass used
assert a.bleet() == 'A: missing bleet()'

assert new A().bark() == 'MyMetaClass: A:
invoked bark()' //new MetaClass used
Thread.start{
    assert a.bark() == 'A: invoked bark()'
//old MetaClass used
```

```
    assert new A().bark() == 'MyMetaClass: A:
invoked bark()' //new MetaClass used
}
```

Classes we define ourselves return a MetaClass when accessing the metaClass property, but many Groovy-supplied classes don't. There's only one instance of a MetaClass in such cases:

```
class A{}
assert new A().metaClass.class ==
MetaClassImpl
assert new ArrayList().metaClass.class ==
ArrayList //class itself returned
```

When we use Groovy-supplied classes without their own MetaClass, both already-created and newly-created classes are affected by changes to the MetaClass:

```
public class MyMetaClass extends
DelegatingMetaClass{
    MyMetaClass(Class theClass){
        super(theClass)
    }
    Object invokeMethod(Object object, String
methodName, Object[] arguments){
        "MyMetaClass:
${super.invokeMethod(object, methodName,
arguments)}"
    }
}

def amc= new MyMetaClass(ArrayList)
amc.initialize()
```

```
def list1= [1, 2, 3]

import
org.codehaus.groovy.runtime.InvokerHelper
InvokerHelper.instance.metaRegistry.setMetaC
lass(ArrayList, amc)
    //all instances of ArrayList will be
affected, even already created ones

assert list1.join(',') == 'MyMetaClass:
1,2,3'
    //new MetaClass used with already
created ArrayList

def list2= [4, 5, 6]
assert list2.join(',') == 'MyMetaClass:
4,5,6'
    //new MetaClass used with newly created
ArrayList

//even in new Thread...
Thread.start{
    assert list1.join(',') == 'MyMetaClass:
1,2,3' //new MetaClass used
    assert list2.join(',') == 'MyMetaClass:
4,5,6' //new MetaClass used
```

```
    assert [7, 8, 9].join(',') ==  
    'MyMetaClass: 7,8,9' //new MetaClass used  
}
```

Other methods besides `invokeMethod` are available on the `MetaClass`:

```
Object invokeStaticMethod(Object object,  
String methodName, Object[] arguments)  
    Object invokeConstructor(Object[]  
arguments)  
    Object getProperty(Object object, String  
property)  
    void setProperty(Object object, String  
property, Object newValue)  
    Object getAttribute(Object object, String  
attribute)  
    void setAttribute(Object object, String  
attribute, Object newValue)  
    Class getTheClass()
```

For example, making the constructor return an instance of something other than what we called the constructor on:

```

public class MyMetaClass extends
DelegatingMetaClass{
    MyMetaClass(Class theClass){
        super(theClass)
    }
    Object invokeConstructor(Object[]
arguments){
        []
    }
}
class A{}

def amc= new MyMetaClass(A)
amc.initialize()
import
org.codehaus.groovy.runtime.InvokerHelper
InvokerHelper.instance.metaRegistry.setMetaC
lass(A, amc)

def a= new A()
assert a.class == ArrayList
assert ( a << 1 << 2 << 3 ).size() == 3

```

#### ExpandoMetaClass

There's some easy-to-use facilities available through the MetaClass, known as ExpandoMetaClass, to which we can add properties and methods easily:

```

class A{
    String text
}

```

```

def a1= new A(text: 'aBCdefG')
assert a1.metaClass.adaptee.class ==
MetaClassImpl //usual MetaClass type

A.metaClass.inSameCase= {->
text.toUpperCase()}
    //triggers conversion of MetaClass of A
to ExpandoMetaClass
    //then adds new instance method
'inUpperCase' to class

def a2= new A(text: 'hiJKLMnOp')
assert a2.metaClass.adaptee.getClass() ==
ExpandoMetaClass
    //MetaClass of A changed for instances
created after conversion trigger only
assert a2.inSameCase() == 'HIJKLMNOP'

assert a1.metaClass.adaptee.class ==
MetaClassImpl //still usual MetaClass type
try{ println a1.inSameCase(); assert false }
catch(e){ assert e in MissingMethodException
} //new method not available

A.metaClass.inLowerCase= {->
text.toLowerCase()}
assert a2.inLowerCase() == 'hijklmnop'

//we can replace the method definition with
another
A.metaClass.inSameCase= {->

```

```
text.toLowerCase()}  
assert a2.inSameCase() == 'hijklmnop'
```

```
//we can add static methods...
```

```
A.metaClass.'static'.inSameCase= {
```

```
it.toLowerCase()}
assert A.inSameCase('qRStuVwXyz') ==
'qrstuvwxyz'
```

We can also add properties and constructors:

```
class A{}

//we can let ExpandoMetaClass manage the
properties...
A.metaClass.character = 'Cat in the Hat'
//add property 'character'

def a1= new A()
assert a1.character == 'Cat in the Hat'

//...or we can manage the properties
ourselves...
def ourProperties =
Collections.synchronizedMap([:])
    //see tutorial on Multi-Threading
to learn about synchronized objects
A.metaClass.setType= { String value ->
    ourProperties[ "${delegate}Type" ] =
value
}
A.metaClass.getType= {->
    ourProperties[ "${delegate}Type" ]
}
a1.type= 'Hatted Cat'
assert a1.type == 'Hatted Cat'
```

```

//we can add our own constructors...
def a2= new A()
A.metaClass.constructor= {-> new A() }
try{
    a2= new A() //be careful when overriding
default or existing constructors
    assert false
}catch(Error e){ assert e in
StackOverflowError }

A.metaClass.constructor= {-> new A() }
try{
    A.metaClass.constructor << {-> new A() }
        // << notation doesn't
allow overriding
    assert false
}catch(e){ assert e in
GroovyRuntimeException }

A.metaClass.constructor= { String s-> new
A(character: s) }
a2 = new A("Thing One")

//We can quote method and property names...
A.metaClass.'changeCharacterToThingTwo'=
        {->
delegate.character = 'Thing Two' }
a2.character= 'Cat in the Hat'
a2.changeCharacterToThingTwo()
assert a2.character == 'Thing Two'

```

```
//...which is handy for dynamically
constructing method/property names...
['Hatted Cat', 'Thing', 'Boy', 'Girl',
'Mother'].each{p->
  A.metaClass."changeTypeTo${p}"= {->
delegate.type= p}
}
a2.changeTypeToBoy()
assert a2.type == 'Boy'
```

```
a2.'changeTypeToHatted Cat'()  
assert a2.type == 'Hatted Cat'
```

We can also add methods for supplied Groovy classes, ones we don't define ourselves:

```
ExpandoMetaClass.enableGlobally()  
    //call 'enableGlobally' method  
before adding to supplied class  
List.metaClass.sizeDoubled = {->  
delegate.size() * 2 }  
    //add method to an interface  
def list = [] << 1 << 2  
assert list.sizeDoubled() == 4
```

We can override MetaClass class methods such as 'invokeMethod' and 'getProperty' using ExpandoMetaClass's easy syntax:

```

class Bird{
  def name= 'Tweety'
  def twirp(){ 'i taught i saw a puddy cat'
}
}
Bird.metaClass.invokeMethod= {name, args->
  def metaMethod=
Bird.metaClass.getMetaMethod(name, args)
  //'getMetaMethod' gets method, which
may be an added or an existing one
  metaMethod?
metaMethod.invoke(delegate, args): 'no such
method'
}
def a= new Bird()
assert a.twirp() == 'i taught i saw a puddy
cat'
assert a.bleet() == 'no such method'

Bird.metaClass.getProperty= {name->
  def metaProperty=
Bird.metaClass.getMetaProperty(name)
  //'getMetaProperty' gets property, which
may be an added or an existing one
  metaProperty?
metaProperty.getProperty(delegate): 'no such
property'
}
def b= new Bird()
assert b.name == 'Tweety'
assert b.filling == 'no such property'

```

## JN3535-Reflection

We can examine classes in Groovy to find out information in the form of strings.

### Examining Classes

To find out a class's name and superclasses:

```
class A{}
assert A.name == 'A'
assert new A().class.name == 'A'
assert A.class.name == 'A' // 'class' is
optionally used here

class B extends A{}
assert B.name == 'B'

class C extends B{}
def hierarchy= []
def s = C
while(s != null){ hierarchy << s.name; s=
s.superclass }
assert hierarchy == [ 'C', 'B', 'A',
'java.lang.Object' ]
```

To examine the interfaces:

```
interface A1{}
interface A2{}
class A implements A1, A2{}
def interfacesA = [] as Set //use a set
because interfaces are unordered
A.interfaces.each{ interfacesA << it.name }
assert interfacesA == [ 'A1', 'A2',
'groovy.lang.GroovyObject' ] as Set

interface B1{}
class B extends A implements B1{}
def interfacesB = [] as Set
B.interfaces.each{ interfacesB << it.name }
assert interfacesB == [ 'B1' ] as Set
//only immediately implemented interfaces
are reported
```

We can check if a class is a class or an interface:

```
assert Observer.isInterface()
assert ! Observable.isInterface()
```

We can examine public fields and their types:

```

class A{
    def adyn //if no modifier, field is
private
    String astr
    public apdyn
    public String apstr
    protected aqdyn
}
interface B1{}
interface B2{}
class B extends A implements B1, B2{
    def bdyn
    int bint
    public bpdyn
    public int bpint
    protected bqdyn
}
def dets = [] as Set
B.fields.each{ //public fields only
    dets << [ it.name, it.type.name ] //name
of field and name of type
}
assert dets == [
    [ 'apstr', 'java.lang.String' ],
    [ 'apdyn', 'java.lang.Object' ],
    [ 'bpint', 'int' ],
    [ 'bpdyn', 'java.lang.Object' ],
    [ '__timeStamp', 'java.lang.Long' ],
//added by Groovy
] as Set

```

We can look at a certain field of a class:

```
assert Math.fields.name as Set == [ 'E',
'PI' ] as Set
assert Math.class.getField('PI').toString()
==
    'public static final double
java.lang.Math.PI'
assert Math.class.getField('PI').getDouble()
== 3.141592653589793
    //we must know the type of the value
```

We can also look at the constructors and methods of a class:

```
assert HashMap.constructors.collect{
it.parameterTypes.name } as Set ==
    [ ['int'], [], ['java.util.Map'], ['int',
'float'] ] as Set
GroovyObject.methods.each{ println it }
    //to print full details of each method of
a class
assert GroovyObject.methods.name as Set ==
    ['invokeMethod', 'getMetaClass',
'setMetaClass',
'setProperty', 'getProperty'] as Set
assert
GroovyObject.getMethod('getMetaClass').toStr
ing() ==
    'public abstract groovy.lang.MetaClass
groovy.lang.GroovyObject.getMetaClass()'
```

Some code to find out all the getters for a class:

```
getters= {
    it.methods.name.findAll{ it =~ /^get[A-Z]/
}.
    collect{
it[3].toLowerCase()+it[4..-1] }.join(', ')
}
assert getters( GroovyObject ) ==
'metaClass, property'
```

To see all nested classes for a particular class (eg, of Character):

```
assert Character.classes.name as Set ==
[ 'java.lang.Character$Subset',
'java.lang.Character$UnicodeBlock' ] as Set
```

To query a particular nested class (eg, Character.UnicodeBlock):

```
Character.UnicodeBlock.fields.name.each{
println it }
//to list all public constants
```

#### Reflecting the Reflection classes themselves

We can use reflection on the reflection classes themselves. For example:

```

assert Class.methods[0].class ==
java.lang.reflect.Method
    //find the class of any method of any
class...
java.lang.reflect.Method.methods.each{
println it.name }
    //...then find its method names...

//...to help us build a custom-formatted
listing of method details
HashMap.class.methods.each{
    println ""$it.name(
    ${it.parameterTypes.name.join(', ')} )
returns \
    $it.returnType.name
    ${it.exceptionTypes.size()>0?'throws ':''}\
    ${it.exceptionTypes.name.join(', ')}""
}

```

We can look at the modifiers of methods and classes:

```

import java.lang.reflect.Modifier
Modifier.methods.name.sort{}.each{ println
it }
    //use reflection on the reflection classes
themselves...

//...to help us build a custom-formatted
listing of modifier details
[
    (ArrayList.getMethod( 'remove',

```

```
[Object] as Class[] )):

[ 'public' ] as Set,
  (Collections.getMethod(
'synchronizedList', [List] as Class[] )):

[ 'public', 'static' ] as Set,

(Math): [ 'public', 'final' ] as Set,

(ClassLoader): [ 'public', 'abstract' ] as
Set,
].each{ key, val->
  def m= key.modifiers
  def mods= [
    ({Modifier.isPublic(it)}):
'public',
    ({Modifier.isProtected(it)}):
'protected',
    ({Modifier.isPrivate(it)}):
'private',
    ({Modifier.isInterface(it)}):
'interface',
    ({Modifier.isAbstract(it)}):
'abstract',
    ({Modifier.isFinal(it)}):
'final',
    ({Modifier.isStatic(it)}):
'static',
    ({Modifier.isVolatile(it)}):
'volatile',
```

```
    ({Modifier.isNative(it)}):  
'native',  
    ({Modifier.isStrict(it)}):  
'strict',  
    ({Modifier.isSynchronized(it)}):  
'synchronized',  
    ({Modifier.isTransient(it)}):  
'transient',  
].collect{ k, v-> k(m)? v: null } as Set
```

```
mods.removeAll( [null] )
assert mods == val
}
```

#### Manipulating Objects

When a class is unknown at compile time (eg, we only have a string representation of a class name), we can use reflection to create objects:

```
assert
Class.forName("java.util.HashMap").newInstance() == [:]

def constructor =
Class.forName("java.util.HashMap").

getConstructor( [ int, float ] as Class[] )
assert constructor.toString() == 'public
java.util.HashMap(int,float)'
assert constructor.newInstance( 12, 34.5f )
== [:]
```

We can examine and change public fields for a class referring using a String for the name:

```

class A{
    public value1
    protected value2
    A( int v ){ value1= v; value2 = v }
}
def a= new A( 100 )
assert A.getField( 'value1' ).get( a ) ==
100 //public fields only

try{ A.getField( 'value2' ).get( a ); assert
false }
catch(Exception e){ assert e instanceof
NoSuchFieldException }

A.getField( 'value1' ).set( a, 350 )
assert a.value1 == 350

```

And we can call methods using a string for the name:

```

assert String.getMethod( 'concat', [ String
] as Class[] ).
    invoke( 'Hello, ', [ 'world!' ] as
Object[] ) == 'Hello, world!'

```

#### Working with Arrays

We can examine and manipulate arrays. To enquire the public array fields of a class:

```
class A{
    public boolean alive
    public int[] codes
    public Date[] dates
    protected boolean[] states
}
//find all public array fields
def pubFields= new A().class.fields.findAll{
it.type.isArray() }.

collect{ [it.name, it.type.name] }
assert pubFields == [
    [ 'codes', '[I' ], //'[I' means array of
int
    [ 'dates', '[Ljava.util.Date;' ], //means
array of object java.util.Date
]
```

To enquire the component type/s of an array:

```

[
    (int[]): [ '[I', 'int' ],
    (Date[]): [
' [Ljava.util.Date;', 'java.util.Date' ],
    (new Date[6].class): [
' [Ljava.util.Date;', 'java.util.Date' ],

//instantiated class
    (String[][]): [
' [[Ljava.lang.String;',
' [Ljava.lang.String;' ],
].each{
    k, v -> assert [ k.name,
k.componentType.name ] == v
}

```

We can create and copy arrays when their component type and size is unknown at compile time:

```

import java.lang.reflect.Array
def a1 = [55, 66] as int[]

//component type and size unknown at compile
time...
def a2 = Array.newInstance(
a1.class.componentType, a1.size() * 2 )
assert a2.class.componentType == int
assert a2.size() == 4
System.arraycopy( a1, 0, a2, 0, a1.size() )
assert a2 as List == [55, 66, 0, 0] as List

```

We can create multi-dimensional arrays in a similar way, where component type and array sizes can be unknown at

compile time:

```
import java.lang.reflect.Array

//assertion checking code...
assert1D= {x,y->
    assert x.size() == y.size()
    for( int i: x.size() - 1 ) assert x[ i ]
== y[ i ]
}
assert2D= {x,y->
    assert x.size() == y.size()
    for( int i: x.size() - 1 ){
        assert x[ i ].size() == y[ i ].size()
        for( int j: x[ i ].size() - 1 ) assert
x[ i ][ j ] == y[ i ][ j ]
    }
}

//each is a 1-D int array with 3 elts
def a0= new char[ 3 ]
def a1= Array.newInstance( char, 3 )
def a2= Array.newInstance( char, [ 3 ] as
int[] )
assert1D( a0, a1 )
assert1D( a0, a2 )

//both are a 2-D 3x4 array of String elts
def b0= new String[3][4]
def b1= Array.newInstance( String, [ 3, 4 ]
as int[] )
```

```
assert2D( b0, b1 )
```

```
//both are a 2-D array of 6 char arrays,  
with undefined tail dimension
```

```
def c0 = new char[6][]
```

```
def c1 = Array.newInstance( char[], [ 6 ] as
int[] )
assert1D( c0, c1 )
```

We can use set() and get() to copy the contents of one array index to another:

```
import java.lang.reflect.Array
def a= [ 12, 78 ] as int[], b= new int[ 4 ]
Array.set( b, 0, Array.get( a, 0 ) )
assert b[ 0 ] == 12
```

This tutorial is loosely based on Sun's tutorial on Java Reflection, but using Groovy code instead.

## Groovy for the Office

We all know Groovy as our super hero for enhancing Java with all the latest programming features. When not in super hero mode, Groovy is just as happy as a mild-mannered office worker. Here are some links to get you started if you need some help around the office:

- The [Scriptom Module](#) can be used to script Word, Excel, PowerPoint etc. on Windows.
- [OpenXML4J](#) is a Java library dedicated to the creation and manipulation of Office Open XML (ECMA-376) and OPC based documents (for example Office 2007 Word, Excel and PowerPoint documents). OpenXML4J provides you a way to create and manipulate Open XML documents for a bunch of scenarios without using any office suite.
- [Apache POI](#) consists of APIs for manipulating various file formats based upon Microsoft's OLE 2 Compound Document format using pure Java. In short, you can read and write MS Excel, Word, PowerPoint files (97-2003 with varying levels of support) using Java.
- [WebTest's Excel Steps](#) lets you test Excel content. The examples are in XML (Ant build format) but you can use Groovy with AntBuilder too.
- [Using Java to Crack Office 2007](#) is an article about using Java to read and write any Office 2007 document. You can use Groovy's XML features to make these examples even simpler.
- Ted Neward's [Best of Both Worlds](#) whitepaper describes how to make the Java and Microsoft/.Net worlds interoperate. Most of those examples apply equally well to Groovy and .Net.
- [JExcel](#) is a commercial offering that provides an effective way to integrate Microsoft Excel into Swing applications.
- [Groovy For OpenOffice](#) is an OpenOffice Extension that adds support for scripting OpenOffice Macros in Groovy. Related Articles:
  - [Record macros in OpenOffice with Groovy](#)
  - [Groovy as a business user language?](#)

## Groovy Quick Start Project

One of the first questions I face when I start playing with a new language, is about how to set a project using this language so that I can build and package my code. While learning how to do this with Groovy I ended up putting together a starter project that has a reusable build script and runner class that allows me to quickly get a new Groovy project up and running.

Groovy Quick Start is meant to help developers new to groovy to get started by providing a way to have a groovy project up and running with a minimum effort. It provides a default layout and a reusable gant script that knows how to compile, test and package your project out of the box.

### Compiling your project

```
prompt> gant compile
```

This target knows how to compile any source files that are available in the default source location. If you execute this target out of the box should report the successful creation of the required build output folders and the compilation of one sample java class and two sample groovy classes

### Testing your project

```
prompt> gant test
```

The test target will compile the source code in the src and test\_src folders, and then it will execute the unit tests located in the tests\_src folder. When you run this target out of the box it will successfully report the execution of two sample test classes

### Preparing to distribute your project

```
prompt> gant distro
```

the distro target will create a distribution folder in your build\_output folder. This dist folder will contain a lib folder with all the jars from the project lib folder, as well as the contents of the src folder packaged as a jar. Also in the dist folder is your README.txt file and a bin folder that contains the sample batch file provided with the project. To test the distribution you can cd into your

%path%\groovyquickstart\build\_output\dist\bin and run the file "run.bat". This launch script will print the help message:

```
usage: runner [option]
```

```
"-h" help           Print out this message containing help.
"-n" name <name to greet> The name of the user to be greeted.
"-r" run            Runs some target.
"-v" version       Print version information.
```

To test the sample commands in the Runner class, enter the command "**run -n "foo" -r**" This will return the output:  
prompt> Hello from the starter class foo

This sample Runner class demonstrates how to use the CLI builder to read parameters from the command line and to execute a class in the project, based on those parameters.

### Packaging your project

```
prompt> gant package
```

This target will create a zip file with the content of your dist folder in the "\build\_output\dist" folder

## Customizing GroovyQuickStart

When you use the default folders for your project artifacts the gant script should work without modifications, If you feel that you need to add or improve your build steps just modify the gant build script to suit your needs.

## Eclipse Support

The GroovyQuickStart can be imported into Eclipse, once you use the Eclipse import facilities you will need to update the project build dependencies to point to the correct location of the Groovy libraries.

For more information review the readme file located in the "groovyquickstart\docs" folder

I hope you can take advantage of Groovy Quick Start Project to get started on your groovy project and please feel free to contact me with any questions at [davilameister@gmail.com](mailto:davilameister@gmail.com)

### *Additional info for unix/linux users*

An example 'run' script to test the distribution: [run](#)

Use that instead of the 'run.bat' that comes packaged in the groovyquickstart.zip .

### *Update*

It appears the original blog/site went down, the zip file is now attached: [groovyquickstart.zip](#)

### *Update (08/2008)*

My blog was out of commission for a while , but I am working on updating this project to use the new joint compiler, in the meantime here is a new zip file containing the run file for linux [groovyquickstart1.zip](#)

## Reference

- [Groovy API](#) - (generated with GroovyDoc)
- [GDK](#) - Groovy methods added to Java SE classes
- [Groovy 1.5 Reference Card at dzone](#) (PDF) (registration necessary)

## User Guide

Welcome to the Groovy User Guide. We hope you find it useful.

The User Guide assumes you have already [downloaded](#) and installed Groovy. See the [Getting Started Guide](#) if this is not the case.

- [Control Structures](#)
  - [Logical Branching](#)
  - [Looping](#)
  - [Returning values from if-else and try-catch blocks](#)
- [Operators](#)
  - [Operator Overloading](#)
- [Bitwise Operations](#)
- [Scripts and Classes](#)
- [Statements](#)
  - [Extended Guide to Method Signatures](#)
- [Reserved Words](#)
- [Strings and GString](#)
- [Closures](#)
  - [Closures - Formal Definition](#)

- [Closures - Informal Guide](#)
- [Builders](#)
  - [How Builders Work](#)
  - [FactoryBuilderSupport](#)
  - [The @DelegatesTo annotation](#)
- [Groovy Truth](#)
- [Input Output](#)
- [Collections](#)
- [Regular Expressions](#)
  - [Documenting Regular Expressions in Groovy](#)
- [GDK Extensions to Object](#)
- [Grape](#)
- [Advanced OO](#)
  - [Groovy way to implement interfaces](#)
- [Annotations with Groovy](#)
- [Generics](#)
- [Using Enums](#)
- [Static Import Usage](#)
- [Scoping and the Semantics of "def"](#)
- [Database features](#)
- [Dynamic Groovy](#)
  - [Evaluating the MetaClass runtime](#)
  - [ExpandoMetaClass](#)
    - [ExpandoMetaClass - Borrowing Methods](#)
    - [ExpandoMetaClass - Constructors](#)
    - [ExpandoMetaClass Domain-Specific Language](#)
    - [ExpandoMetaClass - Dynamic Method Names](#)
    - [ExpandoMetaClass - GroovyObject Methods](#)
    - [ExpandoMetaClass - Interfaces](#)
    - [ExpandoMetaClass - Methods](#)
    - [ExpandoMetaClass - Overriding static invokeMethod](#)
    - [ExpandoMetaClass - Properties](#)
    - [ExpandoMetaClass - Runtime Discovery](#)
    - [ExpandoMetaClass - Static Methods](#)
  - [Global AST Transformations](#)
  - [Local AST Transformations](#)
  - [Per-Instance MetaClass](#)
  - [Runtime mixins](#)
  - [Using invokeMethod and getProperty](#)
  - [Using methodMissing and propertyMissing](#)
- [Ant Integration with Groovy](#)
  - [The groovy Ant Task](#)
  - [The groovyc Ant Task](#)
  - [The groovydoc Ant task](#)
  - [Using Ant from Groovy](#)
  - [Using Ant Libraries with AntBuilder](#)
- [Bean Scripting Framework](#)
- [Compile-time Metaprogramming - AST Transformations](#)
  - [Bindable and Vetoable transformation](#)
  - [Building AST Guide](#)
  - [Category and Mixin transformations](#)
  - [Compiler Phase Guide](#)
  - [Delegate transformation](#)

- [Immutable AST Macro](#)
- [Immutable transformation](#)
- [Lazy transformation](#)
- [Newify transformation](#)
- [PackageScope transformation](#)
- [Singleton transformation](#)
- [GPath](#)
- [Groovy and JMX](#)
  - [Groovy JmxBuilder](#)
- [Groovy Categories](#)
- [Groovy CLI](#)
- [Groovy Console](#)
- [Groovy Math](#)
- [Groovy Shell](#)
- [Groovy Utils](#)
  - [ConfigSlurper](#)
  - [ObjectGraphBuilder](#)
  - [ObservableMap](#)
- [GUI Programming with Groovy](#)
  - [Swing Builder](#)
    - [Alphabetical Widgets List](#)
      - [SwingBuilder.borderLayout](#)
      - [SwingBuilder.action](#)
      - [SwingBuilder.actions](#)
      - [SwingBuilder.bind](#)
      - [SwingBuilder.borderLayout](#)
      - [SwingBuilder.boundedRangeModel](#)
      - [SwingBuilder.box](#)
      - [SwingBuilder.boxLayout](#)
      - [SwingBuilder.button](#)
      - [SwingBuilder.buttonGroup](#)
      - [SwingBuilder.cardLayout](#)
      - [SwingBuilder.checkBox](#)
      - [SwingBuilder.checkBoxMenuItem](#)
      - [SwingBuilder.closureColumn](#)
      - [SwingBuilder.colorChooser](#)
      - [SwingBuilder.comboBox](#)
      - [SwingBuilder.compoundBorder](#)
      - [SwingBuilder.container](#)
      - [SwingBuilder.desktopPane](#)
      - [SwingBuilder.dialog](#)
      - [SwingBuilder.editorPane](#)
      - [SwingBuilder.emptyBorder](#)
      - [SwingBuilder.etchedBorder](#)
      - [SwingBuilder.fileChooser](#)
      - [SwingBuilder.flowLayout](#)
      - [SwingBuilder.formattedTextField](#)
      - [SwingBuilder.frame](#)
      - [SwingBuilder.gridBagConstraints](#)
      - [SwingBuilder.gridBagLayout](#)
      - [SwingBuilder.gridLayout](#)
      - [SwingBuilder.imageIcon](#)
      - [SwingBuilder.internalFrame](#)

- [SwingBuilder.JComponent](#)
- [SwingBuilder.label](#)
- [SwingBuilder.layeredPane](#)
- [SwingBuilder.lineBorder](#)
- [SwingBuilder.list](#)
- [SwingBuilder.loweredBevelBorder](#)
- [SwingBuilder.map](#)
- [SwingBuilder.matteBorder](#)
- [SwingBuilder.menu](#)
- [SwingBuilder.menuBar](#)
- [SwingBuilder.menuItem](#)
- [SwingBuilder.optionPane](#)
- [SwingBuilder.overlayLayout](#)
- [SwingBuilder.panel](#)
- [SwingBuilder.passwordField](#)
- [SwingBuilder.popupMenu](#)
- [SwingBuilder.progressBar](#)
- [SwingBuilder.propertyColumn](#)
- [SwingBuilder.radioButton](#)
- [SwingBuilder.radioButtonMenuItem](#)
- [SwingBuilder.raisedBevelBorder](#)
- [SwingBuilder.raisedEtchedBorder](#)
- [SwingBuilder.scrollBar](#)
- [SwingBuilder.scrollPane](#)
- [SwingBuilder.separator](#)
- [SwingBuilder.slider](#)
- [SwingBuilder.spinner](#)
- [SwingBuilder.spinnerDateModel](#)
- [SwingBuilder.spinnerListModel](#)
- [SwingBuilder.spinnerNumberModel](#)
- [SwingBuilder.splitPane](#)
- [SwingBuilder.springLayout](#)
- [SwingBuilder.tabbedPane](#)
- [SwingBuilder.table](#)
- [SwingBuilder.tableColumn](#)
- [SwingBuilder.tableLayout](#)
- [SwingBuilder.tableModel](#)
- [SwingBuilder.td](#)
- [SwingBuilder.textArea](#)
- [SwingBuilder.textField](#)
- [SwingBuilder.textPane](#)
- [SwingBuilder.titledBorder](#)
- [SwingBuilder.toggleButton](#)
- [SwingBuilder.toolBar](#)
- [SwingBuilder.tr](#)
- [SwingBuilder.tree](#)
- [SwingBuilder.viewport](#)
- [SwingBuilder.widget](#)
- [SwingBuilder.window](#)
- [Categorical Widget List](#)
- [Extending Swing Builder](#)
- [Multithreading with SwingBuilder](#)
- [SwingXBuilder](#)

- [Effects](#)
- [Extending SwingXBuilder](#)
- [Graphs](#)
- [MultiSplitPane](#)
- [Painters](#)
- [Widgets and Common Attributes](#)
- [IDE Support](#)
  - [Eclipse Plugin](#)
    - [About GroovyEclipse Plugin v1](#)
    - [Groovy-Eclipse 2.0.0RC1 New and Noteworthy](#)
    - [Groovy Eclipse 1.5.7 Release Notes](#)
    - [Eclipse Plugin FAQ](#)
    - [Groovy-Eclipse 2.0.0M1 New and Noteworthy](#)
    - [Groovy-Eclipse 2.0.0M2 New and Noteworthy](#)
    - [Groovy-Eclipse 2.0.0 New and Noteworthy](#)
    - [Eclipse Plugin Refactoring](#)
    - [Create Your First Groovy Project](#)
    - [Install Groovy-Eclipse Plugin](#)
    - [Groovy-Eclipse 2.0.1 New and Noteworthy](#)
    - [Groovy-Eclipse coding conventions](#)
      - [\(Deprecated\) Getting Started With Groovy-Eclipse Source Code on Ubuntu](#)
      - [Getting Started With Groovy-Eclipse Source Code](#)
    - [Extending Groovy Eclipse](#)
    - [Groovy-Eclipse 2.0.2 New and Noteworthy](#)
    - [Groovy-Eclipse 2.1.0RC2 New and Noteworthy](#)
    - [Groovy-Eclipse 2.1.0 New and Noteworthy](#)
    - [Groovy-Eclipse 2.1.1RC1 New and Noteworthy](#)
    - [Groovy-Eclipse 2.1.1 New and Noteworthy](#)
    - [Groovy-Eclipse 2.1.2 New and Noteworthy](#)
    - [DSL Descriptors for Groovy-Eclipse](#)
      - [DSL examples](#)
      - [Support for testing DSLs](#)
    - [A unified approach to DSL support in IDEs](#)
    - [Groovy-Eclipse 2.5.0 New and Noteworthy](#)
    - [Groovy-Eclipse 2.5.1 New and Noteworthy](#)
    - [Compiler Switching within Groovy-Eclipse](#)
    - [Groovy-Eclipse compiler plugin for Maven](#)
    - [Groovy-Eclipse 2.1.0RC1 New and Noteworthy](#)
    - [Groovy-Eclipse 2.5.2 New and Noteworthy](#)
    - [Groovy-Eclipse 2.6.0 New and Noteworthy](#)
    - [UCOSP](#)
    - [Groovy-Eclipse 2.6.1 New and Noteworthy](#)
    - [Groovy-Eclipse 2.7.0 New and Noteworthy](#)
    - [Groovy-Eclipse 2.7.1 New and Noteworthy](#)
    - [Groovy-Eclipse 2.7.2 New and Noteworthy](#)
    - [DevelopmentBuilds](#)
  - [IntelliJ IDEA Plugin by JetBrains](#)
    - [Wish List \(JetBrains Edition\)](#)
  - [Old and deprecated pages](#)
    - [JEdit Plugin](#)
    - [Oracle JDeveloper Plugin](#)
  - [Other Plugins](#)
    - [Dash](#)

- [Emacs Groovy Mode](#)
- [Emacs Mode](#)
- [Emacs Plugin](#)
- [NetBeans Plugin](#)
- [TextMate](#)
- [UltraEdit Plugin](#)
- [Integration](#)
- [JSR 223 Scripting with Groovy](#)
- [Logging](#)
- [Migration From Classic to JSR syntax](#)
- [Processing XML](#)
  - [Creating XML using Groovy's MarkupBuilder](#)
  - [Creating XML using Groovy's StreamingMarkupBuilder](#)
  - [Creating XML with Groovy and DOM](#)
  - [Processing XML with XSLT](#)
  - [Reading XML using Groovy's DOMCategory](#)
  - [Reading XML using Groovy's XmlParser](#)
  - [Reading XML using Groovy's XmlSlurper](#)
  - [Reading XML with Groovy and DOM](#)
  - [Reading XML with Groovy and SAX](#)
  - [Reading XML with Groovy and StAX](#)
  - [Reading XML with Groovy and XPath](#)
  - [Updating XML with DOMCategory](#)
  - [Updating XML with XmlParser](#)
  - [Updating XML with XmlSlurper](#)
  - [Using Other XML Libraries](#)
    - [Creating XML with Groovy and DOM4J](#)
    - [Creating XML with Groovy and JDOM](#)
    - [Creating XML with Groovy and XOM](#)
    - [Processing XML with XQuery](#)
    - [Reading XML with Groovy and DOM4J](#)
    - [Reading XML with Groovy and JDOM](#)
    - [Reading XML with Groovy and XOM](#)
  - [Validating XML with a DTD](#)
  - [Validating XML with a W3C XML Schema](#)
  - [Validating XML with RELAX NG](#)
  - [XML Example](#)
- [Process Management](#)
- [Running Groovy on .NET 2.0 using IKVM](#)
- [Things to remember](#)
- [Things you can do but better leave undone](#)
- [Using Spring Factories with Groovy](#)
- [WebStart](#)
- [OSGi and Groovy](#)

## Control Structures

- [Logical Branching](#)
- [Looping](#)
- [Returning values from if-else and try-catch blocks](#)

## Logical Branching

## if - else statement

Groovy supports the usual if - else syntax from Java

```
def x = false
def y = false

if ( !x ) {
    x = true
}

assert x == true

if ( x ) {
    x = false
} else {
    y = true
}

assert x == y
```

Groovy also supports the normal Java "nested" if then else if syntax:

```
if ( ... ) {
    ...
} else if (...) {
    ...
} else {
    ...
}
```

## ternary operator

Groovy also supports the ternary operator:

```
def y = 5
def x = (y > 1) ? "worked" : "failed"
assert x == "worked"
```

See also: the [elvis operator](#)

#### switch statement

The switch statement in Groovy is backwards compatible with Java code; so you can fall through cases sharing the same code for multiple matches.

One difference though is that the Groovy switch statement can handle any kind of switch value and different kinds of matching can be performed.

```
def x = 1.23
def result = ""

switch ( x ) {
    case "foo":
        result = "found foo"
        // lets fall through

    case "bar":
        result += "bar"

    case [4, 5, 6, 'inList']:
        result = "list"
        break

    case 12..30:
        result = "range"
        break
}
```

```
case Integer:  
    result = "integer"  
    break  
  
case Number:  
    result = "number"  
    break  
  
default:  
    result = "default"
```

```
}
```

```
assert result == "number"
```

Switch supports the following kinds of comparisons

- Class case values matches if the switchValue is an instance of the class
- Regular expression case value matches if the string of the switchValue matches the regex
- Collection case value matches if the switchValue is contained in the collection. This also includes ranges too (since they are Lists)
- if none of the above are used then the case value matches if the case value equals the switch value

**default:** must go at the end of the switch/case as Jochen outlined in [this thread](#) from the groovy-user mailing list which Jochen states:

"because a Java switch/Case does not work like a Groovy switch/case. In Java a case can take only int compatible constants, in Groovy it can take expressions. In Java all cases share a scope, in Groovy each case has its own scope. In Groovy we call the isCase method, in Java it has to be a number we switch with. If we for example use a closure as case, then this might cause side effects. There are cases where we can let them behave the same and usually when using the java version you won't see a difference in Groovy besides the placement and logic of default."

So, while in Java the *default* can be placed anywhere in the *switch/case*, the *default* in Groovy is used more as an *else* than assigning a *default* case.

#### **How switch works**

The case statement performs a *match* on the case value using the *isCase(switchValue)* method, which defaults to *all equals(switchValue)* but has been overloaded for various types like Class or regex etc.

So you could create your own kind of matcher class and add an *isCase(switchValue)* method to provide your own kind of matching.

#### **Looping**

Groovy supports the usual while {...} loops like Java.

```
def x = 0
def y = 5

while ( y-- > 0 ) {
    x++
}

assert x == 5
```

#### *for loop*

The for loop in Groovy is much simpler and works with any kind of array, collection, Map etc.

**Note:** you can also use the standard Java / C for loop if you wish.

```
for (int i = 0; i < 5; i++) {
}

// iterate over a range
def x = 0
for ( i in 0..9 ) {
    x += i
}
assert x == 45

// iterate over a list
x = 0
for ( i in [0, 1, 2, 3, 4] ) {
    x += i
}
assert x == 10
```

```
// iterate over an array
array = (0..4).toArray()
x = 0
for ( i in array ) {
    x += i
}
assert x == 10
```

```
// iterate over a map
def map = ['abc':1, 'def':2, 'xyz':3]
x = 0
for ( e in map ) {
    x += e.value
}
assert x == 6
```

```
// iterate over values in a map
x = 0
for ( v in map.values() ) {
    x += v
}
assert x == 6
```

```
// iterate over the characters in a string
def text = "abc"
def list = []
for (c in text) {
```

```
    list.add(c)
}
assert list == ["a", "b", "c"]
```

### *closures*

In addition, you can use closures in place of most for loops, using `each()` and `eachWithIndex()`:

```
def stringList = [ "java", "perl", "python",
                  "ruby", "c#", "cobol",
                  "groovy", "jython",
                  "smalltalk", "prolog", "m", "yacc" ];

def stringMap = [ "Su" : "Sunday", "Mo" :
                 "Monday", "Tu" : "Tuesday",
                 "We" : "Wednesday", "Th" :
                 "Thursday", "Fr" : "Friday",
                 "Sa" : "Saturday" ];

stringList.each() { print " ${it}" };
println "";
// java perl python ruby c# cobol groovy
// jython smalltalk prolog m yacc

stringMap.each() { key, value -> println
"${key} == ${value}" };
// Su == Sunday
// We == Wednesday
// Mo == Monday
// Sa == Saturday
// Th == Thursday
// Tu == Tuesday
```

```
// Fr == Friday
```

```
stringList.forEachWithIndex() { obj, i ->  
println " ${i}: ${obj}" };
```

```
// 0: java  
// 1: perl  
// 2: python  
// 3: ruby  
// 4: c#  
// 5: cobol  
// 6: groovy  
// 7: jython  
// 8: smalltalk  
// 9: prolog  
// 10: m  
// 11: yacc
```

```
stringMap.forEachWithIndex() { obj, i ->  
println " ${i}: ${obj}" };
```

```
// 0: Su=Sunday  
// 1: We=Wednesday  
// 2: Mo=Monday  
// 3: Sa=Saturday
```

```
// 4: Th=Thursday
// 5: Tu=Tuesday
// 6: Fr=Friday
```

### Returning values from if-else and try-catch blocks

Since Groovy 1.6, it is possible for if/else and try/catch/finally blocks to return a value when they are the last expression in a method or a closure. No need to explicitly use the return keyword inside these constructs, as long as they are the latest expression in the block of code.

As an example, the following method will return 1, although the return keyword was omitted.

```
def method() {
    if (true) 1 else 0
}

assert method() == 1
```

For try/catch/finally blocks, the last expression evaluated is the one being returned. If an exception is thrown in the try block, the last expression in the catch block is returned instead. Note that finally blocks don't return any value.

```
def method(bool) {
    try {
        if (bool) throw new Exception("foo")
        1
    } catch(e) {
        2
    } finally {
        3
    }
}

assert method(false) == 1
assert method(true) == 2
```

## Operators

### Operators

In general all operators [supported in Java](#) are identical in Groovy. Groovy goes a step further by allowing you to [customize behavior](#) of operators on Groovy types.

- [Arithmetic and Conditional Operators](#)
- [Collection-based Operators](#)
  - [Spread Operator \(\\*.\)](#)
- [Object-Related Operators](#)
  - [Java field \(.@\)](#)
- [Other Operators](#)
  - [Elvis Operator \(?.\)](#)
  - [Safe Navigation Operator \(?.\)](#)
- [Regular Expression Operators](#)
- [Table of Operators](#)

#### *Arithmetic and Conditional Operators*

See [Operator Overloading](#) for a list of the common operators that Groovy supports.

In addition, Groovy supports the ! (not) operator as follows:

```
def expression = false  
assert !expression
```

For more details about how expressions are forced to a boolean value, see: [Groovy Truth](#).

### *Collection-based Operators*

#### Spread Operator (\*)

The Spread Operator is used to invoke an action on all items of an aggregate object. It is equivalent to calling the collect method like so:

```
parent*.action  
//equivalent to:  
parent.collect{ child -> child?.action }
```

The action may either be a method call or property access, and returns a list of the items returned from each child call. As an example:

```
assert ['cat', 'elephant']*.size() == [3, 8]
```

The Spread operator will work as expected with most of the aggregate-like classes within Groovy. You can also customize your own classes to use it by defining your own `iterator()` method as this example shows:

```

class Person { String name }
class Twin {
    Person one, two
    def iterator() {
        return [one, two].iterator()
    }
}

def tw = new Twin(one: new
Person(name: 'Tom' ),
                two: new
Person(name: 'Tim' ))
assert tw*.name == ['Tom', 'Tim']
// expanded equivalent of above:
assert tw.collect{ it.name } == ['Tom',
'Tim']

```

### Object-Related Operators

- [invokeMethod and get/setProperty](#) (.)
- Java field (.@)
- The spread java field (\*.@)
- Method Reference (.&)
- 'as' - "manual coercion" - asType(t) method
- Groovy == ( equals() ) behavior.
  - "is" for identity
- The instanceof operator (as in Java)

#### Java field (.@)

Groovy dynamically creates getter method for all your fields that can be referenced as properties:

```
class X
{
    def field
}

x = new X()
x.field = 1
println x.field    // 1
```

You can override these getters with your own implementations if you like:

```
class X
{
    def field

    def getField()
    {
        field += 1
    }
}

x = new X()
x.field = 1
println x.field    // 2
```

The @ operator allows you to override this behavior and access the field directly, so to extend the previous sample:

```
println x.@field    // 1
```

It should be mentioned that, while interesting, this is probably not a good thing to do unless you really need to. Overriding a public interface to access the internal state of an object probably means you are about to break something. Not even recommended for use in tests since it increases coupling unnecessarily.

#### **Other Operators**

- `getAt(index)` and `putAt(index, value)` for the subscript operator (e.g. `foo[1]` or `foo['bar']`, i.e. index is either an int or String)
- Range Operator (`..`) - see [Collections#Collections-Ranges](#)
- `isCase()` for the membership operator (`in`)

#### **Elvis Operator (?:)**

The "Elvis operator" is a shortening of Java's [ternary operator](#). One instance of where this is handy is for returning a 'sensible default' value if an expression resolves to `false` or `null`. A simple example might look like this:

```
def displayName = user.name ? user.name :  
"Anonymous" //traditional ternary operator  
usage  
  
def displayName = user.name ?: "Anonymous"  
// more compact Elvis operator - does same  
as above
```

#### **Safe Navigation Operator (?.)**

The Safe Navigation operator is used to avoid a `NullPointerException`. Typically when you have a reference to an object you might need to verify that it is not null before accessing methods or properties of the object. To avoid this, the safe navigation operator will simply return `null` instead of throwing an exception, like so:

```
def user = User.find( "admin" )  
//this might be null if 'admin' does not  
exist  
def streetName = user?.address?.street  
//streetName will be null if user or  
user.address is null - no NPE thrown
```

#### **Regular Expression Operators**

- `find (=~)`
- `match (==~)`

For more details, see: [Regular Expressions](#)

### Table of Operators

Operator Name	Symbol	Description
Spaceship	<=>	Useful in comparisons, returns -1 if left is smaller 0 if == to right or 1 if greater than the right
Regex find	==~	Find with a regular expression? See <a href="#">Regular Expressions</a>
Regex match	===~	Get a match via a regex? See <a href="#">Regular Expressions</a>
<a href="#">Java Field Override</a>	.@	Can be used to override generated properties to provide access to a field
<a href="#">Spread</a>	*.	Used to invoke an action on all items of an aggregate object
Spread Java Field	*.@	Amalgamation of the above two
Method Reference	.&	Get a reference to a method, can be useful for creating closures from methods
asType Operator	as	Used for groovy casting, coercing one type to another.
Membership Operator	in	Can be used as replacement for collection.contains()
Identity Operator	is	Identity check. Since == is overridden in Groovy with the meaning of equality we need some fallback to check for object identity.
<a href="#">Safe Navigation</a>	?.	returns nulls instead of throwing NullPointerExceptions
<a href="#">Elvis Operator</a>	?:	Shorter ternary operator

### Operator Overloading

Groovy supports operator overloading which makes working with Numbers, Collections, Maps and various other data structures easier to use.

Various operators in Groovy are mapped onto regular Java method calls on objects.

This allows you the developer to provide your own Java or Groovy objects which can take advantage of operator overloading. The following table describes the operators supported in Groovy and the methods they map to.

Operator	Method
<code>a + b</code>	<code>a.plus(b)</code>
<code>a - b</code>	<code>a.minus(b)</code>
<code>a * b</code>	<code>a.multiply(b)</code>
<code>a ** b</code>	<code>a.power(b)</code>
<code>a / b</code>	<code>a.div(b)</code>
<code>a % b</code>	<code>a.mod(b)</code>
<code>a   b</code>	<code>a.or(b)</code>
<code>a &amp; b</code>	<code>a.and(b)</code>
<code>a ^ b</code>	<code>a.xor(b)</code>
<code>a++</code> or <code>++a</code>	<code>a.next()</code>
<code>a--</code> or <code>--a</code>	<code>a.previous()</code>
<code>a[b]</code>	<code>a.getAt(b)</code>
<code>a[b] = c</code>	<code>a.putAt(b, c)</code>
<code>a &lt;&lt; b</code>	<code>a.leftShift(b)</code>
<code>a &gt;&gt; b</code>	<code>a.rightShift(b)</code>
<code>switch(a) { case(b) : }</code>	<code>b.isCase(a)</code>
<code>~a</code>	<code>a.bitwiseNegate()</code>
<code>-a</code>	<code>a.negative()</code>
<code>+a</code>	<code>a.positive()</code>

Note that all the following comparison operators handle nulls gracefully avoiding the throwing of [java.lang.NullPointerException](#)

Operator	Method
----------	--------

<code>a == b</code>	<code>a.equals(b)</code> or <code>a.compareTo(b) == 0</code> **
<code>a != b</code>	<code>! a.equals(b)</code>
<code>a &lt;=&gt; b</code>	<code>a.compareTo(b)</code>
<code>a &gt; b</code>	<code>a.compareTo(b) &gt; 0</code>
<code>a &gt;= b</code>	<code>a.compareTo(b) &gt;= 0</code>
<code>a &lt; b</code>	<code>a.compareTo(b) &lt; 0</code>
<code>a &lt;= b</code>	<code>a.compareTo(b) &lt;= 0</code>

\*\* Note: The `==` operator doesn't always exactly match the `.equals()` method. You can think of them as equivalent in most situations. In situations where two objects might be thought "equal" via normal Groovy "coercion" mechanisms, the `==` operator will report them as equal; the `.equals()` method will not do so if doing so would break the normal rules Java has around the equals method. Expect further improvements to Groovy over time to provide clearer, more powerful and more consistent behavior in this area.

#### ***Notes about operations***

Also in Groovy comparison operators handle nulls gracefully. So that `a == b` will never throw a `NullPointerException` whether `a` or `b` or both are null.

```
def a = null  
def b = "foo"  
  
assert a != b  
assert b != a  
assert a == null
```

In addition when comparing numbers of different types the type coercion rules apply to convert numbers to the largest numeric type before the comparison. So the following is valid in Groovy

```
Byte a = 12
```

```
Double b = 10
```

```
assert a instanceof Byte
```

```
assert b instanceof Double
```

```
assert a > b
```

## Bitwise Operations

From Groovy 1.0 beta 10, Groovy supports bitwise operations:

<<, >>, >>>, |, &, ^, and ~.

Operator Symbol	Meaning
<<	Bitwise Left Shift Operator
>>	Bitwise Right Shift Operator
>>>	Bitwise Unsigned Right Shift Operator
	Bitwise Or Operator
&	Bitwise And Operator
^	Bitwise Xor Operator
~	Bitwise Negation Operator
<<=	Bitwise Left Shift Assign Operator
>>=	Bitwise Right Shift Assign Operator
>>>=	Bitwise Unsigned Right Shift Assign Operator
=	Bitwise Or Assign Operator
&=	Bitwise And Assign Operator
^=	Bitwise Xor Operator

For example,

```
assert (1 << 2) == 4           // bitwise left
shift
assert (4 >> 1) == 2           // bitwise
right shift
assert (15 >>> 1) == 7         // bitwise
unsigned right shift
assert (3 | 6) == 7            // bitwise or
assert (3 & 6) == 2            // bitwise and
assert (3 ^ 6) == 5            // bitwise xor
int mostlyOnes = 0xFFFFFFFFE
assert ~mostlyOnes == 1       // bitwise
negation
```

## Scripts and Classes

Classes are defined in Groovy similarly to Java. Methods can be class (static) or instance based and can be public, protected, private and support all the usual Java modifiers like synchronized. Package and class imports use the Java syntax (including static imports). Groovy automatically imports the following:

- java.lang
- java.io
- java.math
- java.net
- java.util
- groovy.lang
- groovy.util

One difference between Java and Groovy is that by default things are public unless you specify otherwise.

Groovy also merges the idea of fields and properties together to make code simpler, please refer to the [Groovy Beans](#) section for details of how they work.

Each class in Groovy is a Java class at the bytecode / JVM level. Any methods declared will be available to Java and vice versa. You can specify the types of parameters or return types on methods so that they work nicely in normal Java code. Also you can implement interfaces or overload Java methods using this approach.

If you omit the types of any methods or properties they will default to java.lang.Object at the bytecode/JVM level.

You can also use another class implemented in Groovy. e.g.

```
//Callee.groovy
class Callee {
    void hello() {
        println "hello, world"
    }
}
```

```
//Caller.groovy
c = new Callee()
c.hello()
```

```
groovy -cp . caller
```

Make sure the classpath is OK.

### Scripts

Groovy supports plain scripts, which do not have a class declaration. Imports are supported at the front of a script in the same way that they can be at the front of a class. Here's the hello world script:

```
println "Nice cheese Gromit!"
```

You can [run](#) scripts in the [interactive terminal](#), from the [command-line](#), in your [IDE](#), as a [Unix script](#), or [embedded](#) in your own Java code.

If you compile the above script to bytecode using [groovyc](#), you get a single class named after the name of the script. e.g. if this was saved in `Foo.script` you'd get a `Foo.class` file.

You can run this Java code on the command line (assuming you're classpath has `groovy.jar` and `asm.jar`).

```
java Foo
```

This will execute the autogenerated `main(String[] args)` method in the bytecode which instantiates the `Foo` class,

which extends the

[Script](#) class and then call its `run()` method. You may also use this class directly in Java code, passing in variables to the script.

```
import groovy.lang.Binding;
import groovy.lang.Script;

public class UseFoo {
    public static void main(String[] args) {
        // lets pass in some variables
        Binding binding = new Binding();
        binding.setVariable("cheese", "Cheddar")
        binding.setVariable("args", args)

        Script foo = new Foo(binding);
        foo.run();
    }
}
```

There's no need to use a Binding if you don't want to; Foo will have a no-argument constructor as well. Though using a Binding you can easily pass in variables. After the end of the script any variables created will be in the Binding for you to access in Java.

Unlike classes, variables are not required to be declared (*def* is not required) in scripts. Variables referenced in a script are automatically created and put into the Binding.

### Scripts and functions

If you just want to write some simple scripts and need some simple functions you can declare functions without writing a class.

One difference from normal class-based groovy is that the *def* keyword is required to define a function outside of a class.

Here's an example of a simple script with a function. Note that if ever you need things like static or instance variables and so forth then maybe its time to actually write a class 😊

```
def foo(list, value) {
    println "Calling function foo() with
param ${value}"
    list << value
}

x = []
foo(x, 1)
foo(x, 2)
assert x == [1, 2]

println "Creating list ${x}"
```

## Statements

Groovy uses a similar syntax to Java although in Groovy semicolons are optional.

This saves a little typing but also makes code look much cleaner (surprisingly so for such a minor change). So normally if one statement is on each line you can omit semicolons altogether - though its no problem to use them if you want to. If you want to put multiple statements on a line use a semicolon to separate the statements.

```
def x = [1, 2, 3]
println x
def y = 5; def x = y + 7
println x
assert x == 12
```

If the end of the line is reached and the current statement is not yet complete it can be spanned across multiple lines. So for things like method parameters or creating lists or for complex if expressions you can span multiple lines.

```
def x = [1, 2, 3,
 4, 5, 6]
println(
  x
)
if (x != null &&
  x.size() > 5) {
  println("Works!")
}
else {
  assert false: "should never happen ${x}"
}
```

### Multiple assignments

The previous code snippets showed some examples of assignments, but Groovy also supports multiple assignments.

In Groovy 1.6, there is only one syntax addition for being able to define and assign several variables at once:

```
def (a, b) = [1, 2]

assert a == 1
assert b == 2
```

A more meaningful example may be methods returning longitude and latitude coordinates. If these coordinates are represented as a list of two elements, you can easily get back to each element as follows:

```
def geocode(String location) {
    // implementation returns [48.824068,
    2.531733] for Paris, France
    [48.824068, 2.531733]
}

def (_lat, _long) = geocode("Paris, France")

assert _lat == 48.824068
assert _long == 2.531733
```

And you can also define the types of the variables in one shot as follows:

```
def (int i, String s) = [1, 'Groovy']

assert i == 1
assert s == 'Groovy'
```

For the assignment (with prior definition of the variables), just omit the def keyword:

```
def firstname, lastname

(firstname, lastname) = "Guillaume
Laforge".tokenize()

assert firstname == "Guillaume"
assert lastname == "Laforge"
```

If the list on the right-hand side contains more elements than the number of variables on the left-hand side, only the first elements will be assigned in order into the variables. Also, when there are less elements than variables, the

extra variables will be assigned null.

So for the case with more variables than list elements, here, c will be null:

```
def elements = [1, 2]  
def (a, b, c) = elements  
  
assert a == 1  
assert b == 2  
assert c == null
```

Whereas in the case where there are more list elements than variables, we'll get the following expectations:

```
def elements = [1, 2, 3, 4]  
def (a, b, c) = elements  
  
assert a == 1  
assert b == 2  
assert c == 3
```

For the curious minds, supporting multiple assignments also means we can do the standard school swap case in one line:

```
// given those two variables  
def a = 1, b = 2  
  
// swap variables with a list  
(a, b) = [b, a]  
  
assert a == 2  
assert b == 1
```

## Comments

The characters `"/"` begin a comment that last for the rest of the line.

```
print "hello" // This is a silly print  
statement
```

The characters `"/"` begin a comment that lasts until the first `"/"`.

```
/* This is a long comment  
about our favorite println */  
println "hello"
```

The character `"#"` is not a comment character.

```
// This doesn't work:  
# Bad comment
```

## Method calls

Method calling syntax is similar to Java where methods can be called on an object (using dot) or a method on the current class can be called. Static and instance methods are supported.

```

class Foo {
  def calculatePrice() {
    1.23
  }

  static void main(args) {
    def foo = new Foo()
    def p = foo.calculatePrice()
    assert p > 0

    println "Found price: " + p
  }
}

```

Notice that the *return* statement is optional at the end of methods. Also you don't need to specify a return type (it will default to Object in the bytecode if none is specified).

#### Optional parenthesis

Method calls in Groovy can omit the parenthesis if there is at least one parameter and there is no ambiguity.

```

println "Hello world"
System.out.println "Nice cheese Gromit!"

```

It is also possible to omit parenthesis when using named arguments. This makes for nicer DSLs:

```

compare fund: "SuperInvestment", withBench:
  "NIKEI"
monster.move from: [3,4], to: [4,5]

```

#### Named parameter passing

When calling a method you can pass in named parameters. Parameter names and values are separated by a colon (like the Map syntax) though the parameter names are identifiers rather than Strings.

Currently this kind of method passing is only implemented for calling methods which take a Map or for constructing JavaBeans.

```
def bean = new Expando(name: "James",
location: "London", id: 123)
println "Hey " + bean.name
assert bean.id == 123
```

### Passing closures into methods

Closures are described in more detail

[here](#)

Closures can be passed into methods like any other object

```
def closure = { param -> param + 1 }
def answer = [1, 2].collect(closure)
assert answer == [2, 3]
```

Though there is some syntax sugar to make calling methods which take a closure easier. Instead of specifying parenthesis, you can just specify a closure. e.g.

```
answer = [1, 2].collect { param -> param + 1
}
assert answer == [2, 3]
```

The above code is equivalent to the previous code, just a little more groovy. If a method takes parameters you can leave the closure outside of the parenthesis (provided that the closure parameter is the last parameter on the underlying method).

```
def value = [1, 2, 3].inject(0) { count,
item -> count + item }
assert value == 6
```

The above code is equivalent to the following (but just neater)

```
def value = [1, 2, 3].inject(0, { count,
item -> count + item })
assert value == 6
```

### Important Note

Note that when using the neater syntax for specifying closures either without parenthesis or by specifying the closure after the parenthesis, the closure must start on the same line. i.e. the { symbol must be on the same line as the method call statement. Otherwise the parser interprets the { as a start of a block.

For a in depth description of what kind of method signatures Groovy supports see the [Extended Guide to Method Signatures](#)

### Dynamic method dispatch

Groovy always uses dynamic dispatch, even if a variable is constrained by a type. The type of the variable only ensures that the variable is at least of that type and avoids you assigning a String to for example int. Dynamic method dispatch is often referred to as *dynamic typing* whereas Java uses *static typing* by default.

```
def dynamicObject = "hello
world".replaceAll("world", "Gromit")
dynamicObject += "!"
assert dynamicObject == "hello Gromit!"
String staticObject = "hello there"
staticObject += "!"
assert staticObject == "hello there!"
```

### Properties

These are described in more detail in the [Groovy Beans](#) section.

To access properties you use dot with the property name. e.g.

```
def bean = new Expando(name:"James",
location:"London", id:123)
def name = bean.name
println("Hey ${name}")
bean.location = "Vegas"
println bean.name + " is now in " +
bean.location
assert bean.location == "Vegas"
```

The above uses a special bean called Expando which allows properties to be added dynamically at runtime.

An Expando is a Map which behaves as a dynamic bean: adding new key/value pairs add the equivalent getter and setter methods, as if they were defined in a real bean.

### Safe navigation

If you are walking a complex object graph and don't want to have NullPointerExceptions thrown you can use the `?.` operator rather than `.` to perform your navigation.

```
def foo = null
def bar = foo?.something?.myMethod()
assert bar == null
```

### Extended Guide to Method Signatures

Groovy supports most of the signatures you know from Java. Since Groovy 1.5 Groovy supports full usage of generics in method declarations. Please see a Java tutorial for details.

#### General Form

I will often show a more general method signature like `def foo(x, y, p1, p2, ..., pn)`

This is not valid Groovy code, it is just there to show that a method named `foo` has  $n+2$  parameters. Usually this means also that  $n$  could be 0. In that case, the real resulting method signature would look like: `def foo(x,y)`

A form of `def foo(x,y, p1, p2, ..., pn, q1, q2, ..., qm)` would mean  $n+m+2$  parameters. Likewise a method call `foo(x, p1, p2, ..., pn)` is a method call with  $n+1$  arguments.

I will use the word **parameter** when a method declaration is involved and **argument** in case of method calls. I will in general use the method name `foo` as a placeholder for a real method name. I will use `T` if a certain type is required.

This could be Object, a primitive type or any class you like to use. If  $T$  cannot be chosen freely, then I will mention the possible types for  $T$

### **Variable Arguments**

Like Java since Java5 Groovy supports variable arguments. To use them you have to give a special method signature where the last parameter is an array. `def foo(p1, ..., pn, T... args)`. Here foo supports  $n$  arguments by default, but also an unspecified number of further arguments exceeding  $n$ .

Example:

```
def foo(Object... args){args.length}
assert foo() == 0
assert foo(1,2) == 2
```

In this example we defined a method `foo`, that is able to take any number of arguments, including no arguments at all. `args.length` will then specify the number of arguments. `T...` is the same syntax as used in Java and is internally represented as array where the parameter has a special flag set. Groovy allows as alternative `T[]`. That means any method that has an array as last parameter is seen by Groovy as a method that takes a variable number of arguments. These methods are seen as such by Java too, since the "variable arguments flag" is set. Methods defined in Java using `T...` behave like a variable arguments method. Methods defined in Java using `T[]` will not have the special flag set, but Groovy will still see them as variable arguments methods.

If in case of `def foo(T[] args)`, `foo` is called with null, then `args` will be null and not an array of length 1 with null as only element. If `foo` is called with an array as argument, then `args` will be that array instead of an array of length 1 containing the given array as only element. `T...` will not behave different. This is no special logic we thought of, that is how variable arguments behave in Java too.

One more important point to mention is maybe a method overloaded with a method that takes variable arguments. Groovy (and java) will select the most specific method, that means if there is a method taking one argument and the parameter type is an array and there is also another method, that takes also only one argument and the argument is not an array, then the other method is preferred. Examples:

```
def foo(Object[] args) {1}
def foo (x) {2}
def foo (x,y) {3}

assert foo() == 1
assert foo(1) == 2
assert foo(1,2) == 3
def x = [1,2] as Object[]
assert foo(x) == 1
assert foo(1,1,1) == 1
```

**Note:** `T[]` or `T...` do have to be the last parameter, this clashes with Closures

### *Closures*

See also [Closures](#).

Groovy allows you to attach "blocks" to method calls like in:

```
foo() { println it }
```

To be able to attach a "block" in this way, the method signature must be `def foo(p1, p2, ..., pn, T t)`, where `T` is `Object`, `Closure` or no explicit type.

Example:

```
def foo(x,closure) {x+closure.call()}
assert 1 == foo(1) {0}
assert 2 == foo(1) {1}
```

You are also allowed to use the "block" as normal argument

```
foo(1, {1})
```

If variable arguments and Closures are combined you will have the problem, that the closure needs to be the last argument, but the parameter enabling variable arguments needs to be the last one too. You could use code like this to check that at runtime:

```
def foo(Object[] args) {
    if (!args || !(args[-1] instanceof
Closure)) {
        throw new
IllegalArgumentException("Last argument must
be a closure")
    }
    ...
}
```

!args will be true if args is null or an array of length 0, args[-1] refers to the last element of the array.

**Note:** the "block" is always the last argument

### *Named Arguments*

See [Maps](#)

Named arguments requires the following method signature `def foo(T t, p1, p2, ..., pn)`, but the real work is done by the compiler where the the method call is defined. A method call `foo(p1:e1, p2:e2, ..., pn:en, q1, q2, ..., qm)` will always be transformed to `foo([p1:e1, p2:e2, ..., pn:en], q1, q2, ..., qm)`. If you mix the positions of the pi and qi elements, then the compiler will still force the same transformed signature.

Example:

```

def foo(x,y,z) {[x,y,z]}
assert foo(a:1,b:2,3,4) == [[a:1, b:2], 3,
4]
assert foo(a:1,3,b:2,4) == [[a:1, b:2], 3,
4]
assert foo(3,4,a:1,b:2) == [[a:1, b:2], 3,
4]

assert foo(4,3,b:2,a:1) == [[b:2, a:1], 4,
3]

```

The type  $\mathbb{T}$  can be Object, Map or no explicit type.

(In fact other types are possible for  $\mathbb{T}$ . Any type compatible to `[ : ].getClass()` can be used. But the type may change for example from HashMap (Groovy 1.0) to LinkedHashMap (since Groovy 1.5), so it is only safe to use the less specialized types.)

Combining named arguments with closures or variable arguments is no problem. You can make the map the first element of the variable arguments part or you can let it have its own parameter: That is for you to decide. If you combine named arguments and closures, you will need two parameters. One for the map and one for the closure.

In case of `def foo(T t, p1, p2, ..., pn)` all named arguments will be in  $t$ , but that also means that you can not make a method call where you access  $p_i$  by name. Example

```

def foo(x,y){}
foo(x:1,y:2)

```

This code will fail at runtime, because the method `foo` expects two arguments, but the map you gave is only one argument.

**Note:** the map is always the first argument

### *Arguments with Default Values*

Groovy supports also the usage of methods with default values for parameters. Any parameter  $\mathbb{T} t$  can have a default value using `T t=x`, where  $x$  is the value. The usage of the default value is not bound to a special type. In fact this is a short form to declare an overloaded method. `def foo(p1, p2, ..., pn, T t=x, q1, q2, ..., qm)` becomes a method `def foo(p1, p2, ..., pn, T t, q1, q2, ..., qm)`, where  $t$  has no default value and a method `def foo(p1, p2, ..., pn, q1, q2, ..., qm)` this implementation:

```
def foo(p1, p2, ..., pn, q1, q2, ..., qm) {
    // note: this is not working groovy code,
    just pseudo code!
    foo(p1, p2, ..., pn, x, q1, q2, ..., qm)
}
```

If multiple default values are used, then the parameter with the default value most right will be eliminated like seen here and the resulting method signature will be processed again. That means `def foo(p1, p2, ..., pn, T t1=x, T t2=y, q1, q2, ..., qm)` will first produce `def foo(p1, p2, ..., pn, T t1, T t2, q1, q2, ..., qm)` as the most general method and then continue the processing with `def foo(p1, p2, ..., pn, T t1=x, q1, q2, ..., qm)` which calls the other method and sets by using `y` for `t2`. since the signature still contains a default value the compiler will create `def foo(p1, p2, ..., pn, q1, q2, ..., qm)`, which calls the most general method using `x` for `t1` and `y` for `{t2}`. This means if you use `n` default values, the compiler will produce `n+1` methods. You are not required to group the parameters with default values together in any way.

Example:

```
def foo(x=1,y=2) {x+y}
assert foo() == 3
assert foo(2) == 4
assert foo(5,10) == 15
```

You can combine default values with maps `def foo(Map m=[:],x,y)` to get optional named arguments, with closures `def foo(x,y,Closure c={})` to get optional closures and theoretically with variable arguments `def foo(x,y, Object[] args=[1,2])`, but there should be only rare cases where default arguments and variable arguments combined like this is making sense.

## Reserved Words

The following words are reserved in Groovy:

```
abstract
as
assert
boolean
break
```

**byte**  
**case**  
**catch**  
**char**  
**class**  
**const**  
**continue**  
**def**  
**default**  
**do**  
**double**  
**else**  
**enum**  
**extends**  
**false**  
**final**  
**finally**  
**float**  
**for**  
**goto**  
**if**  
**implements**  
**import**  
**in**  
**instanceof**  
**int**  
**interface**  
**long**  
**native**  
**new**  
**null**

**package**  
**private**  
**protected**  
**public**  
**return**  
**short**  
**static**  
**strictfp**  
**super**  
**switch**  
**synchronized**  
**this**  
**threadsafe**  
**throw**  
**throws**  
**transient**  
**true**  
**try**

**void**  
**volatile**  
**while**

#### *Modifiers*

The following words (included in the full list above) can be used as modifiers:

**abstract, final, native, private, protected,**  
**public, static,**  
**strictfp, synchronized, threadsafe,**  
**transient, volatile**

This is the same as in Java.

#### *Type declarations*

The following words (included in the full list above) can be used to start type declarations:

**class, enum, interface**

#### *Property names*

The following reserved words (included in the full list above) are allowed as property names:

**as, catch, class, def, default, do, else,**  
**finally, for, goto, if, in, static, switch,**  
**try, while**

## **Strings and GString**

Groovy uses both " and ' for strings. Either can be used. Using either type of string allows you to use strings with quotations easily.

```
println "he said 'cheese' once"  
println 'he said "cheese!" again'
```

The groovy parser supports the notation `\uab12` (i.e. a leading backslash and precisely four hex digits after the 'u'). This notation can be used in strings or anywhere in the program like the Java parser does.

### Concatenation

Strings may be concatenated with "+". For example:

```
#!/usr/bin/env groovy  
  
a = "world"  
print "hello " + a + "\n"
```

### Multi-line strings

Regular strings in Groovy cannot span multiple lines.

As an exception to this rule, a backslash at the end of a line disappears and joins the current line with the next.

```
// this is a compile error  
def foo = "hello
```

If you have a block of text which you wish to use but don't want to have to encode it all (e.g. if its a block of HTML or something) then you can use the `"""` syntax.

```

def name = "James"
def text = """\
hello there ${name}
how are you today?
""

assert text != null
println(text)

```

Because of the leading backslash, the string `text` contains exactly two newlines. There are always represented by the character `'\n'`, regardless of the line-termination conventions of the host system.

It is also possible to use triple single quotes for multi-line strings, but they are not `GStrings`, so interpolating variables doesn't work in that case.

### Slashy String literals

It is possible to use the another notation for String literals with the added benefit of not needing additional backslashes to escape special characters. That is especially handy with regular expressions or Windows file/directory path names.

```

def s = /. *foo.*/
def dirname = /^.*\//
def basename = /[Strings and GString^\/]+$/

```

The only escape that is supported in a slashy string is to use `\/` to embed a slash into the string. A consequence of this is that you can't have a backslash as the last character of a slashy string directly (or Groovy will think you are trying to escape the closing string terminator - and hence won't think you have terminated your string). Instead use:

```

def bs = '\\\\'; def winpath=/C:\windows\system32$bs/ or def
winpath=/C:\windows\system32\${}/.

```

Another gotcha: don't try using an empty slashy string: `def x = //`, the slashes will be treated as a line comment delimiter, but `def z = /${}/` would be fine.

A final gotcha: currently a slashy string can't be the left hand side (LHS) expression of an `assert` statement while you are eliminating the brackets, e.g.:

```
assert 'ab' == 'a' + 'b'    // OK, no slashy
string
assert 'a' + 'b' == /ab/   // OK, slashy
string on RHS
assert (/ab/ == 'a' + 'b') // brackets
currently required if slashy string is on
LHS
```

This is a feature of the current grammar rather than a desired goal and may one day be altered to remove this anomaly - it is a tricky edge case in the grammar.

✓ **Examples**

For more examples, read about [Regular Expressions](#).

### Strings are immutable

This can be seen with these two code snips, which you can cut and paste into groovyConsole:

```
st = ["status":"test"]
sn = st
println sn
st.status = "tset"
println sn
```

Above both variables are references to the map.  
If you do the same thing with Strings, the behavior is different:

```
st = "test"
sn = st
println sn
st = "tset"
println sn
```

Here, sn and st point at the very same map object in memory in the first example, while in the second snippet, at the end, st points at a different place in memory where there's the new immutable string.

### GStrings

Strings that are declared inside double-quotes (i.e. either single double-quotes or triple double-quotes for multi-line strings) can contain arbitrary expressions inside them as shown above using the `$(expression)` syntax in a similar way to JSP EL, Velocity and Jexl. Any valid Groovy expression can be enclosed in the `$(...)` including method calls etc. GStrings are defined the same way as normal Strings would be created in Java. Here is a simple example involving a simple variable and an expression:

```
foxtype = 'quick'
foxcolor = ['b', 'r', 'o', 'w', 'n']
println "The $foxtype ${foxcolor.join()}
fox"
// => The quick brown fox
```

What actually happens is whenever a string expression contains a `$(...)` expression then rather than a normal `java.lang.String` instance, a [GString](#) object is created which contains the text and values used inside the String.

GString can involve lazy evaluation so it's not until the `toString()` method is invoked that the GString is evaluated. This lazy evaluation is useful for things like logging as it allows the calculation of the string, the calls to `toString()` on the values, and the concatenation of the different strings to be done lazily if at all. Here is an example that illustrates lazy evaluation:

```
println new Date()
x = "It is currently ${ new Date() }"
assert x.values[0] instanceof Date
y = "It is currently ${ writer -> writer <<
new Date() }"
assert y.values[0] instanceof Closure
sleep 5000
println x
println y
```

which outputs the following:

```
Thu Apr 17 23:18:17 EST 2008
It is currently Thu Apr 17 23:18:17 EST 2008
It is currently Thu Apr 17 23:18:22 EST 2008
```

To explain this output, we need to understand additional rules about GStrings. Each value in the GString (where \$ appears) is the result of an expression. Most often, the expression is evaluated resulting in an Object, as in all the examples above except for the case of the variable `y` in which the expression is a Closure.

Only when the GString is converted into a String (as for `println`) is the expression's result converted and incorporated into the result String. Each of the values to be substituted into the GString is obtained by applying one of the following methods:

- General Object instances are converted by the `toString()` method; there is special handling for types like array, Stream, Collection, etc.
- Closure instances taking 0 parameters are called and the returned value is converted to String by the same rules as for a general Object (a returned Closure will not be called).
- Closure instances taking 1 parameter are called with a `Writer` as the parameter; characters written to the `Writer` become part of the String result. The return from the Closure is ignored.
- Closure instances taking more than 1 parameter are not called; the GString throws `GroovyRuntimeException`.

So, in the example above, `x` contains an instance of `Date` resulting from `new Date()` being invoked at the time `x` was assigned. When we print `x` later, that calculated value is placed in the appropriate place in the GString and output giving us the date at GString definition time. The other GString, `y`, contains a 1 parameter closure which means that the closure will be stored away in the GString. When we print out `y`, the closure will be invoked and give the date at printing time.

## 🚫 GStrings aren't Strings

GString and String are two distinct classes, and hence use of GString objects as keys for Map objects or comparisons involving GString objects, can produce unexpected results when combined with String objects since a GString and a String won't have the same hashCode nor will they be equal. There is no automatic coercion between the two types for comparisons or map keys, so it's sometimes necessary to explicitly invoke toString() on GString objects.

On the other hand GStrings can easily be made into Strings

```
def s = "String "  
def g = "GString created at ${new Date()}"  
def x = s + g  
assert s instanceof String  
assert g instanceof GString  
assert x instanceof String
```

Unexpected conversion to String can lead to problems when code is expecting a GString, as for methods in groovy.sql classes. That's a case where explicit types can be beneficial:

```
def s = "String "  
def g = "GString created at ${new Date()}"  
GString x = GString.EMPTY + s + g  
assert s instanceof String  
assert g instanceof GString  
assert x instanceof GString  
println x
```

### ***GString Use Cases - Templating***

GString is pretty handy when you don't want to use a template engine, or when you really want full lazy evaluation of GStrings. When some variable embedded in a GString, the toString() is called on that variable to get a string representation, and it's inserted into the final string. But when it's closures you embed, we don't call the toString() on closure. If that was the case, you'd get a string like "Script7\$\_run\_closure1@8dc50e" embedded in your final string.

Instead, we distinguish two use cases:

1. a closure with no parameter, i.e.  
{ -> }
2. a closure with one parameter.

And the expected behaviors:

- For a closure with no parameter, Groovy calls the closure without parameter, and calls toString() on the result.
- For a closure with one parameter, the parameter is actually a StringWriter to which you can append text in the body of your closure.
- For a closure with more parameters, an exception will be thrown.

Illustration:

```
def noParam = { -> println "I'm executed";
return "no param" }
def oneParam = { out -> println "Executed
with a ${out.class}
parameter"; out << "one param" }

assert noParam.toString() != "no param"

println "${noParam} - ${oneParam}"
```

Result:

```
I'm executed
Executed with a class java.io.StringWriter
parameter
no param - one param
```

The embedded closure could be used in a nested manner:

```
def deepest = {-> "deepest"}
def deep = {-> "deeper and $deepest"}
println "how deep is deep? $deep"

// which is equivalent to:
println "how deep is deep? ${{-> "deeper and
${{-> "deepest"}}}}" // for demonstrating
it is a nested usage only
```

result:

**how deep is deep? deeper and deepest  
how deep is deep? deeper and deepest**

### ***GString Use Cases - GSQL***

Another use case for GString is [GSQL](#) where parameters can be passed into SQL statements using this same mechanism which makes for a neat way to integrate Groovy with other languages like SQL. GroovySql then converts the expressions to ? and uses a JDBC PreparedStatement and passes the values in, preserving their types.

If you explicitly want to coerce the GString to a String you can use the toString() method. Groovy can also automatically coerce GStrings into Strings for you.

### **String and StringBuffer Methods**

See: [JN1525-Strings](#).

### **More Examples**

The following example passes the test:

```
aaa = "bread", "apple", "egg"
items = aaa.split(',')
assert items[1] == "apple"
items.each{ println "item: $it" }
```

and outputs:

```
item: "bread"
item: "apple"
item: "egg"
```

### **Closures**

#### **What is a Closure?**

A Groovy Closure is like a "code block" or a method pointer. It is a piece of code that is defined and then executed at a later point. It has some special properties like implicit variables, support for currying and support for free variables (which we'll see later on). We'll ignore the nitty gritty details for now (see the formal definition if you want those) and look at some simple examples.

#### **Simple Example**

```
def clos = { println "hello!" }

println "Executing the Closure:"
clos() //prints
"hello!"
```

Note that in the above example, "hello!" is printed when the Closure is *called*, not when it is defined.

### Parameters

Closure parameters are listed before the `->` token, like so:

```
def printSum = { a, b -> print a+b }
printSum( 5, 7 )
//prints "12"
```

The `->` token is optional and may be omitted if your Closure definition takes fewer than two parameters.

### Parameter notes

A Closure without `->`, i.e. `{ }`, is a Closure with one argument that is implicitly named as 'it'. (see below for details) In some cases, you need to construct a Closure with zero arguments, e.g. using [GString for templating](#), defining [EMC Property](#) etc. You have to explicitly define your Closure as `{ -> }` instead of just `{ }`

You can also use varargs as parameters, refer to the [Formal Guide](#) for details. A JavaScript-style dynamic args could be simulated, refer to the [Informal Guide](#).

### Free variables

Closures may refer to variables not listed in their parameter list. Such variables are referred to as "free" variables. They are "bound" to variables within the scope where they are defined:

```
def myConst = 5
def incByConst = { num -> num + myConst }
println incByConst(10) // => 15
```

Or another example:

```

def localMethod() {
    def localVariable = new java.util.Date()
    return { println localVariable }
}

def clos = localMethod()

println "Executing the Closure:"
clos() //prints the
date when "localVariable" was defined

```

### Implicit variables

Within a Groovy Closure, several variables are defined that have special meaning:

**it**

If you have a Closure that takes a single argument, you may omit the parameter definition of the Closure, like so:

```

def clos = { print it }
clos( "hi there" ) //prints "hi
there"

```

### **this**, **owner**, and **delegate**

**this** : as in Java, **this** refers to the instance of the enclosing class where a Closure is defined

**owner** : the enclosing object (**this** or a surrounding Closure)

**delegate** : by default the same as **owner**, but changeable for example in a [builder](#) or [ExpandoMetaClass](#)

Example:

```
class Class1 {
  def closure = {
    println this.class.name
    println delegate.class.name
    def nestedClos = {
      println owner.class.name
    }
    nestedClos()
  }
}
```

```
def clos = new Class1().closure
clos.delegate = this
clos()
/* prints:
  Class1
  Script1
  Class1$_closure1 */
```

### Closures as Method Arguments

When a method takes a Closure as the last parameter, you can define the Closure inline, like so:

```

def list = ['a', 'b', 'c', 'd']
def newList = []

list.collect( newList ) {
    it.toUpperCase()
}
println newList           // ["A", "B",
"C", "D"]

```

In the above example, the `collect` method accepts a `List` and a `Closure` argument. The same could be accomplished like so (although it is more verbose):

```

def list = ['a', 'b', 'c', 'd']
def newList = []

def clos = { it.toUpperCase() }
list.collect( newList, clos )

assert newList == ["A", "B", "C", "D"]

```

## More Information

Groovy extends `java.lang.Object` and many of the `Collection` and `Map` classes with a number of methods that accept `Closures` as arguments. See [GDK Extensions to Object](#) for practical uses of Groovy's `Closures`.

See Also:

- [Closures - Formal Definition](#)
- [Closures - Informal Guide](#)

## Closures - Formal Definition

### Formal Guide

A closure in Groovy is an anonymous chunk of code that may take arguments, return a value, and reference and use variables declared in its surrounding scope. In many ways it resembles anonymous inner classes in Java, and closures are often used in Groovy in the same way that Java developers use anonymous inner classes. However,

Groovy closures are much more powerful than anonymous inner classes, and far more convenient to specify and use.

In functional language parlance, such an anonymous code block might be referred to as an anonymous lambda expression in general or lambda expression with unbound variables or a closed lambda expression if it didn't contain references to unbound variables (like `threshold` in the earlier example). Groovy makes no such distinction.

Strictly speaking, a closure can't be defined. You can define a block of code that refers to local variables or fields/properties, but it becomes a closure only when you "bind" (give it a meaning) this block of code to variables. The closure is a semantic concept, like an instance, which you cannot define, just create. Strictly spoken a closure is only a closure if all free variables are bound. Unless this happens it is only partially closed, hence not really a closure. Since Groovy doesn't provide a way to define a closed lambda function and a block of code might not be a closed lambda function at all (because it has free variables), we refer to both as closure - even as syntactic concept. We are talking about it as syntactic concept, because the code of defining and creating an instance is one, there is no difference. We very well know that this terminology is more or less wrong, but it simplifies many things when talking about code in a language that doesn't "know" the difference.

### ***Syntax for Defining a Closure***

A closure definition follows this syntax:

```
{ [closureArguments->] statements }
```

Where `[closureArguments->]` is an optional comma-delimited list of arguments, and `statements` are 0 or more Groovy statements. The arguments look similar to a method's parameter list, and these arguments may be typed or untyped. When a parameter list is specified, the `->` character is required and serves to separate the arguments from the closure body. The *statements* portion consists of 0, 1, or many Groovy statements.

Some examples of valid closure definitions:

```

{ item++ }

{ println it }

{ ++it }

{ name -> println name }

{ String x, int y -> println "hey ${x} the
value is ${y}" }

{ reader ->
  while (true) {
    def line = reader.readLine()
  }
}

```

++++ Note: The examples could definitely be made more real-life MWS

### **Closure semantics**

Closures appear to be a convenient mechanism for defining something like an inner class, but the semantics are in fact more powerful and subtle than what an inner class offers. In particular, the properties of closures can be summarized in this manner:

1. They have one implicit method (which is never specified in a closure definition) called `doCall()`
2. A closure may be invoked via the `call()` method, or with a special syntax of an unnamed `()` invocation. Either invocation will be translated by Groovy into a call to the Closure's `doCall()` method.
3. Closures may have 1...N arguments, which may be statically typed or untyped. The first parameter is available via an implicit untyped argument named *it* if no explicit arguments are named. If the caller does not specify any arguments, the first parameter (and, by extension, *it*) will be null.
4. The developer does not have to use *it* for the first parameter. If they wish to use a different name, they may specify it in the parameter list.
5. Closures always return a value. This may occur via either an explicit *return* statement, or as the value of the last statement in the closure body (e.g. an explicit return statement is optional).
6. A closure may reference any variables defined within its enclosing lexical scope. Any such variable is said to be bound to the closure.
7. Any variables bound to a closure are available to the closure even when the closure is returned outside of the enclosing scope.

8. Closures are first class objects in Groovy, and are always derived from the class *Closure*. Code which uses closures may reference them via untyped variables or variables typed as *Closure*.
9. The body of a closure is not executed until it is explicitly invoked, e.g. a closure is not invoked at its definition time.
10. A closure may be *curried* so that a copy of the closure is made with one or more of its parameters fixed to a constant value.

These properties are explained further in the following sections.

#### Closures are anonymous

Closures in Groovy are always represented as anonymous blocks. Unlike a Java or Groovy class, you cannot have a named closure. You may however reference closures using untyped variables or variables of type *Closure*, and pass such references as method arguments and arguments to other closures.

#### Implicit method

Closures are considered to have one implicitly defined method, which corresponds to the closure's arguments and body. You cannot override or redefine this method. This method is always invoked by the `call()` method on the closure, or via the special unnamed `()` syntax. The implicit method name is `doCall()`.

#### Closure Arguments

A closure always has at least one argument, which will be available within the body of the closure via the implicit parameter *it* if no explicit parameters are defined. The developer never has to declare the *it* variable - like the *this* parameter within objects, it is implicitly available.

If a closure is invoked with zero arguments, then *it* will be null.

Explicit closure arguments may be specified by the developer as defined in the syntax section. These arguments are a list of 1 or more argument names which are comma separated. The parameter list is terminated with a `->` character. Each of these arguments may be specified "naked" e.g. without a type, or with an explicit static type. If an explicit parameter list is specified, then the *it* variable is not available.

For arguments that have a declared type, this type will be checked *at runtime*. If a closure invocation has 1 or more arguments which do not match the declared argument type(s), then an exception will be thrown at runtime. Note that this argument type checking always occurs at runtime; there is no static type checking involved, so the compiler will not warn you about mis-matched types.

Groovy has special support for excess arguments. A closure may be declared with its last argument of type *Object[]*. If the developer does this, any excess arguments at invocation time are placed in this array. This can be used as a form of support for variable numbers of arguments. For example:

```
def c = {
    format, Object[] args ->
    aPrintfLikeMethod (format, args)}
c ("one", "two", "three");
c ("1");
```

Both invocations of *c* are valid. Since the closure defines two arguments (*format* and *args*) and the last argument is of type *Object[]*, the first parameter in any call to *c* will be bound to the *format* argument and the remaining parameters will be bound to the *args* argument. In the first call of *c* the closure will receive the parameter *args* with 2 elements ("two", "three") while the *format* parameter will contain the string "one". In the second call the closure will receive the parameter *args* with no elements and the *format* parameter will contain the string "1".

++++ *What Exception is thrown? MWS*

#### **Closure Return Value**

Closures always have a return value. The value may be specified via one or more explicit *return* statement in the closure body, or as the value of the last executed statement if *return* is not explicitly specified. If the last executed statement has no value (for example, if the last statement is a call to a void method), then null is returned.

There is currently no mechanism for statically declaring the return type of a closure.

#### **References to External Variables**

Closures may reference variables external to their own definition. This includes local variables, method parameters, and object instance members. However, a closure may only reference those variables that the compiler can lexically deduce from the physical location of the closure definition within the source file.

Some examples might serve to clarify this. The following example is valid and shows a closure using a method's local variables and a method parameter:

```

public class A {
    private int member = 20;

    private String method()
    {
        return "hello";
    }

    def publicMethod (String name_)
    {
        def localVar = member + 5;
        def localVar2 = "Parameter: ${name_}";
        return {
            println "${member} ${name_}
${localVar} ${localVar2} ${method()}"
        }
    }
}

A sample = new A();
def closureVar =
sample.publicMethod("Xavier");
closureVar();

```

The above code will print out:

```

20 Xavier 25 Parameter: Xavier hello

```

Looking at the definition of class *A*, the closure inside of *publicMethod* has access to all variables that *publicMethod* may legally access. This is true whether the variables are local variables, parameters, instance members, or method

invocations.

When a closure references variables in this way, they are bound to the closure. At the same time, the variables are still available normally to the enclosing scope, so the closure may read/change any such values, and code from the outer scope may read/change the same variables.

If such a closure is returned from its enclosing scope, the variables bound with the closure also live on. This binding occurs when the closure is instantiated. If an object method or instance member is used within a closure, then a reference to that object is stored within the closure. If a local variable or parameter is referenced, then the compiler re-writes the local variable or parameter reference so that the local variable or parameter is taken off the stack and stored in an heap based object.

It's important to keep in mind that these references only are ever allowed according to the lexical structure available to the compiler (in this case, the A class). This process does *not* occur dynamically by looking at the call stack. So the following will not work:

```
class A {
    private int member = 20;

    private String method()
    {
        return "hello";
    }

    def publicMethod (String name_)
    {
        def localVar = member + 5;
        def localVar2 = "Parameter: name_";
        return {
            // Fails!
            println "${member} ${name_}
${localVar} ${localVar2} ${method()}
${bMember}"
        }
    }
}
```

```
class B {  
    private int bMember = 12;  
  
    def bMethod (String name_)  
    {  
        A aInsideB = new A();  
        return (aInsideB.publicMethod  
(name_));  
    }  
  
}
```

```
B aB = new B();  
closureVar = aB.bMethod("Xavier");  
closureVar();
```

The above code is similar to the first example, except that we now have a class *B* which dynamically instantiates an object of type *A* and then calls *A*.publicMethod(). However, in this code the closure within publicMethod() is trying to reference a member from *B*, and this is not allowed since the compiler cannot statically determine that this is available. Some older languages allowed this sort of reference to work, by dynamically examining the call stack at runtime, but this is disallowed in Groovy.

Groovy supports the special *owner* variable which can be used when a closure argument is hiding an object member variable. For example:

```
class HiddenMember {  
    private String name;  
  
    def getClosure (String name)  
    {  
        return { name -> println (name)}  
    }  
}
```

In the above code the *println (name)* call is referencing the parameter *name*. If the closure needs to access the name instance variable of class HiddenMember, it can use the owner variable to indicate this:

```
class HiddenMember {
    private String name;

    def getClosure (String name)
    {
        return { name -> println ("Argument:
${name}, Object: ${owner.name}") }
    }
}
```

#### The Closure Type

All closures defined in Groovy are derived from the type *Closure*. Each unique closure definition with a Groovy program creates a new unique class which extends *Closure*. If you wish to specify the type of a closure in a parameter, local variable, or object member instance, then you should use the *Closure* type.

The exact type of a closure is not defined unless you are explicitly subclassing the *Closure* class. Using this example:

```
def c = { println it }
```

The exact type of the closure referenced by *c* is not defined, we know only that it is a subclass of *Closure*.

#### Closure creation and invocation

Closures are created implicitly when their surrounding scope encounters them. For example, in the following code two closures are created:

```

class A {
    private int member = 20;

    private method()
    {
        println ("hello");
    }

    def publicMethod (String name_)
    {
        def localVar = member + 5
        def localVar2 = "Parameter: name_";
        return {
            println "${member} ${name_}
${localVar} ${localVar2} ${method()}"
        }
    }
}

A anA = new A();
closureVar = anA.publicMethod("Xavier");
closureVar();
closureVar2 = anA.publicMethod("Xavier");
closureVar2();

```

In the above example, *closureVar* holds a reference to a different closure object than *closureVar2*. Closures are always implicitly created in this manner - you cannot *new* a closure programmatically.

Closures may be invoked using one of two mechanisms. The explicit mechanism is to use the `call()` method:

```
closureVar.call();
```

You may also use the implicit nameless invocation approach:

```
closureVar();
```

If you are looking at the Closure javadoc, you may notice that the call method within the Closure class is defined as:

```
public Object call (Object[] args);
```

Despite this method signature, you do not have to manually write code to turn parameters into the Object[] array. Instead, invocations use the normal method argument syntax, and Groovy converts such calls to use an object array:

```
closure ("one", "two", "three")  
closure.call ("one", "two", "three")
```

Both calls above are legal Groovy. However, if you are dealing with a Closure from Java code you will need to create the Object[] array yourself 😊

#### Fixing Closure Arguments to Constant Values Via Currying

You can fix the values for one or more arguments to a closure instance using the `curry()` method from the *ClosureType*. In fact, this action is often referred to as currying in functional programming circles, and the result is generally referred to as a *Curried Closure*. Curried closures are very useful for creating generic closure definitions, and then creating several curried versions of the original with differing parameters bound to them.

When the `curry()` method is called on a closure instance with one or more arguments, a copy of the closure is first made. The incoming arguments are then bound permanently to the new closure instance so that the parameters 1..N to the `curry()` call are bound to the 1..N parameters of the closure. The new *curried closure* is then returned the caller.

Callers to the new instance will have their invocation parameters bound to the new closure in the N+1 parameter position of the original closure.

A simple example of this would be:

```
def c = { arg1, arg2-> println "${arg1}
${arg2}" }
def d = c.curry("foo")
d("bar")
```

The above code defines a closure *c*, and then calls *c.curry("foo")*. This returns a curried closure with the *arg1* value permanently bound to the value "foo". On the invocation *d("bar")*, the "bar" parameter comes into the closure in the *arg2* argument. The resulting output would be *foo bar*.

See also: [Functional Programming with Groovy](#)

#### Special Case: Passing Closures to Methods

Groovy has a special case for defining closures as method arguments to make the closure syntax easier to read. Specifically, if the last argument of a method is of type *Closure*, you may invoke the method with an explicit closure block outside of the parenthesis. For example, if a class has a method:

```
class SomeCollection {
    public void each (Closure c)
}
```

Then you may invoke *each()* with a closure definition outside of the parenthesis:

```
SomeCollection stuff = new SomeCollection();
stuff.each() { println it }
```

The more traditional syntax is also available, and also note that in Groovy you can elide parenthesis in many situations, so these two variations are also legal:

```

SomeCollection stuff = new SomeCollection();
    stuff.each { println it }      // Look ma,
no parens
    stuff.each ({ println it })   // Strictly
traditional

```

The same rule applies even if the method has other arguments. The only restriction is that the Closure argument must be last:

```

class SomeCollection {
    public void inject (x, Closure c)
    }

    stuff.inject(0) {count, item -> count +
item }      // Groovy
    stuff.inject(0, {count, item -> count +
item })     // Traditional

```

This syntax is only allowed when explicitly defining a closure within the method call. You cannot do this with a variable of type closure, as this example shows:

```

class SomeCollection {
    public void inject (x, Closure c)
    }

    counter = {count, item -> count + item }
    stuff.inject(0) counter      //
Illegal! No Groovy for you!

```

When you are not defining a closure inline to a method call, you cannot use this syntax and must use the more verbose syntax:

```

class SomeCollection {
    public void inject (x, Closure c)
    }

    def counter = {count, item -> count + item
    }

    stuff.inject(0,counter)

```

### *Comparing Closures to Anonymous Inner Classes*

Groovy includes closures because they allow the developer to write more concise and more easily understood code. Where Java developers may use single-method interfaces (Runnable, the Command pattern) combined with anonymous inner classes, Groovy allows you to accomplish the same sort of tasks in a less verbose manner. In addition, closures have fewer constraints than anonymous inner classes and include extra functionality.

Most closures are relatively short, isolated, and anonymous snippets of code that accomplish one specific job. Their syntax is streamlined to make closure definitions very short and easy to read without additional clutter. For example, in Java code you might see code like this for an imaginary GUI system:

```

Button b = new Button ("Push Me");
b.onClick (new Action() {
    public void execute (Object target)
    {
        buttonClicked();
    }
});

```

The same code in Groovy would look like this:

```

Button b = new Button ("Push Me");
b.onClick { buttonClicked() }

```

The Groovy code accomplishes the same task but is much clearer and without extra syntactical clutter. This is the first

rule of Groovy closures - closures are trivially easy to write. In addition, closures may reference any variables in its outer

defining scope without the restrictions of anonymous inner classes - in particular, such variables do not need to be final.

Closures also carry their state around with them, even when they reference local variables and parameters.

Closures may also take advantage of Groovy's optional dynamic typing so that you don't have to statically declare all of your closure arguments or return types (in fact, a Groovy closure can take varying numbers of parameters from invocation to invocation).

What Groovy closures lack compared to an approach using Command-like interfaces is the level of static typing involved. A Java interface rigidly enforces what type of objects can be used and the method(s) that may be called in it. In Groovy, all closures type equally as *Closure* and type checking of arguments (if specified in the closure definition) is deferred until Runtime.

### ***Closures as map keys and values***

It's possible to put closures in a map, both as keys and values.

#### **Closures as keys**

You can use a closure as a key. However, when putting it into the map you must "escape" it (as you would any other identifier you don't want treated as a string) by enclosing it in parens, like so:

```
f = { println "f called" }
m = [ (f): 123 ]
```

When accessing the value of the closure in the map you must use `get(f)` or `m[f]` as `m.f` will treat `f` as a string.

```
println m.get(f)    // 123
println m[f]       // 123
println m.f        // null
```

#### **Closures as values**

You can use a closure as a value and call that closure as if it were a method on the map, similarly to `Expandos`.

```
m = [ f: { println 'f called' } ]
m.f()      // f called

m = new Expando( f: { println 'f called' } )
m.f()      // f called
```

#### *Extending groovy with the use directive*

You can provide your own specialized methods supporting closures by implementing a Java class containing such methods. These methods must be static and contain at least two parameters. The *first* parameter to the method must be the type on which the method should operate, and the *last* parameter must be a Closure type.

Consider the example below, which is a variant of the eachFile method which simply ignores files, and just prints the directories within the dir object on which the method operates.

```
dir = new File("/tmp")
use(ClassWithEachDirMethod.class) {
    dir.eachDir {
        println it
    }
}
```

Take note of the use() directive. This will tell groovy where the eachDir method is implemented. Below is the Java code required to support the eachDir method on the File object, as shown.

```
public class ClassWithEachDirMethod {
    public static void eachDir(File self,
Closure closure) {
        File[] files = self.listFiles();
        for (int i = 0; i < files.length; i++)
    {
            if (files[i].isDirectory()) {
                closure.call(files[i]);
            }
        }
    }
}
```

To support additional parameters, these should be placed between the first and last.

## Closures - Informal Guide

### Informal Guide

When using the Java programming language most executable code is enclosed in either static class methods or instance methods. (Code can also be enclosed in constructors, initializers, and initialization expressions, but those aren't important here.) A method encloses code within curly brackets and assigns that block of code a method name. All such methods must be defined inside of a class of some type. For example, if you were to write a method that returned the square of any integer it may look like this:

```
package example.math;

public class MyMath {
    public static int square(int
numberToSquare){
        return numberToSquare *
numberToSquare;
    }
}
```

Now in order to use the square() method you need to reference the class and the method by name as follows:

```
import example.math.MyMath;
...
int x, y;
x = 2;
y = MyMath.square(x); // y will equal 4.
```

You can do the same thing in Groovy, but in groovy you can alternatively define the code without having to declare a class and a method as follows:

```
{ numberToSquare -> numberToSquare *
numberToSquare }
```

In Groovy, this anonymous code block is referred to as a **closure** definition (see the Formal Guide section below for a more elaborate definition of terms). A closure definition is one or more program statements enclosed in curly brackets. A key difference between a closure and method is that closures do not require a class or a method name.

As you can see, the executable code is the same except you didn't need to declare a class or assign the code a method name. While illustrative, the previous example is not all that useful because there is no way to use that closure once its created. It has no identifier (method name) so how can you call it? To fix that you assign the closure to a variable when it's created. You can then treat that variable as the identifier of the closure and make calls on it.

The following shows the square() method re-written as a closure:

```

def x = 2

// define closure and assign it to variable
'c'
def c = { numberToSquare -> numberToSquare *
numberToSquare }

// using 'c' as the identifier for the
closure, make a call on that closure
def y = c(x) // shorthand form for
applying closure, y will equal 4
def z = c.call(x) // longhand form, z will
equal 4

```

What is really nice about closures is that you can create a closure, assign it to a variable, and then pass it around your program like any other variable. At first this seems a bit, well useless, but as you learn more about Groovy you'll discover that closures are used all over the place.

As an example, let's extend the `java.util.Vector` class from Java by adding a single method that allows you to apply a closure to every element in the vector. My new class, `GVector`, looks as follows:

```

package example

public class GVector extends
java.util.Vector {
    public void apply( c ){
        for (i in 0..<size()){
            this[i] = c(this[i])
        }
    }
}

```

The `apply()` method takes a closure as an input parameter. For each element in the `GVector`, the closure is called passing in the element. The resulting value is then used to replace the element. The idea is that you can modify the contents of the `GVector` in place using a closure which takes each element and converts into something else.

Now we can call our new `apply()` method with any closure we want. For example, we will create a new `GVector`, populate it with some elements, and pass in the closure we created earlier, the one that squares an integer value.

```
import example  
def gVect = new GVector()  
gVect.add(2)  
gVect.add(3)  
gVect.add(4)  
  
def c = { numberToSquare -> numberToSquare * numberToSquare }  
  
gVect.apply(c) // the elements in the GVector have all been squared.
```

Because the `apply()` method on the `GVector` can be used with any closure, you can use any closure. For example, the following uses a closure that simply prints out the item its passed.

```
import example  
def gVect = new GVector()  
gVect.add(2)  
gVect.add(3)  
gVect.add(4)  
  
def c2 = { value -> println(value) }  
  
gVect.apply(c2) // the elements in the GVector have all been printed.
```

If you were to run the above script, assuming `GVector` from earlier is on your classpath, the output would look like this:

```
C: /> groovy myscript.groovy
4
9
16
C: />
```

In addition to assigning closures to variables, you can also declare them directly as arguments to methods. For example, the above code could be re-written in the following manner:

```
import example
def gVect = new GVector()
gVect.add(2)
gVect.add(3)
gVect.add(4)

gVect.apply{ value -> println(value) } //
elements in GVector have been printed.
```

This example accomplishes the same thing as the first, but the closure is defined directly as an argument to the `apply` method of `GVector`.

The other important difference of a closure to a normal method is that a closure can refer to variables from the scope in which it is called (in fact this is where this language construct gets its name). Here is an example:

```

class Employee {
    def salary
}
def highPaid(emps) {
    def threshold = 150
    return emps.findAll{ e -> e.salary >
threshold }
}

def emps = [180, 140, 160].collect{ val ->
new Employee(salary:val) }

println emps.size()           // prints 3
println highPaid(emps).size() // prints 2

```

In this example, the closure block { e -> e.salary > threshold } refers to the threshold variable defined in the highPaid() method. The example also used a closure to create the emps list.

#### Simulate Javascript-style variable arguments

Groovy supports varargs as Closure parameter, but that requires the use of a Object[] and the closure code has to access the varargs as an Array.

In JavaScript, function arguments are fully dynamic and you could call a function with any number of arguments different from the number of arguments defined in a function, e.g.

```
//javascript
function doSomething( var0, var1){
  alert( 'var0: ' + var0 + ', var1: ' +
var1);
}

doSomething( 'value0', 'value1', 'value2' );
//one argument more than in the defined
function
```

If your closure defined more argument than closure call, then you could use:

```
def doSomething = {var0, var1 = null -> }

doSomething( 'value0' )

doSomething( 'value0' , 'value1' )
```

However, you can't do the job exactly like a JavaScript function when your closure has defined less argument than the closure call. If you are the implementer of the closure, you could use varargs (as described in the [Formal Guide](#)) to allow your closure to take more arguments than it is defined, and you have to access the extra variables as an Object[].

There are cases you may want to take more arguments but not using varargs/Object[]. For example, as an API provider, you expose an API that take a closure as argument. The closure may define one or two parameters up to the user. (this is a typical case when passing JavaScript function) The following is an example about how to simulate such behavior:

```
// sample entity
class User{
  def username, password, version, salt =
  'RANDOM' ;
}
```

```

// your API, provide a Map of changes to
update a entity. the map value may be static
value, or a closure that take up to 2 params
def update( entity, Map changes ){
  changes?.each {k, v ->
    def newValue;
    if (v instanceof Closure){
      switch (v.parameterTypes.length) {
        case 0: newValue = v(); break;
        case 1: newValue = v(entity[k]); break;
// if one params, the closure is called with
the field value
        case 2: newValue = v(entity[k],entity);
break; // if two params, the closure is
called with teh field value and the entity
      }
    }else{
      newValue = v
    }
    entity[k] = newValue
  }
}

// user code
def user1 = new User(username:'user1',
password:'pass1', version:0)

```

```
update( user1, [password:{p,e-> Hash.md5(p,
e.salt) }, version:{v-> v+1 }] //assume
there is a MD5 util
```

#### Other Examples

- You could define a closure that take a closure as argument, and combine the use of other Groovy techniques to do a lot of things. See the [Closure, Category and JPA example](#)

#### *Closures vs. Code Blocks*

A closure looks a lot like a regular Java or Groovy code block, but actually it's not the same. The code within a regular code block (whether its a method block, static block, synchronized block, or just a block of code) is executed by the virtual machine as soon as it's encountered. With closures the statements within the curly brackets are not executed until the call() is made on the closure. In the previous example the closure is declared in line, but it's not executed at that time. It will only execute if the call() is explicitly made on the closure. This is an important differentiator between closures and code blocks. They may look the same, but they are not. Regular Java and Groovy blocks are executed the moment they are encountered; closures are only executed if the call() is invoked on the closure.

## Builders

#### *Tree Based Syntax*

Groovy has special syntax support for List and Maps. This is great because it gives a concise representation of the actual object being defined, so its easier to keep track of what a program or script is doing. But what about programs which contain arbitrary nested tree structures. Surely, they are the hardest ones to keep track of what is going on. Isn't that an area where syntactic help will be most beneficial?

The answer is definitely yes and Groovy comes to the party with its **builder** concept. You can use it for DOM-like APIs or Ant tasks or Jelly tags or Swing widgets or whatever. Each may have their own particular factory mechanism to create the tree of objects - however they can share the same builder syntax to define them - in a concise alternative to XML or lengthy programming code. [See How Builders Work](#)

#### *Example*

[Note: the syntax in some of these examples is slightly out-dated. See chapter 8 of [GINA](#) in the mean-time until these examples are updated.]

Here's an example:

```

def f = frame(size:[300,300], text:'My
Window') {
    label(bounds:[10,10,290,30], text:'Save
changes')
    panel(bounds:[10,40,290,290]) {
        button(text:'OK', action:{ save
close })
        button(text:'Cancel', action:{ close
})
    }
}

```

The above invokes a number of methods on the owner class using named-parameter passing syntax. Then the button method would create JButton etc. The {} is used to define a closure which adds its content to the newly created node. Also notice that the action parameter is passed as a closure - which is ideal for working with UI centric listeners etc.

Note that within the 'markup' you can embed normal expressions - i.e. this markup syntax is a normal part of the Groovy language. e.g.

```

def f = frame(text: calculateFieldNamefoo,
1234){

    // lets iterate through some map
    map = [1:"hello", 2:"there"]

    for e in map {
        labelname:e.value
        textfieldname:e.value
    }
}

```

Using this simple mechanism we can easily create any structured tree of data - or provide an event based model too. Note in Groovy you can just overload the invokeMethodName, arguments to have a simple polymorphic tree creation - such as for DOM is structures or Ant tasks or Jelly tags etc.

Here's an example of some HTML using some mixed content which is typically hard to do neatly in some markup languages

```

html {
  head {
    title"XML encoding with Groovy"
  }
  body {
    h1"XML encoding with Groovy"
    p"this format can be used as an
alternative markup to XML"

    / an element with attributes and
text content /
    ahref:'http://groovy.codehaus.org'
["Groovy"]

    / mixed content /
    p [
      "This is some",
      b"mixed",
      "text. For more see the",

    ahref:'http://groovy.codehaus.org'
["Groovy"],
      "project"
    ]
    p "some text"
  }
}

```

Finally here's an example of creating some name-spaced XML structure XSD...

```
def builder = NodeBuilder.newInstance()
def xmlns = new
groovy.xml.NamespaceBuilder(builder)

def xsd =
xmlns.namespace('http://www.w3.org/2001/XMLSchema
chema', 'xsd')

def root =
xsd.schema(xmlns:['foo':'http://someOtherNam
espace']) {
    annotation {
        documentation("Purchase order schema
for Example.com.")
        //documentation(xmlns=[xml.lang:'en'])
["Purchase order schema for Example.com."]
    }
    element(name:'purchaseOrder',
type:'PurchaseOrderType')
    element(name:'comment', type:'xsd:string')
    complexType(name:'PurchaseOrderType') {
        sequence {
            element(name:'shipTo',
type:'USAddress')
            element(name:'billTo',
type:'USAddress')
            element(minOccurs:'0', ref:'comment')
            element(name:'items', type:'Items')
        }
        attribute(name:'orderDate',
type:'xsd:date')
```

```

}
complexType(name: 'USAddress') {
  sequence {
    element(name: 'name',
type: 'xsd:string')
    element(name: 'street',
type: 'xsd:string')
    element(name: 'city',
type: 'xsd:string')
    element(name: 'state',
type: 'xsd:string')
    element(name: 'zip',
type: 'xsd:decimal')
  }
  attribute(fixed: 'US', name: 'country',
type: 'xsd:NMTOKEN')
}
complexType(name: 'Items') {
  sequence {
    element(maxOccurs: 'unbounded',
minOccurs: '0', name: 'item') {
      complexType {
        sequence {
          element(name: 'productName',
type: 'xsd:string')
          element(name: 'quantity') {
            simpleType {

restriction(base: 'xsd:positiveInteger') {
              maxExclusive(value: '100')
            }
          }
        }
      }
    }
  }
}

```

```

        }
    }
    element(name: 'USPrice',
type: 'xsd:decimal')
        element(minOccurs: '0',
ref: 'comment')
            element(minOccurs: '0',
name: 'shipDate', type: 'xsd:date')
                }
                attribute(name: 'partNum',
type: 'SKU', use: 'required')
            }
        }
    }
}
/* Stock Keeping Unit, a code for
identifying products */
simpleType(name: 'SKU') {
    restriction(base: 'xsd:string') {
        pattern(value: '\\d{3}-[A-Z]{2}')
    }
}

```

```
}  
  }  
}
```

There's a converter `org.codehaus.groovy.tools.xml.DomToGroovy` from XML to groovy markup so you can try out this new markup language on any XML documents you have already.

### ***Special cases***

To output elements or attributes with a '-' in their name, you need to quote the names. For example, to generate a web-app descriptor for a Servlet app:

```
def builder = new groovy.xml.MarkupBuilder()  
builder.'web-app' {  
    'display-name' 'My Web Application'  
}
```

generates:

```
<web-app>  
  <display-name>My Web  
Application</display-name>  
</web-app>
```

### ***Read from external variable***

Most builder examples are inline usage. To use a builder to build for an external variable, you may use:

```
class MyConfig{
    static nodes = {
        'first entry' ( key: 'value' )
    }
}

def result = new
YourBuilder().invokeMethod( 'rootNode' ,
MyConfig.nodes )
```

#### Related links

Andy Glover introduces builders through an [astronomical example](#)

#### How Builders Work

| *Work in Progress*

**This section I will discuss how builders work and how you can create your own builder. Builders are based on the [builder pattern](#) from the [GOF design pattern book](#). It provides a way to build your own [DSL](#) and represents a powerful concept in Groovy. Let me start by saying I am not an expert on the subject of Groovy and this is a good thing for the following reasons. I will take you through a step by step example from a novice point of view. Much of the Groovy documentation is very useful but you have to know a lot to get the most out of the pages. I am starting some pages to learn more about the power of Groovy and share that understanding to other novices like me. It is also my hope that this will encourage other novices to take up the challenge and write their own pages.**

#### *Things you need to know*

- Hierarchic structures - XML data in groovy

- Closures
- Method Handlers
- Properties
- Meta Programming

Let me also add that here are many examples of creating builders in Groovy, so I will use those examples to keep this presentation short.

I find it easier to understand concrete examples so I will start by using a standard builder MarkupBuilder; so let us see it in use:

## MarkupBuilder Example

```
// create a builder, (note: this is not in
one of the packages that are automatically
imported

def builder = new groovy.xml.MarkupBuilder()
// construct a builder step(1)

// create a simple xml markup

builder.stocks {
// step (2)
    stock(symbol: 'JAVA')
// step (3-1)
    stock(symbol: 'MSFT')
// step (3-2)
    stock(symbol: 'IBM' )
}

==== result output =====>
<stocks>
  <stock symbol='JAVA' />
  <stock symbol='MSFT' />
  <stock symbol='IBM' />
</stocks>
```

So what is going on here:

step(1) - we just create a **MarkupBuilder** and save it in the variable *builder*, pretty much standard Groovy

step(2) - we invoke the method **stocks** on *builder*

The first thing to note is that *builder* does not know the method **stocks** so in java for example the compiler will give an error. In a dynamic language, where these decisions are made at runtime, the builder will *missing method exception*.

WORK IN PROGRESS

## FactoryBuilderSupport

### FactoryBuilderSupport

[SwingBuilder](#) is one of the most used Groovy builders. It follows the standard structure of BuilderSupport but uses the concept of factories to build each node. Seeing that the concept was useful enough for other builders the basic implementation was taken out of SwingBuilder and FactoryBuilderSupport was born (and SwingBuilder was retrofitted of course). [How Builders Work](#)

The **Factory** interface is the basic building block, the builder will call the factory's methods at specific points during node building, let's see them in their invocation order:

- **Object newInstance( FactoryBuilderSupport builder, Object name, Object value, Map attributes )**  
**throws InstantiationException, IllegalAccessException**  
Responsible for creating the object that responds to the node 'name' and its called during builder.createNode
- **boolean onHandleNodeAttributes( FactoryBuilderSupport builder, Object node, Map attributes )**  
Gives the factory the ability to process the attributes as it may see fit with the option of stopping the builder to process them itself (by returning true).
- **void setParent( FactoryBuilderSupport builder, Object parent, Object child )**  
**void setChild( FactoryBuilderSupport builder, Object parent, Object child )**  
allows the factory to setup parent/child relationships.
- **boolean isLeaf()**  
Lets the builder know if the node allows for further nodes to be nested on the current node.
- **void onNodeCompleted( FactoryBuilderSupport builder, Object parent, Object node )**  
Is the last method called from the factories perspective, it will let you handle any cleanup the node may require.

But that's not everything FactoryBuilderSupport has to offer. The factories may require contextual information on the current node being built to do its work, onNodeCompleted may require information that it is only available when newInstance is invoked, or newInstance may need to inspect the parent to decide what is the best way to create the node, just to mention a few scenarios, that's why FactoryBuilderSupport enables the following helping methods:

Method	Description
Map getContext()	returns the context of the current node*
Map getParentContext()	returns the context of the parent of the current node**
Factory getCurrentFactory()	returns the factory that built the current node
Factory getParentFactory()	returns the factory of the parent of the current node (if any)
Object getParentNode()	returns the parent of the current node (if any)**

Object getCurrent()	returns the current node*
---------------------	---------------------------

\* **Note:** In the newInstance(...) method, since the "current" node has not yet been created, getCurrent() and getContext() will return the parent node or parent context of the node that is currently being constructed.

\*\* **Note:** In the newInstance(...) method, since the "current" node has not yet been created, getParentNode() and getParentContext() will return the grandparent node or grandparent context of the node that is currently being constructed.

The builder is marked as abstract so you are required to create a subclass for your own builders, despite that it doesn't enforce the implementation of any method at all. There are a couple of protected methods though, that when overwritten will give you more control over the builder's internal workings:

- **Factory resolveFactory( Object name, Map attributes, Object value )**

Usually what you would like in a subclass of FactoryBuilderSupport is a **1 to 1** relation on node names to factories, but for those cases where you would like **n to 1** you can override this method and plug in your custom selection mechanism.

- **void preInstantiate( Object name, Map attributes, Object value )**  
**void postInstantiate( Object name, Map attributes, Object node )**  
**void handleNodeAttributes( Object node, Map attributes )**  
**Object postNodeCompletion( Object parent, Object node )**

These methods are called during the lifecycle of a node, you can override them at any time but there is also a way to extend the behavior associated with those calls without overwriting the methods: you may register a closure, in fact as many as you like, to hook your own logic. The closures will be called from last to first as they were registered.

**TODO** document build() methods

## The @DelegatesTo annotation

### Groovy, DSLs and builders

#### *DSLs made easy*

The Groovy language is a platform of choice for building DSLs. Using closures, it's quite easy to create custom control structures, as well as it is simple to create builders. Imagine that you have the following code:

```
email {
  from 'dsl-guru@mycompany.com'
  to 'john.doe@waitaminute.com'
  subject 'The pope has resigned!'
  body {
    p 'Really, the pope has resigned!'
  }
}
```

One way of implementing this is using the builder strategy, which implies a method, named *email* which accepts a closure as an argument. The method may delegate subsequent calls to an object that implements the *from*, *to*, *subject* and *body* methods. Again, *body* is a method which accepts a closure as an argument and that uses the builder strategy.

Implementing such a builder is not complicated:

```
def email(Closure cl) {
  def email = new Email()
  def code = cl.rehydrate(email, this,
  this)
  code.resolveStrategy = DELEGATE_ONLY
  code()
}
```

the *Email* class implements the *from*, *to*, ... methods. By calling *rehydrate*, we're creating a copy of the closure for which we set the *delegate*, *owner* and *thisObject* values. Setting the *owner* and the "this" object is not very important here since we will use the *DELEGATE\_ONLY* strategy which says that the method calls will be resolved only against the delegate of the closure. Then, we're just calling the code and we're done!

#### ***Documentation is important***

One of the problems with the code that we've shown is that the user of the *email* method doesn't have any information about the methods that he's allowed to call inside the closure. The only possible information is from the method documentation. There are two issues with this: first of all, documentation is not always written, and if it is, it's not always available (javadoc not downloaded, for example). Second, it doesn't help IDEs. What would be really interesting, here, is for IDEs to help the developer by suggesting, once they are in the closure body, methods that exist on the *Email* class.

Moreover, if the user calls a method in the closure which is not defined by the *Email* class, the IDE should at least issue a warning (because it's very likely that it will break at runtime).

#### ***Type checking builders***

Another problem with the code that we've shown is that it is not compatible with static type checking. If you try to perform type checking on this code:

```

email {
  from 'dsl-guru@mycompany.com'
  to 'john.doe@waitamminute.com'
  subject 'The pope has resigned!'
  body {
    p 'Really, the pope has resigned!'
  }
}

```

Then the type checker will know that there's an *email* method accepting a *closure*, but it will complain for every method call **inside** the closure, because *from*, for example, is not a method which is defined in the class. Indeed, it's defined in the *Email* class and it has absolutely no hint to help it knowing that the closure delegate will, at runtime, be of type *Email*.

#### **@DelegatesTo**

For those reasons, Groovy 2.1 introduces a new annotation named `@DelegatesTo`. The goal of this annotation is to solve both the documentation issue, that will let your IDE know about the expected methods in the closure body, and it will also solve the type checking issue, by giving hints to the compiler about what are the potential receivers of method calls in the closure body.

The idea is to annotate the *Closure* parameter of the *email* method:

```

def email(@DelegatesTo(Email) cl) { ... }

```

What we've done here is telling the compiler (or the IDE) that when the method will be called with a closure, the delegate of this closure will be set to an object of type *Email*. But there is still a problem: the default delegation strategy is not the one which is used in our method. So we will give more information and tell the compiler (or the IDE) that the delegation strategy is also changed:

```

def
email(@DelegatesTo(strategy=Closure.DELEGATE
_ONLY, value=Email) cl) { ... }

```

Now, both the IDE and the type checker (if you are using `@TypeChecked`) will be aware of the delegate and the delegation strategy. This is very nice because it will both allow the IDE to provide smart completion, but it will also remove errors at compile time that exist only because the behaviour of the program is normally only known at

runtime!

## DelegatesTo modes

@DelegatesTo supports multiple modes that we will describe with examples in this section.

### Simple delegation

In this mode, the only mandatory parameter is the *value* which says to which class we delegate calls. Nothing more. We're telling the compiler that the type of the delegate will **always** be of the type documented by *@DelegatesTo* (note that it can be a subclass, but if it is, the methods defined by the subclass will not be visible to the type checker).

```
def build(@DelegatesTo(BuilderItem) cl) {  
  ... }  
}
```

### Delegation strategy

In this mode, you must specify both the delegate class **and** a delegation strategy. This must be used if the closure will not be called with the default delegation strategy, which is *Closure.OWNER\_FIRST*.

```
def  
build(@DelegatesTo(strategy=Closure.DELEGATE  
_FIRST, value=BuilderItem) { ... }  
}
```

### Delegate to parameter

In this variant, we will tell the compiler that we are delegating to another parameter of the method. Take the following code:

```
def exec(Object target, Closure code) {  
  def clone = code.rehydrate(target, this,  
this)  
  clone()  
}
```

Here, the delegate which will be used is **not** created inside the *exec* method. In fact, we take an argument of the method and delegate to it. Usage may look like this:

```

def email = new Email()
exec(email) {
  from '...'
  to '...'
  send()
}

```

Each of the method calls are delegated to the *email* parameter. This is a widely used pattern which is also supported by *@DelegatesTo* using a companion annotation:

```

def exec(@DelegatesTo.Target target,
@DelegatesTo code) { ... }

```

A closure is annotated with *@DelegatesTo*, but this time, without specifying any class. Instead, we're annotating another parameter with *@DelegatesTo.Target*. The type of the delegate is then determined at compile time. One could think that we are using the parameter type, which in this case is *Object* but this is not true. Take this code:

```

class Greeter {
  void sayHello() { println 'Hello' }
}
def greeter = new Greeter()
exec(greeter) {
  sayHello()
}

```

Remember that this works out of the box **without** having to annotate with *@DelegatesTo*. However, to make the IDE aware of the delegate type, or the **type checker** aware of it, we need to add *@DelegatesTo*. And in this case, it will now that the *greeter* variable is of type *Greeter*, so it will not report errors on the *sayHello* method **even if the exec method doesn't explicitly define the target as of type Greeter**. This is a very powerful feature, because it prevents you from writing multiple versions of the same *exec* method for different receiver types!

In this mode, the *@DelegatesTo* annotation also supports the *strategy* parameter that we've described upper.

### Multiple closures

In the previous example, the `exec` method accepted only one closure, but you may have methods that take multiple closures:

```
def guardWith(Closure guard, Closure code,  
Closure fallback) { ... }
```

Then nothing prevents you from annotating each closure with `@DelegatesTo`:

```
def guardWith(@DelegatesTo(Foo) Closure  
guard, @DelegatesTo(Bar) Closure code,  
@DelegatesTo(Baz) Closure fallback) { ... }
```

But more importantly, if you have multiple closures **and** multiple arguments, you can use several targets:

```
def guardWith(@DelegatesTo.Target('guarded')  
guarded,  
  
@DelegatesTo.Target('codeObject')  
codeObject,  
  
@DelegatesTo.Target('fallbackObject')  
fallbackObject,  
                @DelegatesTo(target='guarded')  
Closure guard,  
  
@DelegatesTo(target='codeObject') Closure  
code,  
  
@DelegatesTo(target='fallbackObject')  
Closure fallback) { ... }
```

At this point, you may wonder why we don't use the parameter names as references. The reason is that the information (the parameter name) is not always available (it's a debug-only information), so it's a limitation of the JVM.

## Statically compiled builders

### *Leveraging @DelegatesTo*

So far, we've only talked about static type checking (*@TypeChecked*) and IDE completion, but not about static compilation (*@CompileStatic*). So the question is whether *@CompileStatic* support it too (and the answer is of course yes!). In this section, we will show you how you can create a type-safe, statically compiled builder leveraging *@DelegatesTo*. As an example, we will use an *HtmlBuilder*, which is pretty much like a *MarkupBuilder*, apart from the fact that we will only allow some tags (for the sake of the example). Also make sure that we are not saying this is the **best** way to implement such a builder, we're just showing you the capabilities of *@DelegatesTo* with regards to static compilation. Let's start with the top level class:

```
import groovy.transform.CompileStatic

@CompileStatic
class HTMLBuilder {
    final StringBuilder sb = new
    StringBuilder()
    String getText() { sb.toString() }
}
```

The usage would be:

```
def builder = new HTMLBuilder()
builder.html {
    ...
}
```

So we miss the "html" method (other properties/methods removed for clarity):

```

@CompileStatic
class HTMLBuilder {

    public static void delegateToTag(Class
clazz, Closure body) {
        def tag = clazz.newInstance()
        def clone = body.rehydrate(tag,
null, null)
        clone()
    }

    void html(@DelegatesTo(HTMLTag) Closure
html) {
        delegateToTag(HTMLTag, html)
    }

    private static class HTMLTag {}
}

```

Here, we defined two methods:

- *delegateToTag* takes a class argument and a closure, then creates a new instance of this class. The closure is then cloned and its delegate is set to the tag and the closure is called.
- *html* tells that the closure will delegate to an HTML tag, so we create an empty (for now) HTML class

We have two problems to solve:

- make the tag builder aware of the stringbuilder so that we can generate our text
- make the tag itself able to support sub-tags

To solve the first issue, we will create an abstract base class for all tags:

```

@CompileStatic
abstract class Tag {
    protected final StringBuilder sb
    Tag(StringBuilder sb) { this.sb = sb }

    void openTag() {}
    void closeTag() {}

    public void delegateToTag(Class clazz,
Closure body) {
        Tag tag = (Tag) clazz.newInstance(sb)
        def clone = body.rehydrate(tag, this,
this)
        tag.openTag()
        clone()
        tag.closeTag()
    }
}

```

Then our HTMLTag just needs to extends that class:

```

@InheritConstructors
private static class HTMLTag extends Tag { }

```

Note that we also moved the *delegateToTag* method and removed the static modifier, so that we can pass the stringbuilder accross several tag instances, and it now calls *openTag* before calling the closure, and *closeTag* after. So now, the main builder looks like this:

```

@CompileStatic
class HTMLBuilder extends Tag {

    HTMLBuilder() { super(new
StringBuilder()) }

    void html(@DelegatesTo(HTMLTag) Closure
html) {
        delegateToTag(HTMLTag, html)
    }

    @InheritConstructors
    private static class HTMLTag extends Tag
    {}
}

```

The next step is to generate HTML:

```

@InheritConstructors
    private static class HTMLTag extends Tag
    {
        void openTag() { sb.append '<html>' }
        void closeTag() { sb.append '</html>' }
    }
}

```

Now what we will do is add a *head* and a *body* tag:

```

@InheritConstructors
    private static class HTMLTag extends Tag
    {
        void openTag() { sb.append '<html>' }
        void closeTag() { sb.append '</html>' }
    }

    void body(@DelegatesTo(BodyTag)
Closure body) {
        delegateToTag(BodyTag, body)
    }

    void head(@DelegatesTo(HeadTag)
Closure head) { delegateToTag(HeadTag, head)
}
}

@InheritConstructors
private static class BodyTag extends Tag
{
    void openTag() { sb.append '<body>' }
    void closeTag() { sb.append '</body>' }
}

}

@InheritConstructors
private static class HeadTag extends Tag
{
    void openTag() { sb.append '<head>' }
    void closeTag() { sb.append '</head>' }
}

}

```

Last but not least, let's make the body tag support a *p* tag:

```
void p(String body) { sb << "<p>$body</p>" }
```

Now let's see that we can statically compile the usage too:

```
@CompileStatic  
String test() {  
    def builder = new HTMLBuilder()  
    builder.html {  
        body {  
            p 'Hello, static builder!'  
        }  
    }  
    builder.text  
}  
test()
```

This prints:

```
<html><body><p>Hello, static  
builder!</p></body></html>
```

But one problem here is that our *p* tag doesn't support subtags... Can we make it support it too? Sure!

```

void p(@DelegatesTo(PTag) Closure p) {
    delegateToTag(PTag, p)
}

// ...
@InheritConstructors
static class PTag extends Tag {
    void openTag() { sb << "<p>" }
    void closeTag() { sb << "</p>" }
}

```

And here's how to use it:

```

builder.html {
    body {
        p 'Hello, static builder!'
        p {
            sb << 'Inside!' // sb works
too!
        }
    }
}

```

As you can see, here, inside the *p* tag, we're using *sb*. This is possible because *delegateToTag* sets the *owner* to *sb*. The static compiler recognizes it and is able to use it, without a single compilation error! So, let's put below the complete example:

```

import groovy.transform.CompileStatic
import groovy.transform.InheritConstructors
@CompileStatic

```

```
abstract class Tag {
    protected final StringBuilder sb
    Tag(StringBuilder sb) { this.sb = sb }

    void openTag() {}
    void closeTag() {}

    public void delegateToTag(Class clazz,
Closure body) {
        Tag tag = (Tag) clazz.newInstance(sb)
        def clone = body.rehydrate(tag, this,
this)
        tag.openTag()
        clone()
        tag.closeTag()
    }
}
@CompileStatic
class HTMLBuilder extends Tag {

    HTMLBuilder() { super(new
StringBuilder()) }

    void html(@DelegatesTo(HTMLTag) Closure
html) {
        delegateToTag(HTMLTag, html)
    }

    String getText() { sb }

    @InheritConstructors
```

```

private static class HTMLTag extends Tag
{
    void openTag() { sb.append '<html>' }
    void closeTag() { sb.append '</html>' }
}

void body(@DelegatesTo(BodyTag)
Closure body) {
    delegateToTag(BodyTag, body)
}

void head(@DelegatesTo(HeadTag)
Closure head) { delegateToTag(HeadTag, head) }
}

@InheritConstructors
private static class BodyTag extends Tag
{
    void openTag() { sb.append '<body>' }
    void closeTag() { sb.append '</body>' }
}

void p(String body) { sb << body }

void p(@DelegatesTo(PTag) Closure p)
{
    delegateToTag(PTag, p)
}
}

@InheritConstructors
private static class HeadTag extends Tag
{
    void openTag() { sb.append '<head>' }
    void closeTag() { sb.append '</head>' }
}

```

```
}  
  
}  
@InheritConstructors  
static class PTag extends Tag {  
    void openTag() { sb << "<p>" }  
    void closeTag() { sb << "</p>" }  
}  
  
}  
@CompileStatic  
String test() {  
    def builder = new HTMLBuilder()  
    builder.html {  
        body {  
            p 'Hello, static builder!'  
            p {  
                sb << 'Inside!'  
            }  
        }  
    }  
}
```

```
builder.text
}
test()
```

Congratulations, you've built your first statically compiled builder in Groovy!

## Groovy Truth

### Boolean expressions

Groovy supports the standard conditional operators on boolean expressions, e.g.:

```
def a = true
def b = true
def c = false
assert a
assert a && b
assert a || c
assert !c
```

In addition, Groovy has special rules for coercing non-boolean objects to a boolean value.

### Collections

Empty collections are coerced to false.

```
def numbers = [1,2,3]
assert numbers //true, as numbers is not
empty
numbers = []
assert !numbers //true, as numbers is now an
empty collection
```

### Iterators and Enumerations

Iterators and Enumerations with no further elements are coerced to false.

```
assert [].iterator() // false because the
Iterator is empty
assert [0].iterator() // true because the
Iterator has a next element
def v = new Vector()
assert !v.elements() // false because the
Enumeration is empty
v.add(new Object())
assert v.elements() // true because the
Enumeration has more elements
```

#### Maps

Non-empty maps are coerced to true.

```
assert ['one':1]
assert ![:]
```

#### Matchers

Matching regex patterns are coerced to true.

```
assert ('Hello World' =~ /World/) //true
because matcher has at least one match
```

#### Strings

Non-empty Strings, GStrings and CharSequences are coerced to true.

```
// Strings  
assert 'This is true'  
assert !''  
//GStrings  
def s = ''  
assert !("$s")  
s = 'x'  
assert ("$s")
```

#### Numbers

Non-zero numbers are coerced to true.

```
assert !0 //yeah, 0s are false, like in Perl  
assert 1 //this is also true for all other  
number types
```

#### Object references

Non-null object references are coerced to true.

```
assert new Object()  
assert !null
```

#### Input Output

Groovy provides a number of [helper methods](#) for working with I/O. All of these work with standard Java Reader/Writer and InputStream/OutputStream and File and URL classes.

The use of closures allows resources to be processed ensuring that things are properly closed irrespective of exceptions. e.g. to iterate through each line of a file the following can be used...

```
new File("foo.txt").eachLine { line ->  
println(line) }
```

If for whatever reason the `println()` method were to throw an exception, the `eachLine()` method ensures that the file resource is correctly closed. Similarly if an exception occurs while reading, the resource will be closed too.

If you wish to use a reader/writer object or an input/output stream object there are helper methods to handle the resource for you via a closure - which will automatically close down any resource if an exception occurs. e.g.

```
def count=0, MAXSIZE=100
new File("foo.txt").withReader { reader ->
    while (reader.readLine() != null) {
        if (++count > MAXSIZE) throw new
RuntimeException('File too large!')
    }
}
```

and

```
def fields = ["a":"1", "b":"2", "c":"3"]
new File("foo.ini").withWriter { out ->
    fields.each() { key, value ->
        out.writeLine("${key}=${value}")
    }
}
```

Further Information

- [Streams, Readers, and Writers](#) for more on Input and Output.
- See also: [Process Management](#)

## Collections

Groovy has native language support for collections, lists, maps and arrays.

### Lists

You can create lists as follows. Notice that `[]` is the empty list expression.

```
def list = [5, 6, 7, 8]
assert list.get(2) == 7
assert list[2] == 7
assert list instanceof java.util.List

def emptyList = []
assert emptyList.size() == 0
emptyList.add(5)
assert emptyList.size() == 1
```

Each list expression creates an implementation of [java.util.List](#).

See [Lists and Sets](#) for more information on using Lists.

## Ranges

Ranges allow you to create a list of sequential values. These can be used as Lists since [Range](#) extends [java.util.List](#).

Ranges defined with the `..` notation are inclusive (that is the list contains the from and to value).

Ranges defined with the `..<` notation are half-open, they include the first value but not the last value.

```

// an inclusive range
def range = 5..8
assert range.size() == 4
assert range.get(2) == 7
assert range[2] == 7
assert range instanceof java.util.List
assert range.contains(5)
assert range.contains(8)

// lets use a half-open range
range = 5..<8
assert range.size() == 3
assert range.get(2) == 7
assert range[2] == 7
assert range instanceof java.util.List
assert range.contains(5)
assert ! range.contains(8)

//get the end points of the range without
using indexes
def range = 1..10
assert range.from == 1
assert range.to == 10

```

Note that ranges are implemented efficiently, creating a lightweight Java object containing a from and to value.

Ranges can be used for any Java object which implements [java.lang.Comparable](#) for comparison and also have methods `next()` and `previous()` to return the next / previous item in the range.

e.g. you can use Strings in a range

```
// an inclusive range
def range = 'a'..'d'
assert range.size() == 4
assert range.get(2) == 'c'
assert range[2] == 'c'
assert range instanceof java.util.List
assert range.contains('a')
assert range.contains('d')
assert ! range.contains('e')
```

Ranges can be used to iterate using the `for` statement.

```
for (i in 1..10) {
    println "Hello ${i}"
}
```

but alternatively you can achieve the same effect, by iterating a range with *each* method:

```
(1..10).each { i ->
    println "Hello ${i}"
}
```

Ranges can be also used in the `switch` statements:

```
switch (years) {
    case 1..10: interestRate = 0.076; break;
    case 11..25: interestRate = 0.052; break;
    default: interestRate = 0.037;
}
```

## Maps

Maps can be created using the following syntax. Notice that `[:]` is the empty map expression.

Map keys are strings by default: `[a:1]` is equivalent to `["a":1]`. But if you really want a variable to become the key, you have to wrap it between parentheses: `[(a):1]`.

```
def map = [name:"Gromit", likes:"cheese",
id:1234]
assert map.get("name") == "Gromit"
assert map.get("id") == 1234
assert map["name"] == "Gromit"
assert map['id'] == 1234
assert map instanceof java.util.Map

def emptyMap = [:]
assert emptyMap.size() == 0
emptyMap.put("foo", 5)
assert emptyMap.size() == 1
assert emptyMap.get("foo") == 5
```

Maps also act like beans so you can use the property notation to get/set items inside the Map provided that the keys are Strings which are valid Groovy identifiers.

```
def map = [name:"Gromit", likes:"cheese",
id:1234]
assert map.name == "Gromit"
assert map.id == 1234

def emptyMap = [:]
assert emptyMap.size() == 0
emptyMap.foo = 5
assert emptyMap.size() == 1
assert emptyMap.foo == 5
```

Note: by design map.foo will always look for the key foo in map. This means foo.class will return null on an empty map and not result in calling the method getClass()

See [Maps](#) for more information on using maps.

#### Getting efficient with the star-dot '\*' operator

You can perform operations on all the members of a collection using the '\*' operator, e.g.:

```
assert [1, 3, 5] == ['a', 'few',
'words']*.size()
```

#### Enhanced Collection Methods

In addition to providing the literal syntax for collections, Groovy adds some additional methods to make working with collections more convenient. As an example, you can find big words from a list as follows:

```
def words = ['ant', 'buffalo', 'cat',
'dinosaur']
assert words.findAll{ w -> w.size() > 4 } ==
['buffalo', 'dinosaur']
```

Or you can find the first letters of some words as follows:

```
def words = ['ant', 'buffalo', 'cat',  
            'dinosaur']  
assert words.collect{ it[0] } == ['a', 'b',  
            'c', 'd']
```

In addition to `findAll` and `collect` shown above, you have methods like `findIndexOf`, `grep`, `any`, `every`, `min`, `max`, `flatten`, `intersect`, `disjoint`, `sort`, `join` and others. Simply look up the [GDK doco](#) for more details. You might want to look up the added methods for `Collection`, `List` and `Object` to start with.

Some more details about these methods can also be found in the Quick Start Guide under [JN1015-Collections](#).

### **Slicing with the subscript operator**

You can index into `Strings`, `Lists`, `arrays`, `Maps`, `regexs` and such like using the subscript expression.

```
def text = "nice cheese gromit!"
def x = text[2]

assert x == "c"
assert x.class == String

def sub = text[5..10]
assert sub == 'cheese'

def map = [name:"Gromit", likes:"cheese",
id:1234]

assert map["name"] == "Gromit"
assert map.name == "Gromit"

def list = [10, 11, 12]
def answer = list[2]
assert answer == 12
```

Notice that you can use ranges to extract part of a List/array/String/regex. This is often referred to as *slicing* in scripting languages like Python. You can also use a list of indexes too.

```
def list = 100..200
def sub = list[1, 3, 20..25, 33]
assert sub == [101, 103, 120, 121, 122, 123,
124, 125, 133]
```

You can update items using the subscript operator too

```
def list = ["a", "b", "c"]
list[2] = "d"
list[0] = list[1]
list[3] = 5
assert list == ["b", "b", "d", 5]
```

You can use negative indices to count from the end of the List, array, String etc.

```
def text = "nice cheese gromit!"
def x = text[-1]
assert x == "!"

def name = text[-7..-2]
assert name == "gromit"
```

Also if you use a backwards range (the starting index is greater than the end index) then the answer is reversed.

```
def text = "nice cheese gromit!"
def name = text[3..1]
assert name == "eci"
```

### Dynamic objects (Expandos)

The Expando is not a collection in the strictest sense, but in some ways it is similar to a Map, or objects in JavaScript that do not have to have their properties defined in advance. It allows you to create dynamic objects by making use of Groovy's [closure mechanisms](#). An [Expando](#) is different from a map in that you can provide synthetic methods that you can call on the object.

```

def player = new Expando()
player.name = "Dierk"
player.greeting = { "Hello, my name is
$name" }

println player.greeting()
player.name = "Jochen"
println player.greeting()

```

The `player.greeting` assignment passes in a [closure](#) to execute when `greeting()` is called on the [Expando](#). Notice that the closure has access to the properties assigned to the [Expando](#), even though these values may change over time, using Groovy's [GString](#) "\$variableOrProperty" notation.

## Regular Expressions

Groovy supports regular expressions natively using the `~"pattern"` expression, which creates a compiled Java Pattern object from the given pattern string. Groovy also supports the `=~` (create Matcher) and `==~` (returns boolean, whether String matches the pattern) operators.

For matchers having groups, `matcher[index]` is either a matched String or a List of matched group Strings.

```

import java.util.regex.Matcher
import java.util.regex.Pattern

// ~ creates a Pattern from String
def pattern = ~/foo/
assert pattern instanceof Pattern
assert pattern.matcher("foo").matches()
// returns TRUE
assert pattern.matcher("foobar").matches()
// returns FALSE, because matches() must
match whole String

```

```
// =~ creates a Matcher, and in a boolean
context, it's "true" if it has at least one
match, "false" otherwise.
```

```
assert "cheesecheese" =~ "cheese"
```

```
assert "cheesecheese" =~ /cheese/
```

```
assert "cheese" == /cheese/ /*they are
both string syntaxes*/
```

```
assert ! ("cheese" =~ /ham/)
```

```
// ==~ tests, if String matches the pattern
```

```
assert "2009" ==~ /\d+/ // returns TRUE
```

```
assert "holla" ==~ /\d+/ // returns FALSE
```

```
// lets create a Matcher
```

```
def matcher = "cheesecheese" =~ /cheese/
```

```
assert matcher instanceof Matcher
```

```
// lets do some replacement
```

```
def cheese = ("cheesecheese" =~
/cheese/).replaceFirst("nice")
```

```
assert cheese == "nicecheese"
```

```
assert "color" ==
```

```
"colour".replaceFirst(/ou/, "o")
```

```
cheese = ("cheesecheese" =~
```

```
/cheese/).replaceAll("nice")
```

```
assert cheese == "nicenice"
```

```
// simple group demo
```

```
// You can also match a pattern that
```

```
includes groups. First create a matcher
```

```

object,
// either using the Java API, or more simply
with the =~ operator. Then, you can index
// the matcher object to find the matches.
matcher[0] returns a List representing the
// first match of the regular expression in
the string. The first element is the string
// that matches the entire regular
expression, and the remaining elements are
the strings
// that match each group.
// Here's how it works:
def m = "foobarfoo" =~ /o(b.*r)f/
assert m[0] == ["obarf", "bar"]
assert m[0][1] == "bar"

// Although a Matcher isn't a list, it can
be indexed like a list. In Groovy 1.6
// this includes using a collection as an
index:

matcher = "eat green cheese" =~ "e+"

assert "ee" == matcher[2]
assert ["ee", "e"] == matcher[2..3]
assert ["e", "ee"] == matcher[0, 2]
assert ["e", "ee", "ee"] == matcher[0, 1..2]

matcher = "cheese please" =~ /([^e]+)e+/
assert ["se", "s"] == matcher[1]
assert [["se", "s"], [" ple", " pl"]] ==

```

```

matcher[1, 2]
assert [["se", "s"], [" ple", " pl"]] ==
matcher[1 .. 2]
assert [["chee", "ch"], [" ple", " pl"],
["ase", "as"]] == matcher[0, 2..3]
// Matcher defines an iterator() method, so
it can be used, for example,
// with collect() and each():
matcher = "cheese please" =~ /([^e]+)e+/
matcher.each { println it }
matcher.reset()
assert matcher.collect { it } ==
                [["chee", "ch"], ["se",
"s"], [" ple", " pl"], ["ase", "as"]]
// The semantics of the iterator were
changed by Groovy 1.6.
// In 1.5, each iteration would always
return a string of the entire match,
ignoring groups.
// In 1.6, if the regex has any groups, it
returns a list of Strings as shown above.

// there is also regular expression aware
iterator grep()
assert ["foo", "moo"] == ["foo", "bar",
"moo"].grep(~/.*oo$/)
// which can be written also with findAll()

```

## method

```
assert ["foo", "moo"] == ["foo", "bar",  
"moo"].findAll { it ==~ /.*oo/ }
```

Since a `Matcher` coerces to a boolean by calling its `find` method, the `==~` operator is consistent with the simple use of Perl's `==~` operator, when it appears as a predicate (in 'if', 'while', etc.). The "stricter-looking" `===~` operator requires an exact match of the whole subject string. It returns a `Boolean`, not a `Matcher`.

Regular expression support is imported from Java. Java's regular expression language and API is documented [here in the Pattern JavaDocs](#).

## More Examples

**Goal:** Capitalize words at the beginning of each line:

```
def before=''  
apple  
orange  
y  
banana  
'''  
  
def expected=''  
Apple  
Orange  
Y  
Banana  
'''  
  
assert expected ==  
before.replaceAll(/(?m)^\w+/,  
    { it[0].toUpperCase() + ((it.size() > 1)  
    ? it[1..-1] : '') })
```

**Goal:** Capitalize every word in a string:

```
assert "It Is A Beautiful Day!" ==
    ("it is a beautiful
day!".replaceAll(/\w+/,
    { it[0].toUpperCase() + ((it.size()
> 1) ? it[1..-1] : '') })))
```

Add `.toLowerCase()` to make the rest of the words lowercase

```
assert "It Is A Very Beautiful Day!" ==
    ("it is a VERY beautiful
day!".replaceAll(/\w+/,
    { it[0].toUpperCase() + ((it.size()
> 1) ? it[1..-1].toLowerCase() : '') })))
```

### Gotchas

How to use backreferences with `String.replaceAll()`

GStrings do **not** work as you'd expect:

```
def replaced = "abc".replaceAll(/(a)(b)(c)/,
"$1$3")
```

Produces an error like the following:

[] illegal string body character after dollar sign:

solution: either escape a literal dollar sign `"\$5"` or bracket the value expression `"${5}"` @ line []

Solution:

Use `'` or `/` to delimit the replacement string:

```
def replaced = "abc".replaceAll( / (a) (b) (c) / ,
' $1$3 ' )
```

## Documenting Regular Expressions in Groovy

Here are some suggestions for using regular expressions in Groovy. This page mainly focuses on documenting regular expressions in Groovy, however, those suggestions are applicable to any programming language that supports regular expressions such as Perl and Java.

*Note:* I couldn't get boldface and colors to work in code-listings under this wiki. This page was originally developed in the MediaWiki at [https://www.ngdc.noaa.gov/wiki/index.php?title=Regular\\_Expressions\\_in\\_Groovy](https://www.ngdc.noaa.gov/wiki/index.php?title=Regular_Expressions_in_Groovy) and the formatting looks better there.

### Reference Links

Here are some useful reference links that you may want to open up along side this page:

- [Groovy Regular Expressions](#)
- [java.util.regex.PatternAPI](#)
- [java.util.regex.MatcherAPI](#)
- [PLEAC Pattern Matching](#), PLEAC is Programming Language Examples Alike Cookbook and serves many programming languages

### Documenting the RegEx

It's important to document any regular expression, or "regex" for short, that is more than a trivial match. Documenting regexes is the key to making them understandable so they can be debugged and modified either by someone else or by you after you've had time to forget the details.

#### Overview

- **Include a sample of text to match**
  - give a plain English description of your goal
  - omit excess lines of sample if long
- **Use extended patterns with comments**
  - mark capturing groups by number
  - include "landmark" keys in the pattern
- **Include debugging feedback**
  - use debugging lines

#### Include a sample of text to match

Having a sample of the input that the regular expression is being applied to look at right on screen is a great help in deciphering what the pattern is trying to match. This is best done in a block comment before the regular expression pattern is defined.

- give a plain English description of your goal
  - note "landmark" keys in the pattern that you rely on to reliably parse the data
  - list any sub-parts (captured groups) of the pattern you wish to use after the match
- omit excess lines of sample if long

For example, on a system that has remotely mounted disk space with names like "/nfs/data" or "/nfs/DATA" we wish to gather the space free in kilobytes and the name on which the space is mounted. The output from the "df -k" (disk free space in kilobytes, on linux/mac/unix systems) could be parsed by this pattern:

```
pattern =  
~/(?i)(\d+)\s+\d+%\s+(\/nfs\/data.*)/
```

The "(?i)" is a match flag that means the pattern is case insensitive. The digits followed immediately by "%" is the key "landmark" in this regex, in all the output of the df this sequence is always preceded by the available space, regardless of whether the entry is remote and split across two lines or local and contained in one line.

To summarize the parts (regular expression constructs) here:

- **(\d+)** - One or more digits, captured for later use, "+" means 1 or more repetitions, see the [Pattern API](#)
- **\s+** - One or more whitespace characters
- **\d+%** - One or more digits followed by "%", the percentage of disk used
- **(\nfs\data.\*)** - look for a partition name that starts out with "/nfs/data"
  - **\** - A literal "/", escaped by "\" since a slash by itself starts or ends the pattern
  - **.\*** - Matches 0 or more characters, "." is a wildcard, "\*" means 0 or more repetitions

Following the suggestions above, a header comment is added:

```
/*  
    The input string to parse is from the  
output of the 'df -k' command.  
    Example input:  
  
        Filesystem                    1K-blocks  
Used Available Use% Mounted on  
        /dev/sda1                    4185956  
1206996    2762888    31% /  
        /dev/sda11                   30679784  
28324040    772140    98% /extra  
<..lines omitted..>  
        fas3050c-1-2.b:/vol/canis  
                                    10485760  
6599936    3885824    63% /nfs/data_d2/dog_data  
        fas6070-1-1.b:/vol/felis
```

314572800

54889600 259683200 18% /nfs/DATA-1/cat\_data

We want the available disk space in KB, for /nfs/data remote

disk partitions, which is "3885824" and "259683200" in the sample

above. Note that partitions that start out with "/nfs/data" may be

either upper or lower case. Capture the partition name for debugging.

Capture the space available in KB as the number before the

percentage number (digits followed by '%'). The pattern match

is relying on there being just one instance of a numeric

percentage (\d+%) occurring in the each usable output line.

The header line contains a "%", but the characters preceding it

are non-numeric, "Use%", so it is ignored. Also, many of the lines

we want are split across two lines, but it is only the second line

that contains the information we want.

The "(?i)" match flag indicates the pattern is case insensitive.

```
* /  
pattern =  
~/(?i)(\d+)\s+\d+%\s+(\nfs\data.*)/
```

#### Use extended patterns with comments

The **extended match mode** is enabled by a pattern match flag which allows white space and comments to be embedded into the pattern. You can then describe, piece by piece, the parts of the regular expression without dumping those details into the already large header comment suggested above. [Pattern match flags](#) are discussed in more detail later in this document.

In Groovy, this match flag is "(?x)" and can be combined with other flags you wish to turn on such as "(?ix)" for both extended and case-insensitive modes. This is done in conjunction with Groovy "here" documents (triple quoting), which is handled somewhat differently than the "slashed" quoting used for regular expression patterns. The three examples below are equivalent, but I've highlighted in **red** what is removed from the first, and colored **green** the new text in the second and third examples. (Note: the original author didn't figure out how to get the red and green color to work in the code listings for this wiki. If anyone knows how to fix this, please do.)

```
// 1. slashy regex  
pattern =  
~/(?i)(\d+)\s+\d+%\s+(\nfs\data.*)/
```

```
// 2. string converted to a regex  
regex =  
"(?i)(\d+)\s+\d+%\s+(\nfs/data.*)"  
pattern = ~regex
```

```
// 3. here document string converted to a  
regex  
regex =  
'''(?ix)(\d+)\s+\d+%\s+(\nfs/data.*)'''  
pattern = ~regex
```

Essentially, when converting a slashy regex to a string based pattern:

- Forward slashes don't need to be escaped by back slashes so "\\" becomes "/"
- Double the remaining back slashes. Back slashes need to be escaped by back slashes when quoting strings (either normal or here documents)
- If you want to match whitespace, then you must use

```
\\s
```

- You can match "#" with

```
\\#
```

so that it's not interpreted as the beginning of a comment

What does the third example buy you? Now newlines and comments can be included. The third example (here document) above can also be written:

```
// 3. here document string converted to a  
regex  
regex = '''(?ix)      # comments are now  
allowed!  
                (\\d+)  # disk space  
                \\s+  
                \\d+%   # one or more numbers  
followed by "%"  
                \\s+  
                (/nfs/data.*) # partition  
name '''  
pattern = ~regex
```

This allows you to

- **mark capturing groups by number**
  - I mark these with a numbered comment like "# 1: The disk space we want"
- **explain "landmark" keys in the pattern**
  - For example

```
\\d%   # a number followed by %
```

Not every line needs a comment, but don't leave out any important key matches.

Expanding on the example above:

```
/*
    The input string to parse is from the
    output of the 'df -k' command.
    Example input:

        Filesystem                1K-blocks
Used Available Use% Mounted on
        /dev/sda1                  4185956
1206996    2762888    31% /
        /dev/sda11                 30679784
28324040    772140    98% /extra
<..lines omitted..>
        fas3050c-1-2.b:/vol/canis
                                10485760
6599936    3885824    63% /nfs/data_d2/dog_data
        fas6070-1-1.b:/vol/felis
                                314572800
54889600 259683200    18% /nfs/DATA-1/cat_data
```

We want the available disk space in KB,  
for /nfs/data remote  
disk partitions, which is "3885824" and  
"259683200" in the sample  
above. Note that partitions that start  
out with "/nfs/data" may be  
either upper or lower case. (#2:)  
Capture the partition name for debugging.

(#1:) Capture the space available in KB as the number before the percentage number (digits followed by '%'). The pattern match is relying on there being just one instance of a numeric percentage (\d+%) occurring in the each usable output line.

The header line contains a "%", but the characters preceding it are non-numeric, "Use%", so it is ignored. Also, many of the lines we want are split across two lines, but it is only the second line that contains the information we want.

The "(?i)" match flag indicates the pattern is case insensitive.

The extended mode (?x) allows whitespace and comments starting with "#" to be embedded in the regular expression.

\*/

```
regex = '''(?ix)           # case insensitive,
extended format
        (\\d+)           # 1: The disk space
we want
        \\s+            # some whitespace
```

```
by %          \\d+%          # a number followed
whitespace   \\s+           # some more
              (/nfs/data.*)   # 2: partition
```

```
name ' '
```

```
pattern = ~regex
```

If that's not any easier to understand than what we started out with,

```
pattern =
```

```
~/ (?i) (\d+) \s+ \d+ % \s+ (\/nfs\/data.*) /
```

I'll just assume you're the sort of person who never reads code comments.

Include debugging feedback

- use debugging lines
  - they are easy to turn on/off with a flag variable
  - they verify the regular expression is working

While developing regular expressions, you will probably want to be able to easily test the result. An easy option is to add debugging lines. The debugging lines can be controlled by a boolean flag to turn them on or off. For little development programs and snippets in the groovyConsole, this is easier than setting up logging. The example above can be expanded with a 'debugging' flag and debugging lines like this:

```
/*
```

```
    The input string to parse is from...
```

```
    <...the rest of the header comment from
```

```
above...>
```

```
*/
```

```
boolean debugging = true
```

```
if (debugging) {
```

```
    // Test data
```

```
    dfkOutput = ' '
```

```
Filesystem
```

```
1K-blocks
```

```
Used
```

```
Available Use% Mounted on
```

```
/dev/sda1
```

```
4185956
```

```
1206996
```

```
2762888 31% /
```

```

/dev/sda1          30679784  28324040
772140  98% /extra
fas3050c-1-2.b:/vol/canis
                    10485760   6599936
3885824  63% /nfs/data_d2/dog_data
fas6070-1-1.b:/vol/felis
                    314572800  54889600
259683200  18% /nfs/DATA-1/cat_data
' ' '
} else {
    // Real data
    dfkOutput = 'df -k'.execute().text
}

long kbAvail = 0

regex = '''(?ix)      # enable
case-insensitive matches, extended patterns
        (\\d+)      # 1: The disk space
we want
        \\s+       # some whitespace
        \\d+%      # a number followed
by %
        \\s+       # some more
whitespace
        (/nfs/data.*) # 2: partition
name ' ' '

pattern = ~regex
matcher = pattern.matcher(dfkOutput)

```

```

if (debugging) {
    println ""matcher pattern:
/-----\\
${matcher.pattern()}
\\-----/" ""

    println "match
count=${matcher.getCount()}"
}

for (i=0; i < matcher.getCount(); i++) {
    if (debugging) {
        println "    text matched in
matcher[${i}]: '" + matcher[i][0] + "'"
        println "    free space in
(group 1): '" + matcher[i][1] + "'"
        println "    partition name
(group 2): '" + matcher[i][2] + "'"
    }
    kbAvail += matcher[i][1].toLong()
}

```

```
}
```

```
println "KB available=${kbAvail}"
```

With 'debugging = true', this prints some information to show the regular expression is working on the test data:

```

matcher pattern:
/-----\
(?ix)      # enable case-insensitive
matches, extended patterns
           (\d+)      # 1: The disk space we
want
           \s+        # some whitespace
           \d+%       # a number followed by
%
           \s+        # some more whitespace
           (/nfs/data.*) # 2: partition
name
\-----/
match count=2
    text matched in matcher[0]: '3885824
63% /nfs/data_d2/dog_data'
    free space in (group 1): '3885824'
    partition name (group 2):
'/nfs/data_d2/dog_data'
    text matched in matcher[1]: '259683200
18% /nfs/DATA-1/cat_data'
    free space in (group 1):
'259683200'
    partition name (group 2):
'/nfs/DATA-1/cat_data'
KB available=263569024

```

You can see by the output above that most of the input is ignored because it doesn't meet the described pattern. For those entries that are split across two lines, it turns out that all the information we want is in the second line, which still meets the pattern criteria, and the first line is ignored for not matching.

And if you set 'debugging = false', only the result is printed:

**KB available=263569024**

### Pattern Match Flags

The Java regular expression support includes many options modeled after Perl, which is one of the strongest regular expression parsing languages. Since Groovy gets its regular expression capability from Java (which copied Perl), what works in Java applies equally well to Groovy. Looking at the Java [java.util.regex.PatternAPI](#) we see that there is support for pattern match flags under the section called "Special constructs (non-capturing)." Specifically the line indicating

| (?idmsux-idmsux) Nothing, but turns match flags on - off

These capture nothing, but activate the match flags "idmsux". These correspond mostly to similarly named flags in Perl:

## Match Flags

Flag	Java/Groovy	Perl	Description
<b>i</b>	CASE_INSENSITIVE	ignore case	Do case insensitive pattern matching
<b>d</b>	UNIX_LINES	<i>not in Perl</i>	Enables Unix lines mode, only '\n' line terminator affects ., ^ and \$
<b>m</b>	MULTILINE	multiline	Enables multiline mode. In multiline mode the expressions ^ and \$ match just after or just before, respectively, a line terminator or the end of the input sequence. By default these expressions only match at the beginning and the end of the entire input sequence.

<b>s</b>	DOTALL	single line	In Perl this is called Single-line mode, treating the input as as single line even if it includes line terminators. Normally the "." wildcard doesn't match line terminators, but in Dotall mode it matches all characters.
<b>u</b>	UNICODE_CASE	<i>not in Perl</i>	Enables Unicode-aware case folding.
<b>x</b>	COMMENTS	eXtended	Extended mode allows whitespace, including newlines, and comments beginning with "#" and ending with a newline. Since whitespace in the pattern is ignored, use '\s' to indicate whitespace you wish to match.
<b>g</b>	<i>automatic in Java/Groovy</i>	global	Global matching keeps track of a current position in the input, so you can step through each place the pattern matches the input. Groovy sets up a result array that is an array of arrays of strings. The first dimension represents the number of matches, the second dimension contains the substrings that represent actual text matched and any captured groups.

**Example usage of Match Flags**

This is how the Match Flags may be used in **Groovy**. Below is a pattern that will grab the disk space value from the output of the unix "df" (disk freespace) command, but only on lines containing "/nfs/data", the "(?i)" indicates a case insensitive match:

```
pattern =
\~/ (?i) (\d+) \s+ \d+% \s+ (\/nfs\/data.* ) /
```

Note that the "(?i)" match flag comes at the beginning and is wrapped in parentheses. The "?" indicates this is a non-capturing group. Normally anything in a regular expression you wish to capture is wrapped in parentheses. Assuming the match over all succeeded, this "captured" group is available for later use as a substring that matched the portion of the regular expression in the parentheses.

For comparison, here is the equivalent pattern in **Perl**:

```
$dfkOutput = \~  
/(\d+)\s+\d+%\s+(\nfs\data.*)/i
```

Notice that the "i" indicating a case-insensitive match is appended at the end of the pattern.

#### **The eXtended Pattern Match Flag (x)**

The key flag to making regular expressions more readable is the "x" flag, which meant "eXtended" in Perl, and in Java/Groovy refers to "Comments," in a less mnemonic way.

Although Java and Groovy use the [java.util.regex.PatternAPI](#), which was modeled after Perl regular expressions, the extended mode wasn't of much use in Java. You *could* write

```

// Java regex with extended format
String patternStr =
    "(?ix)                # case
insensitive, extended format\n" +
    "(\\d+)                # 1: The
disk space we want\n" +
    "\\s+                  # some
whitespace\n" +
    "\\d+%                 # a number
followed by %\n" +
    "\\s+                  # some more
whitespace\n" +
    "(/nfs/data.*)        # 2:
partition name";

Pattern pattern =
Pattern.compile(patternStr);
Matcher m = pattern.matcher(dfOutput);

```

But it would be a little less typing to simulate this with ordinary comments:

```

// Java regex with comments simulating
extended format
String patternStr =
    "(?i)" + // case
insensitive
    "(\\d+)" + // 1: The
disk space we want
    "\\s+" + // some
whitespace
    "\\d+%" + // a number
followed by %
    "\\s+" + // some more
whitespace
    "(/nfs/data.*)"; // 2:
partition name

Pattern pattern =
Pattern.compile(patternStr);
Matcher m = pattern.matcher(dfOutput);

```

However, in Groovy a regex combined with a here document, this becomes much cleaner (and more Perl-like):

```

// Groovy regex
regex = '''(?ix)          # enable
case-insensitive matches, extended patterns
        (\\d+)           # 1: The disk
space we want
        \\s+             # some whitespace
        \\d+%           # a number
followed by %
        \\s+             # some more
whitespace
        (/nfs/data.*)   # 2: partition
name '''

pattern = ~regex
matcher = pattern.matcher(dfOutput)

```

For comparison, here's the Perl equivalent:

```

# Perl regex
$dfOutput =~
    m!                # begin match
        (\d+)         # 1: Disk space we're
after
        \s+           # one+ whitespace
characters
        \d+%         # digits followed by
' % '
        \s+           # one+ whitespace
characters
        (/nfs/data.*) # 2: partition name
    !ix;              # end match with
case-(i)nsensitive
                        # and e(x)tended format
options

```

In Perl you can use "m<delim>" to start a match instead of "/", so I could pick "!" as the delimiter and avoid having to escape the "/" characters in the partition name. Also Perl regular expressions use a single backslash, as in Groovy single line slashy regex patterns. Unfortunately, **for multi-line Groovy patterns defined as a here document string, you need to double the backslashes.**

#### *Other Notes*

Test code fragments in the groovyConsole

Groovy takes a couple of seconds to compile and run, and during iterative testing of a regular expression, this time becomes noticeable. Paste your code fragment into groovyConsole, set up some test input and a print statement to show the result. You can then run rapidly there without the compile lag. The print statement can become your debugging line when copying code back to your main project.

#### Conquer Complex Patterns by Dividing them into Simpler Categories

Beware of trying to deal with complicated patterns with a "one-size-fits-all" monster regular expression. Often it's easier to use an initial pattern to decide the category of input, and then select (if-then-else or switch-case) an appropriate simpler regex that deals with just that sub-category of input. You may need to include messages to warn against unexpected input that doesn't match any of the known categories.

## GDK Extensions to Object

Groovy adds a number of methods to `java.lang.Object`, most of which deal with types that serve as collections or aggregates, such as Lists or DOM Nodes.

Return Value	Method	Description
Boolean	<code>any {closure}</code>	returns <code>true</code> if the closure returns true for any item
List	<code>collect {closure}</code>	returns a list of all items that were returned from the closure
Collection	<code>collect(Collection collection) {closure}</code>	same as above, but adds each item to the given collection
void	<code>each {closure}</code>	simply executes the closure for each item
void	<code>eachWithIndex {closure}</code>	same as <code>each{}</code> except it passes two arguments: the item and the index
Boolean	<code>every {closure}</code>	returns <code>true</code> if the closure returns true for <i>all</i> items
Object	<code>find {closure}</code>	returns the first item that matches the closure expression
List	<code>findAll {closure}</code>	returns all items that match the closure expression
Integer	<code>indexOf {closure}</code>	returns the index of the first item that matched the given expression

See the [GDK documentation on Object](#) for the complete list of added methods.

Since the "return" keyword is optional in Groovy, closures in this context act as "predicates" and return the boolean result of whatever expression you given in your closure. These predicates allow you to apply perform operations on aggregate objects in a very concise manner.

## Examples

```
def numbers = [ 5, 7, 9, 12 ]
assert numbers.any { it % 2 == 0 }
//returns true since 12 is even

assert numbers.every { it > 4 }
//returns true since all #s are > 4

assert numbers.findAll { it in 6..10 } ==
[7,9]           //returns all #s > 5 and <
11

assert numbers.collect { ++it } == [6, 8,
10, 13]        //returns a new list with
each # incremented

numbers.eachWithIndex{ num, idx -> println
"$idx: $num" } //prints each index and
number
```

## Grape

### Quick Start

Grape lets you quickly add maven repository dependencies to your classpath. Here are the most common solutions:

### Add a Dependency

```
@Grab(group='org.springframework',
module='spring', version='2.5.6')
import
org.springframework.jdbc.core.JdbcTemplate
```

Yes, you can annotate an import in Groovy. You can also search for dependencies on [mvnrepository.com](http://mvnrepository.com) and it will provide you the @Grab annotation form of the pom.xml entry.

### Specify Additional Repositories

Not all dependencies are in maven central. You can add new ones like this:

```
@GrabResolver(name='restlet',  
root='http://maven.restlet.org/')  
@Grab(group='org.restlet',  
module='org.restlet', version='1.1.6')
```

### Maven Classifiers

Some maven dependencies need classifiers in order to be able to resolve. You can fix that like this:

```
@Grab(group='net.sf.json-lib',  
module='json-lib', version='2.2.3',  
classifier='jdk15')
```

### Excluding Transitive Dependencies

Sometimes you will want to exclude transitive dependencies as you might be already using a slightly different but compatible version of some artifact. You can do this as follows:

```
@Grab('net.sourceforge.htmlunit:htmlunit:2.8  
)  
@GrabExclude('xml-apis:xml-apis')
```

### JDBC Drivers

Because of the way JDBC drivers are loaded, you'll need to configure Grape to attach JDBC driver dependencies to the system class loader. I.e:

```
@GrabConfig(systemClassLoader=true)
@Grab(group='mysql',
module='mysql-connector-java',
version='5.1.6')
```

### Using Grape From the Groovy Shell

From groovysh use the method call variant:

```
groovy.grape.Grape.grab([group:'org.springframework',
module:'spring', version:'2.5.6'])
```

### Proxy settings

If you are behind a firewall and/or need to use Groovy/Grape through a proxy server, you can specify those settings on the command line via the `http.proxyHost` and `http.proxyPort` system properties:

```
groovy -Dhttp.proxyHost=yourproxy
-Dhttp.proxyPort=8080 yourscrip.groovy
```

Or you can make this system wide by adding these properties to your `JAVA_OPTS` environment variable:

```
JAVA_OPTS = -Dhttp.proxyHost=yourproxy
-Dhttp.proxyPort=8080
```

### Logging

If you want to see what Grape is doing set the system property `"groovy.grape.report.downloads"` to `"true"` (e.g. add `"-Dgroovy.grape.report.downloads=true"` to `JAVA_OPTS`) and Grape will print the following infos to `System.error`:

- Starting resolve of a dependency
- Starting download of an artifact
- Retrying download of an artifact
- Download size and time for downloaded artifacts

## Detail

Grape (The Groovy Adaptable Packaging Engine or Groovy Advanced Packaging Engine) is the infrastructure enabling the `grab()` calls in Groovy, a set of classes leveraging Ivy to allow for a repository driven module system for Groovy. This allows a developer to write a script with an essentially arbitrary library requirement, and ship just the script. Grape will, at runtime, download as needed and link the named libraries and all dependencies forming a transitive closure when the script is run from existing repositories such as Ibiblio, Codehaus, and java.net.

Grape follows the Ivy conventions for module version identification, with naming change.

- `group` - Which module group the module comes from. Translates directly to a Maven groupId or an Ivy Organization. Any group matching `/groovy[x][\.\.*]^/` is reserved and may have special meaning to the groovy endorsed modules.
- `module` - The name of the module to load. Translated directly to a Maven artifactId or an Ivy artifact.
- `version` - The version of the module to use. Either a literal version '1.1-RC3' or an Ivy Range '[2.2.1,)' meaning 2.2.1 or any greater version).

The downloaded modules will be stored according to Ivy's standard mechanism with a cache root of `~/ .groovy/g rape`

## Usage

### Annotation

One or more `groovy.lang.Grab` annotations can be added at any place that annotations are accepted to tell the compiler that this code relies on the specific library. This will have the effect of adding the library to the classloader of the groovy compiler. This annotation is detected and evaluated before any other resolution of classes in the script, so imported classes can be properly resolved by a `@Grab` annotation.

```
import com.jidesoft.swing.JideSplitButton
@Grab(group='com.jidesoft',
module='jide-oss', version='[2.2.1,2.3.0)')
public class TestClassAnnotation {
    public static String testMethod () {
        return JideSplitButton.class.name
    }
}
```

An appropriate `grab(...)` call will be added to the static initializer of the class of the containing class (or script class in the case of an annotated script element).

### Multiple Grape Annotations

In order to use a Grape annotation multiple times you must use the `Grapes` annotation, e.g.:

```
@Grapes ([
    @Grab(group='commons-primitives',
module='commons-primitives', version='1.0'),
    @Grab(group='org.ccil.cowan.tagsoup',
module='tagsoup', version='0.9.7')])
class Example {
// ...
}
```

Otherwise you'll encounter the following error:

```
Cannot specify duplicate annotation on the
same member
```

#### Method call

Typically a call to grab will occur early in the script or in class initialization. This is to insure that the libraries are made available to the ClassLoader before the groovy code relies on the code. A couple of typical calls may appear as follows:

```

import groovy.grape.Grape
// random maven library
Grape.grab(group: 'com.jidesoft',
module: 'jide-oss', version: '[2.2.0,)'')
Grape.grab([group: 'org.apache.ivy',
module: 'ivy', version: '2.0.0-beta1',
conf: ['default', 'optional']],
    [group: 'org.apache.ant', module: 'ant',
version: '1.7.0'])

// endorsed Groovy Module
// FUTURE grab('Scriptom')

```

\* Multiple calls to grab in the same context with the same parameters should be idempotent. However, if the same code is called with a different ClassLoader context then resolution may be re-run.

- ~~grab is disabled by default. Starting calling Grape.initGrape() will enable grab. Any calls to grab before initGrape() is called will be ignored. Hence Grape managed classloading is opt in only. Multiple calls to Grape.initGrape() after the first successful call are ignored.~~
- If the args map passed into the grab call has an attribute noExceptions that evaluates true no exceptions will be thrown.
- grab requires that a RootLoader or GroovyClassLoader be specified or be in the ClassLoader chain of the calling class. By default failure to have such a ClassLoader available will result in module resolution and an exception being thrown (~~if initGrape() has been called~~).
  - The ClassLoader passed in via the classLoader: argument and it's parent classloaders.
  - The ClassLoader of the object passed in as the referenceObject: argument, and it's parent classloaders.
  - The ClassLoader of the class issuing the call to grab

#### **grab(HashMap) Parameters**

- group: - <String> - Which module group the module comes from. Translates directly to a Maven groupId. Any group matching /groovy(|\..|\*|x\..)/ is reserved and may have special meaning to the groovy endorsed modules.
- module: - <String> - The name of the module to load. Translated directly to a Maven artifactId.
- version: - <String> and possibly <Range> - The version of the module to use. Either a literal version '1.1-RC3' or an Ivy Range '[2.2.1,)' meaning 2.2.1 or any greater version).
- classifier: - <String> - The Maven classifier to resolve by.
- conf: - <String>, default 'default' - The configuration or scope of the module to download. The default conf is default: which maps to the maven runtime and master scopes.
- force: - <boolean>, defaults true - Used to indicate that this revision must be used in case of conflicts,

- independently of
- conflicts manager
- `changing`: - <boolean>, default false - Whether the artifact can change without it's version designation changing.
- `transitive`: - <boolean>, default true - Whether to resolve other dependencies this module has or not.

There are two principal variants of `grab`, one with a single Map and one with an arguments Map and multiple dependencies map. A call to the single map `grab` is the same as calling `grab` with the same map passed in twice, so `grab` arguments and dependencies can be mixed in the same map, and `grab` can be called as a single method with named parameters.

There are synonyms for these parameters. Submitting more than one is a runtime exception.

- `group:`, `groupId:`, `organisation:`, `organization:`, `org:`
- `module:`, `artifactId:`, `artifact:`
- `version:`, `revision:`, `rev:`
- `conf:`, `scope:`, `configuration:`

#### Arguments Map arguments

- `classLoader`: - <GroovyClassLaoder> or <RootClassLoader> - The ClassLoader to add resolved Jars to
- `refObject`: - <Object> - The closest parent ClassLoader for the object's class will be treated as though it were passed in as `classLoader`:
- `validate`: - <boolean>, default false - Should poms or ivy files be validated (true), or should we trust the cache (false).
- `noExceptions`: - <boolean>, default false - If ClassLoader resolution or repository querying fails, should we throw an exception (false) or fail silently (true).

#### Command Line Tool

Grape added a command line executable 'grape' that allows for the inspection and management of the local grape cache.

```
grape install <groupId> <artifactId>  
[<version>]
```

This installs the specified groovy module or maven artifact. If a version is specified that specific version will be installed, otherwise the most recent version will be used (as if '\*' we passed in).

```
grape list
```

Lists locally installed modules (with their full maven name in the case of groovy modules) and versions.

```
grape resolve (<groupId> <artifactId>  
<version>)+
```

This returns the file locations of the jars representing the artifacts for the specified module(s) and the respective transitive dependencies. You may optionally pass in `-ant`, `-dos`, or `-shell` to get the dependencies expressed in a format applicable for an ant script, windows batch file, or unix shell script respectively. `-ivy` may be passed to see the dependencies expressed in an ivy like format.

## Advanced configuration

### Repository Directory

If you need to change the directory grape uses for downloading libraries you can specify the `grape.root` system property to change the default (which is `~/groovy/grape`)

```
groovy -Dgrape.root=/repo/grape  
yourscript.groovy
```

### Customize Ivy settings

//TODO expand on discussion of `grapeConfig.xml`

You can customize the ivy settings that Grape uses by creating a `~/groovy/grapeConfig.xml` file. If no such file exists, [here](#) are the default settings used by Grape:

```
<ivysettings>
  <settings
defaultResolver="downloadGrapes" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/ivy-[revision].xml" />
          <artifact
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/[type]s/[artifact]-[revisio
n].[ext]" />
            </filesystem>
            <!-- todo add 'endorsed groovy
extensions' resolver here -->
            <ibiblio name="codehaus"
root="http://repository.codehaus.org/"
m2compatible="true" />
            <ibiblio name="ibiblio"
m2compatible="true" />
            <ibiblio name="java.net2"
root="http://download.java.net/maven/2/"
m2compatible="true" />
          </chain>
        </resolvers>
      </ivysettings>
```

For more information on how to customize these settings, please refer to the [Ivy documentation](#).

*Add your local Maven2 repository*

If you find yourself wanting to reuse artifacts that you already have locally in your Maven2 repository, then you can add this line to your ~/.groovy/grapeConfig.xml:

```
<ibiblio name="local"  
root="file:${user.home}/.m2/repository/"  
m2compatible="true" />
```

And further customize your Grape configuration:

```
<?xml version="1.0"?>
<ivysettings>
  <settings
defaultResolver="downloadGrapes" />
  <resolvers>
    <chain name="downloadGrapes">
      <!-- todo add 'endorsed groovy
extensions' resolver here -->
      <ibiblio name="local"
root="file:${user.home}/.m2/repository/"
m2compatible="true" />
      <filesystem name="cachedGrapes">
        <ivy
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/ivy-[revision].xml" />
        <artifact
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/[type]s/[artifact]-[revisio
n].[ext]" />
      </filesystem>
      <ibiblio name="codehaus"
root="http://repository.codehaus.org/"
m2compatible="true" />
      <ibiblio name="ibiblio"
m2compatible="true" />
      <ibiblio name="java.net2"
root="http://download.java.net/maven/2/"
m2compatible="true" />
    </chain>
  </resolvers>
</ivysettings>
```

## More Examples

Using Apache Commons Collections:

```
// create and use a primitive array
import
org.apache.commons.collections.primitives.ArrayIntList

@Grab(group='commons-primitives',
module='commons-primitives', version='1.0')
def createEmptyInts() { new ArrayIntList() }

def ints = createEmptyInts()
ints.add(0, 42)
assert ints.size() == 1
assert ints.get(0) == 42
```

Using TagSoup:

```
// find the PDF links in the Java 1.5.0
documentation
@Grab(group='org.ccil.cowan.tagsoup',
module='tagsoup', version='0.9.7')
def getHtml() {
    def parser = new XmlParser(new
org.ccil.cowan.tagsoup.Parser())

parser.parse("http://java.sun.com/j2se/1.5.0
/download-pdf.html")
}
html.body.**.a.@href.grep(~/.*\pdf/).each
{ println it }
```

Using Google Collections:

```
// Google Collections example

import com.google.common.collect.HashBiMap
@Grab(group='com.google.code.google-collecti
ons', module='google-collect',
version='snapshot-20080530')
def getFruit() { [grape:'purple',
lemon:'yellow', orange:'orange'] as
HashBiMap }
assert fruit.lemon == 'yellow'
assert fruit.inverse().yellow == 'lemon'
```

Launching a Jetty server to serve Groovy templates:



```
@Grapes([

@Grab(group='org.eclipse.jetty.aggregate',
module='jetty-server',
version='8.1.7.v20120910'),

@Grab(group='org.eclipse.jetty.aggregate',
module='jetty-servlet',
version='8.1.7.v20120910'),
    @Grab(group='javax.servlet',
module='javax.servlet-api',
version='3.0.1')])

import org.eclipse.jetty.server.Server
import org.eclipse.jetty.servlet.*
import groovy.servlet.*

def runServer(duration) {
    def server = new Server(8080)
    def context = new
ServletContextHandler(server, "/",
ServletContextHandler.SSESSIONS);
    context.resourceBase = "."
    context.addServlet(TemplateServlet,
"* .gsp")
    server.start()
    sleep duration
    server.stop()
}

runServer(10000)
```

Grape will download Jetty and its dependencies on first launch of this script, and cache them. We're creating a new Jetty Server on port 8080, then expose Groovy's TemplateServlet at the root of the context — Groovy comes with its own powerful template engine mechanism. We start the server and let it run for a certain duration. Each time someone will hit <http://localhost:8080/somepage.gsp>, it will display the somepage.gsp template to the user — those template pages should be situated in the same directory as this server script.

**See Also:**

[Using Hibernate with Groovy](#)

<http://stackoverflow.com/questions/192432/getting-groovys-grape-going>

## Advanced OO

[Implement Interfaces using a Closure or a Map](#)

### Groovy way to implement interfaces

Groovy provides some very convenient ways to implement interfaces.

*Implement interfaces with a closure*

An interface with a single method can be implemented with a closure like so:

```
// a readable puts chars into a CharBuffer  
and returns the count of chars added  
def readable = { it.put("12 34".reverse());  
5 } as Readable  
  
// the Scanner constructor can take a  
Readable  
def s = new Scanner(readable)  
assert s.nextInt() == 43
```

You can also use a closure to implement an interface with more than one method. The closure will be invoked for each method on the interface. Since you need a closure whose parameter list matches that of all of the methods you typically will want to use an array as the sole parameter. This can be used just as it is for any Groovy closure and will collect all of the arguments in an array. For example:

```
interface X {
    void f()
    void g(int n)
    void h(String s, int n)
}
```

```
x = {Object[] args -> println "method called
with $args"} as X
x.f()
x.g(1)
x.h("hello",2)
```

#### *Implement interfaces with a map*

More commonly an interface with multiple methods would be implemented with a map like so:

```
impl = [
    i: 10,
    hasNext: { impl.i > 0 },
    next: { impl.i-- },
]
iter = impl as Iterator
while ( iter.hasNext() )
    println iter.next()
```

Note this is a rather contrived example, but illustrates the concept.

You only need to implement those methods that are actually called, but if a method is called that doesn't exist in the map a *NullPointerException* is thrown. For example:

```
interface X
{ void f(); void g(int n); void h(String s,
int n); }

x = [ f: {println "f called"} ] as X
x.f()
//x.g()      // NPE here
```

Be careful that you don't accidentally define the map with {}. Can you guess what happens with the following?

```
x = { f: {println "f called"} } as X
x.f()
x.g(1)
```

What we've defined here is a closure with a label and a block. Since we've just defined a single closure every method call will invoke the closure. Some languages use {} to define maps so this is an easy mistake until you get used to using [] to define maps in Groovy.

Note that using the "as" operator as above requires that you have a static reference to the interface you want to implement with a map. If you have a reference to the java.lang.Class object representing the interface (i.e. do not know or can not hard code the type at script write time) you want to implement, you can use the asType method like this:

```
def loggerInterface = Class.forName(
    'my.LoggerInterface' )
def logger = [
    log : { Object[] params ->
println "LOG: ${params[0]}"; if(
params.length > 1 )
params[1].printStackTrace() },
    close : { println
"logger.close called" }
].asType( loggerInterface )
```

See also:

- [Developer Testing using Closures instead of Mocks](#)

## Annotations with Groovy

### *Introduction*

Java 5 and above supports the use of annotations to include metadata within programs. Groovy 1.1 and above also supports such annotations.

Annotations are used to provide information to tools and libraries. They allow a declarative style of providing metadata information and allow it to be stored directly in the source code. Such information would need to otherwise be provided using non-declarative means or using external files. We won't discuss guidelines here for when it is appropriate to use annotations, just give you a quick run down of annotations in Groovy.

Annotations are defined much like Java class files but use the `@interface` keyword. As an example, here is how you could define a `FeatureRequest` Annotation in Java:

```

// Java
public @interface FeatureRequest {
    String key();
    String summary();
    String assignee() default
"[unassigned]";
    String status() default "[open]";
    String targetVersion() default
"[unassigned]";
}

```

This annotation represents the kind of information you may have in an issue tracking tool. You could use this annotation in a Groovy file as follows:

```

@FeatureRequest (
    key="GROOVY-9999",
    summary="Support Graphical Annotations",
    assignee="Pete",
    status="Open",
    targetVersion="5.0"
)
class SomeClassWhereFeatureCouldBeUsed {
    // ...
}

```

Now if you had tools or libraries which understood this annotation, you could process this source file (or the resulting compiled class file) and perform operations based on this metadata.

As well as defining your own annotations, there are many existing tools, libraries and frameworks that make use of annotations. See some of the examples referred to at the end of this page. As just one example, here is how you could use annotations with Hibernate or JPA:

```
import javax.persistence.*

@Entity
@Table(name="staff")
class Staff implements Serializable {
    @Id @GeneratedValue
    Long id
    String firstname
    String lastname
    String position
}
```

*Example*

As another example, consider this [XStream](#) example. XStream is a library for serializing Java (and Groovy) objects to XML (and back again if you want). Here is an example of how you could use it without annotations:

```
//
require(groupId: 'com.thoughtworks.xstream',
artifactId: 'xstream', version: '1.3')
import com.thoughtworks.xstream.*

class Staff {
    String firstname, lastname, position
}

def xstream = new XStream()
xstream.classLoader = getClass().classLoader
def john = new Staff(firstname: 'John',
                    lastname: 'Connor',
                    position: 'Resistance
Leader')

println xstream.toXML(john)
```

This results in the following output:

```
<Staff>
  <firstname>John</firstname>
  <lastname>Connor</lastname>
  <position>Resistance Leader</position>
</Staff>
```

Just as an aside, not related to annotations, here is how you could write the XML to a file:

```
new File("john.xml").withOutputStream { out
->
    xstream.toXML(john, out)
}
```

And how you would read it back in:

```
//
require(groupId: 'com.thoughtworks.xstream',
artifactId: 'xstream', version: '1.3')
// require(groupId: 'xpp3',
artifactId: 'xpp3_min', version: '1.1.4c')
import com.thoughtworks.xstream.*

class Staff {
    String firstname, lastname, position
}

def xstream = new XStream()
def john
// now read back in
new File("john.xml").withInputStream { ins
->
    john = xstream.fromXML(ins)
}

println john.dump()
// => <Staff@12d96f2 firstname=John
lastname=Connor position=Resistance Leader>
```

Now, on to the annotations ...

XStream also allows you to have more control over the produced XML (in case you don't like its defaults). This can be done through API calls or with annotations. Here is how we can annotate our Groovy class with XStream annotations to alter the resulting XML:

```
// ...
import
com.thoughtworks.xstream.annotations.*

@XStreamAlias("person")
class Associate {
    @XStreamAsAttribute
    @XStreamAlias('first-name')
    String firstname

    @XStreamAlias('surname')
    String lastname

    @XStreamOmitField
    String position
}

msg = new Associate(firstname:'Sarah',
                    lastname:'Connor',
                    position:'Protector')
Annotations.configureAliases(stream,
Associate)
println stream.toXML(msg)
```

When run, this produces the following output:

```
<person first-name="Sarah">
  <surname>Connor</surname>
</person>
```

#### *Differences to Java*

Annotations may contain lists. When using such annotations with Groovy, remember to use the square bracket list notation supported by Groovy rather than the braces used by Java, i.e.:

```
// Java
@ManagedOperationParameters({
    @ManagedOperationParameter(name="x",
description="The first number"),
    @ManagedOperationParameter(name="y",
description="The second number")})
```

Would become:

```
// Groovy
@ManagedOperationParameters([
    @ManagedOperationParameter(name="x",
description="The first number"),
    @ManagedOperationParameter(name="y",
description="The second number")])
```

#### *More Examples*

Annotations are also used in examples contained within the following pages:

- [Using JUnit 4 with Groovy](#)
- [Using TestNG with Groovy](#)
- [Using Instinct with Groovy](#)
- [Using Popper with Groovy](#)
- [Singleton Pattern](#)
- [Using Spring Factories with Groovy](#)

- [Groovy and JMX](#)

## Generics

Java 1.5 introduced Generics. Using generics you can write code that can be statically checked to a greater degree at compile time. In some ways this is at odds with the emphasis of dynamic languages where in general, the type of objects can not be determined until runtime. But Groovy aims to accommodate Java's static typing when possible, hence Groovy 1.5 now also understands Generics. Having said that, Groovy's generics support doesn't aim to be a complete clone of Java's generics. Instead, Groovy aims to allow generics at the source code level (to aid cut and pasting from Java) and also where it makes sense to allow good integration between Groovy and Java tools and APIs that use generics.

You can include generics in your definitions like this:

```
import java.lang.reflect.Method  
Iterable<Method> methods =  
String.methods.grep{  
it.name.startsWith('get') }  
assert methods.name == [ "getBytes",  
"getBytes", "getBytes", "getBytes",  
"getChars", "getClass" ]
```

*Implementation note:* Java's generics implementation incorporates a feature known as "type erasure" which "throws away" generic type information after completing static type checking. This allows Java to easily integrate with legacy "non-generics" libraries. Groovy currently does a little further and throws away generics information "at the source level". Generics information is kept within signatures where appropriate (see for example the method *foo* below within class *D*).

You can define classes using generics like this:

```
class A extends ArrayList<Long> {}

class B<T> extends HashMap<T,List<T>> {}

class C<Y,T extends
Map<String,Map<Y,Integer>>> {}

class D {
    static < T > T foo(T t) {return null}
}
```

## Using Enums

Some examples (inspired by the Java Enum Tutorial):

```
enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}

def tellItLikeItIs(Day day) {
    switch (day) {
        case Day.MONDAY:
            println "Mondays are bad."
            break
        case Day.FRIDAY:
            println "Fridays are better."
            break
        case Day.SATURDAY:
        case Day.SUNDAY:
```

```
        println "Weekends are best."
        break
    default:
        println "Midweek days are
so-so."
        break
    }
}
```

```
tellItLikeItIs(Day.MONDAY)        // =>
Mondays are bad.
tellItLikeItIs(Day.WEDNESDAY)    // =>
Midweek days are so-so.
tellItLikeItIs(Day.FRIDAY)       // =>
```

```
Fridays are better.  
tellItLikeItIs(Day.SATURDAY) // =>  
Weekends are best.
```

Or with a bit of refactoring, you could write the switch like this:

```
def today = Day.SATURDAY  
switch (today) {  
  // Saturday or Sunday  
  case [ Day.SATURDAY, Day.SUNDAY ]:  
    println "Weekends are cool"  
    break  
  // a day between Monday and Friday  
  case Day.MONDAY..Day.FRIDAY:  
    println "Boring work day"  
    break  
  default:  
    println "Are you sure this is a  
valid day?"  
}
```

Here is a coin example:

```

enum Coin {
    penny(1), nickel(5), dime(10),
quarter(25)
    Coin(int value) { this.value = value }
    private final int value
    public int value() { return value }
}

assert Coin.values().size() == 4

def pocketMoney = 2 * Coin.quarter.value() +
5 * Coin.dime.value()
assert pocketMoney == 100

// another way to do above
def coins = [ Coin.quarter ] * 2 + [
Coin.dime ] * 5
println coins // => [ quarter, quarter,
dime, dime, dime, dime, dime ]
println coins.sum{ it.value() } // => 100

```

Here is a planet example:

```

enum Planet {
    MERCURY(3.303e+23, 2.4397e6),
    VENUS(4.869e+24, 6.0518e6),
    EARTH(5.976e+24, 6.37814e6),
    MARS(6.421e+23, 3.3972e6),
    JUPITER(1.9e+27, 7.1492e7),
    SATURN(5.688e+26, 6.0268e7),

```

```

    URANUS(8.686e+25, 2.5559e7),
    NEPTUNE(1.024e+26, 2.4746e7)
    private final double mass // in
kilograms
    private final double radius // in metres
    Planet(double mass, double radius) {
        this.mass = mass
        this.radius = radius
    }
    private double mass() { return mass }
    private double radius() { return radius
}
    // universal gravitational constant (m3
kg-1 s-2)
    public static final double G =
6.67300E-11
    double surfaceGravity() { return G *
mass / (radius * radius) }
    double surfaceWeight(double otherMass) {
return otherMass * surfaceGravity() }
}

double earthWeight = 75.0 // kg
double mass =
earthWeight/Planet.EARTH.surfaceGravity()
for (p in Planet.values()) {
    printf("Your weight on %s is %f\n", p,
p.surfaceWeight(mass))
}

```

```
// =>  
// Your weight on MERCURY is 28.331821  
// Your weight on VENUS is 67.874932  
// Your weight on EARTH is 75.000000  
// Your weight on MARS is 28.405289  
// Your weight on JUPITER is 189.791814
```

```
// Your weight on SATURN is 79.951165
// Your weight on URANUS is 67.884540
// Your weight on NEPTUNE is 85.374605
```

## Static Import Usage

Groovy's *static import* capability allows you to reference imported classes as if they were static methods in your own class. This is similar to Java's *static import* capability but works with Java 1.4 and above and is a little more dynamic than Java in that it allows you to define methods with the same name as an imported method as long as you have different types. If you have the same types, the imported class takes precedence. Here is a sample of its usage:

```
import static java.awt.Color.LIGHT_GRAY
import static Boolean.FALSE as F
import static Calendar.getInstance as now
import static java.lang.Integer.*

println LIGHT_GRAY
// => java.awt.Color[r=192,g=192,b=192]

println !F
// => true

println now().time
// => Sun Apr 29 11:12:43 EST 2007

println "Integers are between $MIN_VALUE and
$MAX_VALUE"
// => Integers are between -2147483648 and
2147483647

def toHexString(int val, boolean upperCase)
{
    def hexval = upperCase ?
toHexString(val).toUpperCase() :
toHexString(val)
    return '0x' + hexval
}
println toHexString(15, true)
// => 0xF
println toHexString(15, false)
// => 0xf
```

The first static import illustrates defining `LIGHT_GRAY` as if it was defined locally as a static field. The next two examples show renaming (called *aliasing*) of a field and a method respectively. The final example illustrates wild-carding for fields and methods and also selecting between the locally defined `toHexString` and imported `toHexString` based on parameter matching.

As another example, here is how to statically import some of the Math functions:

```
import static java.lang.Math.*
println sin(123.456) * cos(456.789)
// => 0.24733809349262376
```

**Note:** Groovy does not check beyond your import class if what you statically import exists. If you import a nonexisting method, field or property Groovy will not fail at compile time, but later when executing the compiled code.

## Scoping and the Semantics of "def"

**Note:** This page may not follow the explanations from the JLS.

Java has two basic informal rules for scoping:

**Principle #1:** "A variable is only visible in the block it is defined in and in nested blocks".

**Principle #2:** "A variable can't be visible more than one time".

Java does know classwide variables and local variables. Local variables are defined as method parameter or inside the method block. Classwide variables are defined as attributes of the class. Of course Java does violate this first principle I showed at top here a little since I can access class wide variables from outside the class, if the access modifier is for example public. A local variable of the same name as an attribute does not violate the second principle, as the attribute is hidden by the local variable and with this no longer visible without using a qualifier like "this".

### Now, what about Groovy?

In Groovy we also have these two principles, but since we have different constructs, we may lay out these principles in a different way. Let us start with local variables.

- In Groovy you are neither allowed to define two local variables of the same name, just like in Java.
- You are allowed to hide an attribute by defining a local variable of the same name, just like in Java.

### So what is different?

Scripts are. When you define a variable in a script it is always local. But methods are not part of that scope. So defining a method using different variables as if they were attributes and then defining these variables normally in the script leads to problems. Example:

## Example of disallowed definition in scripts

```
String attribute = "bar"
void aMethod(){
    assert attribute == "bar" // Not allowed !
}
aMethod()
```

Executing this code you get an exception talking about a missing property or field.

The only things the method has access to are:

- the binding,
  - attributes defined by the base class, and
  - the dynamic properties defined by the MetaClass (explanations for these will follow).
- "attribute" here is no field, no property, no dynamic defined property and not part of the binding.

### When is something in the Binding and when not?

That's easy. When it is not defined, it is in the binding.

```
String localVar = "I am a local variable"
bindingVar = "I am a binding variable"
```

The trick is - and that is admittedly not easy for Java programmers - to **not** to define the variable before using it, and it will go into the binding. Any defined variable is local. **Please note: the binding exists only for scripts.**

### What is this "def" I heard of?

"def" is a replacement for a type name. In variable definitions it is used to indicate that you don't care about the type. In variable definitions it is mandatory to either provide a type name explicitly or to use "def" in replacement. This is needed to make variable definitions detectable for the Groovy parser.

These definitions may occur for local variables in a script or for local variables and properties/fields in a class.

#### ✓ Rule of thumb

You can think of "def" as an alias of "Object" and you will understand it in an instant.

"def" can also replace "void" as the return type in a method definition.

```
def dynamic = 1
dynamic = "I am a String stored in a
variable of dynamic type"
int typed = 2
typed = "I am a String stored in a variable
of type int??" // throws
ClassCastException
```

The assignment of a string, to a variable of type int will fail. A variable typed with "def" allows this.

### A Closure is a block

In the terms of the principles above a closure is a block. A variable defined in a block is visible in that block and all blocks that are defined in that block. For example in Java:

#### a Java block

```
{
  int i=1;
  {
    System.out.println (i);
  }
}
```

Such a block may be defined freely as in the example above, or by loops, synchronized statements, try-catch, switch, ... all that has "{".

In Groovy we have an additional structure with "{", the closure. To follow the principles above, it is not allowed to define two variables of the same name in a closure.

#### Not allowed

```
def closure = { int i; int i }
```

And of course, the same for combinations with nested closures.

## invalid double definition of variables

```
def outer = {  
  int i  
  def inner = { int i }  
}
```

A block ends with its corresponding "}". So it is allowed to reuse that name later in a different block.

## Allowed

```
def closure1 = { parameter ->  
  println parameter  
}  
def closure2 = { parameter ->  
  println parameter  
}
```

Both closures define a local variable named "parameter", but since these closures are not nested, this is allowed. **Note: unlike early versions of Groovy and unlike PHP, a variable is visible in the block, not outside. Just like in Java.**

And "it"?

"it" is a special variable name, that is defined automatically inside a closure. It refers always to the first parameter of the closure, or null, if the closure doesn't have any parameters.

## implicit it in closures

```
def c = { it }  
assert c() == null  
assert c(1) == 1
```

When using nested closures (closures in closures) the meaning of "it" depends on the closure you are in.

```
def outer = {  
    def inner = { it+1 }  
    inner(it+1)  
}  
assert outer(1) == 3
```

You see, that "it" is used two times, the "it" in "inner" means the first parameter of the closure "inner", the following "it" means the first parameter of "outer". This helps a lot when copying code from one place to another containing closure that are using it.

### The keyword "static"

"static" is a modifier for attributes (and methods, but this is not at issue here). It defines the "static scope". That means all variables not defined with "static" are not part of that static scope and as such not visible there. There is no special magic to this in Groovy. So for an explanation of "static" use any Java book.

### Database features

Groovy supports a few neat ways to work with SQL more easily and to make SQL more Groovy. You can perform queries and SQL statements, passing in variables easily with proper handling of statements, connections and exception handling thanks to closures.

```
import groovy.sql.Sql

def foo = 'cheese'
def sql =
Sql.newInstance("jdbc:mysql://localhost:3306
/mydb", "user",
                "pswd",
                "com.mysql.jdbc.Driver")

sql.eachRow("select * from FOOD where
type=${foo}") {
    println "Gromit likes ${it.name}"
}
```

In the above example, you can refer to the various columns by name, using the property syntax on the row variable (e.g. `it.name`) or you can refer to the columns by their index (e.g. `it[0]`) For example:

```
import groovy.sql.Sql

def foo = 'cheese'
def sql =
Sql.newInstance("jdbc:mysql://localhost:3306
/mydb", "user",
                "pswd",
                "com.mysql.jdbc.Driver")

def answer = 0
sql.eachRow("select count(*) from FOOD where
type=${foo}") { row ->
    answer = row[0]
}
assert answer > 0
```

Or you can create a **DataSet** which allows you to query SQL using familiar closure syntax so that the same query could work easily on in memory objects or via SQL. e.g.

```

import groovy.sql.Sql

def sql =
Sql.newInstance("jdbc:mysql://localhost:3306
/mydb", "user",
                "pswd",
                "com.mysql.jdbc.Driver")

def food = sql.dataSet('FOOD')
def cheeses = food.findAll { it.type ==
'cheese' }
cheeses.each { println "Eat ${it.name}" }

```

#### Advanced Usage

In this example, we create a table, make changes to it and confirm the changes worked.

```

def sql =
Sql.newInstance("jdbc:mysql://localhost:3306
/mydb",
                "user", "pswd", "com.mysql.jdbc.Driver")

// delete table if previously created
try {
    sql.execute("drop table PERSON")
} catch(Exception e){}

// create table
sql.execute('''create table PERSON (
    id integer not null primary key,

```

```
        firstname varchar(20),
        lastname varchar(20),
        location_id integer,
        location_name varchar(30)
    )''')

// now let's populate the table
def people = sql.dataSet("PERSON")
people.add( firstname:"James",
            lastname:"Strachan", id:1, location_id:10,
            location_name:'London' )
people.add( firstname:"Bob",
            lastname:"Mcwhirter", id:2, location_id:20,
            location_name:'Atlanta' )
people.add( firstname:"Sam",
            lastname:"Pullara", id:3, location_id:30,
            location_name:'California' )

// do a query to check it all worked ok
def results = sql.firstRow("select
firstname, lastname from PERSON where
id=1").firstname
def expected = "James"
assert results == expected

// allow resultSets to be able to be changed
sql.resultSetConcurrency =
java.sql.ResultSet.CONCUR_UPDATABLE

// change the data
sql.eachRow("select * from PERSON") {
```

```
        it.firstname = it.firstname * 2
    }

    // reset resultSetsConcurrency back to read
    only (no further changes required)
    sql.resultSetConcurrency =
    java.sql.ResultSet.CONCUR_READ_ONLY

    // do a query to confirm that our change
    actually worked
    results = sql.firstRow("select firstname,
```

```
lastname from PERSON where id=1").firstname
expected = "JamesJames"
assert results == expected
```

#### *Combining with MarkupBuilder Example*

Here's an example of using Groovy SQL along with [GroovyMarkup](#)

```

import groovy.sql.Sql
import groovy.xml.MarkupBuilder

def sql =
Sql.newInstance("jdbc:mysql://localhost:3306
/mydb", "user",
                "pswd",
                "com.mysql.jdbc.Driver")

// lets output some XML builder
// could be SAX / StAX / DOM / TrAX / text
etc
def xml = new MarkupBuilder()

def ignore = 'James'
sql.eachRow("select * from person where
firstname != ${ignore}") { person ->
    // lets process each row by emitting
some markup
    xml.customer(id:person.id,
type: 'Customer',
                name:"$person.firstname
$person.lastname" )
}

```

This could generate, dynamically something like

```
<customers>
  <customer id="123" type="Customer"
foo="whatever">
    <role>partner</role>
    <name>James</name>
    <location id="5" name="London"/>
  </customer>
</customers>
```

There's an example [test case which demonstrates](#) all of these query mechanisms in action.

#### *Stored procedure support*

An example of calling a `FullName` stored procedure which takes a param ('Sam' in the example) and returns a `VAR CHAR`.

```
sql.call("{? = call FullName(?)}",
[Sql.VARCHAR, 'Sam']) { name ->
  assert name == 'Sam Pullara'
}
```

The same example again but with a `GString` variation:

```
def first = 'Sam'
sql.call("${Sql.VARCHAR = call
FullName($first)}") { name ->
  assert name == 'Sam Pullara'
}
```

Here is an example of a stored procedure with an out parameter:

```

sql.call '{call Hemisphere(?, ?, ?)}',
['Guillaume', 'Laforge', Sql.VARCHAR], {
dwells ->
    println dwells // => Northern Hemisphere
}

```

Some additional advanced stored procedure examples:

```

import java.sql.Connection
import java.sql.DriverManager
import javax.sql.DataSource
import groovy.sql.Sql
import oracle.jdbc.driver.OracleTypes

driver = oracle.jdbc.driver.OracleDriver
Connection conn =
DriverManager.getConnection(

'jdbc:oracle:thin:sirtest/sirtest@duck.aplpi
.lan:1521:orcl');

/*
 *
 * Here we call a procedural block with a
closure.
 * ${Sql.INTEGER} and ${Sql.VARCHAR} are out
parameters
 * which are passed to the closure.
 *
 */

```

```

Sql sql = new Sql(conn);
def a="foo";
String foo = "x";
println "${a}=${a}"
undefinedVar = null
println ""
--Simple demonstration of call with closure.
--Closure is called once with all returned
values.
""
sql.call("begin ${Sql.INTEGER}:=20;
${Sql.VARCHAR}:='hello world';end;") {
    answer,string ->

    println "number=[${answer}]
string=[${string}]"
    println "answer is a ${answer.class}";
    println "string is a ${string.class}";
    answer += 1;
    println "now number=${answer}"
    println
    ""[${string.replaceAll('o','O')}]""
}

/*
 * Here we execute a procedural block. The
block returns four out
 * parameters, two of which are cursors. We
use Sql.resultSet function
 * to indicate that the cursors should be

```

```
returned as GroovyResultSet.
```

```
*
```

```
*
```

```
*
```

```
*/
```

```
println """--next we see multiple return  
values including two ResultSets
```

```
--(ResultSets become GroovyResultSets)
```

```
--Note the GroovyResultSet.eachRow()  
function!!
```

```
"""
```

```
def tableClosure = {println
```

```
"table:${it.table_name}"};
```

```
println("tableClosure is a
```

```
${tableClosure.class}");
```

```
String owner = 'SIRTEST';
```

```
sql.call("""declare
```

```
type crsr is ref cursor;
```

```
tables crsr;
```

```
objects crsr;
```

```
begin
```

```
select count(*) into ${Sql.INTEGER} from  
all_tables where owner= ${owner} ;
```

```
open tables for select * from all_tables  
where owner= ${owner} ;
```

```
${Sql.resultSet OracleTypes.CURSOR} :=
```

```
tables;
```

```
select count(*) into ${Sql.INTEGER} from  
all_objects where owner= ${owner} ;
```

```

open objects for select * from all_objects
where owner= ${owner};
${Sql.resultSet OracleTypes.CURSOR} :=
objects;
end;
"""
){t,user_tables,o,user_objects ->
  println "found ${t} tables from a total of
${o} objects"
      // eachRow is a new method on
GroovyResultSet
  user_tables.eachRow(){x ->println
"table:${x.table_name}" }
  user_objects.eachRow(){println
"object:${it.object_name}" }
}

/*
 * Determine if we have the stored procedure
'fred' needed
 * for the next test.
 *
 */
Integer procLines = 0
sql.eachRow("select count(*) lines from
user_source where name='FRED' and
type='FUNCTION' ") {
    procLines = it.lines
}

```

```

if(procLines ==0) {
    print ""
    --to demonstrate a function accepting an
    inout parameter
    --and returning a value, create the
    following function in your schema
    create or replace function fred(foo in out
    varchar2) return number is
    begin
    foo:='howdy doody';
    return 99;
    end;
    ""

}else{
    /*
    * Here is a call to a function, passing in
    inout parameter.
    * The function also returns a value.
    */
    println "Next call demonstrates a function
    accepting inout parameter and returning a
    value"
    sql.call("{ ${Sql.INTEGER} = call
    fred(${Sql.inout(Sql.VARCHAR(foo))}) }") {
        answer,string ->
        println "returned number=[${answer}]
    inout string coming back=[${string}]"
    }
}

```

```

println "--Same again, but this time passing
a null inout parameter"
sql.call("{ ${Sql.INTEGER} = call
fred(${Sql.inout(Sql.VARCHAR(undefinedVar))}
) }") {
    answer,string ->
    println "returned number=[${answer}]
inout string coming back=[${string}]"
    answer = answer + 1;
    println "Checked can increment returned
number, now number=${answer}"
    println
    ""[${string.replaceAll('o','O')}]""
}

}

/*
 * Finally a handy function to tell Sql to
expand a variable in the
 * GString rather than passing the value as
a parameter.
 *
 */
["user_tables","all_tables"].each(){table ->
    sql.eachRow("select count(*) nrows from
${Sql.expand table}") {
        println "${table} has ${it.nrows}

```

```
rows "  
    }  
}
```

#### *Clob Notes*

CLOB are objects that cannot be extracted with a piece of code like:

```
data = sql.rows("select clobdata from ....")
```

because later

```
data.each { ... do something ... }
```

can fail. This happens cause the object is not *reachable* any more since the connection might be already closed: note I say might cause at least on Oracle the errors are random (and hard to understand).

A possible way to act on CLOB data is to use `eachRow`

```
data = sql.eachRow("select clobdata from  
....") { ... do something ... }
```

#### *Further Information*

There is also an additional [GSQL](#) module you might want to check out.

## Dynamic Groovy

This section details how to go about using the dynamic features of Groovy such as implementing the `GroovyObject` interface and using `ExpandoMetaClass`, an expandable `MetaClass` that allows adding of methods, properties and constructors.

- [Using `invokeMethod` and `getProperty`](#)
- [Using `methodMissing` and `propertyMissing`](#)
- [Evaluating the `MetaClass` runtime](#)
- [Using `ExpandoMetaClass` to add behaviour](#)
- [Customizing `MetaClass` for a single instance](#)
- [Runtime mixins](#)

Compile-time metaprogramming is also available using [Compile-time Metaprogramming - AST Transformations](#)

## Dynamic Method Invocation

You can invoke a method even if you don't know the method name until it is invoked:

```
class Dog {
  def bark() { println "woof!" }
  def sit() { println "(sitting)" }
  def jump() { println "boing!" }
}

def doAction( animal, action ) {
  animal."$action"()
  //action name is passed at invocation
}

def rex = new Dog()

doAction( rex, "bark" )
//prints 'woof!'
doAction( rex, "jump" )
//prints 'boing!'
```

You can also "spread" the arguments in a method call, when you have a list of arguments:

```
def max(int i1, int i2) {
  Math.max(i1, i2)
}

def numbers = [1, 2]
assert max( *numbers ) == 2
```

This also works in combination of the invocation with a GString:

```
someObject. "$methodName" (*args)
```

## Evaluating the MetaClass runtime

### Evaluating the MetaClass runtime

Since 1.1, Groovy supports a much richer set of APIs for evaluating the MetaClass runtime. Using these APIs in combination with [ExpandoMetaClass](#) makes Groovy an extremely powerful language for meta-programming

#### *Finding out methods and properties*

To obtain a list of methods ( or [MetaMethod](#) instances in Groovy speak) for a particular Groovy class use can inspect its MetaClass:

```
println obj.metaClass.methods  
println obj.metaClass.methods.find {  
it.name.startsWith("to") }
```

The same can be done for properties:

```
println obj.metaClass.properties  
println obj.metaClass.properties.find {  
it.name.startsWith("to") }
```

#### *Using respondsTo and hasProperty*

Obtaining a list of methods sometimes is a little more than what you want. It is quite common in meta-programming scenarios to want to find out if an object supports a particular method.

Since 1.1, you can use `respondsTo` and `hasProperty` to achieve this:

```
class Foo {  
    String prop  
    def bar() { "bar" }  
    def bar(String name) { "bar $name" }  
    def add(Integer one, Integer two) { one +
```

```
two}
}

def f = new Foo()

if(f.metaClass.respondsTo(f, "bar")) {
    // do stuff
}

if(f.metaClass.respondsTo(f, "bar", String))
{
    // do stuff
}

if(!f.metaClass.respondsTo(f, "bar",
Integer)) {
    // do stuff
}

if(f.metaClass.respondsTo(f, "add", Integer,
Integer)) {
    // do stuff
}
```

```
if(f.metaClass.hasProperty(f, "prop")) {
    // do stuff
}
```

The respondsTo method actually returns a List of MetaMethod instances so you can use it to both query and evaluate the resulting list.

 respondsTo only works for "real" methods and those added via [ExpandoMetaClass](#) and not for cases where you override invokeMethod or methodMissing. It is impossible in these cases to tell if an object responds to a method without actually invoking the method.

## ExpandoMetaClass

### Using ExpandoMetaClass to add behaviour

Groovy 1.1 includes a special MetaClass called an ExpandoMetaClass that allows you to dynamically add methods, constructors, properties and static methods using a neat closure syntax.

How does it work? Well every java.lang.Class is supplied with a special "metaClass" property that when used will give you a reference to an ExpandoMetaClass instance.

For example given the Java class java.lang.String to obtain its ExpandoMetaClass you can use:

```
String.metaClass.swapCase = {->
    def sb = new StringBuffer()
    delegate.each {
        sb << (Character.isUpperCase(it
as char) ? Character.toLowerCase(it as char)
:
        Character.toUpperCase(it
as char))
    }
    sb.toString()
}
```

This adds a method called swapCase to the String class.



By default `ExpandoMetaClass` doesn't do inheritance. To enable this you must call `ExpandoMetaClass.enableGlobally()` before your app starts such as in the main method or servlet bootstrap

Further Reading:

- [ExpandoMetaClass - Borrowing Methods](#) — Borrowing methods from other classes
- [ExpandoMetaClass - Constructors](#) — Adding or overriding constructors
- [ExpandoMetaClass Domain-Specific Language](#)
- [ExpandoMetaClass - Dynamic Method Names](#) — Dynamically creating method names
- [ExpandoMetaClass - GroovyObject Methods](#) — Overriding `invokeMethod`, `getProperty` and `setProperty`
- [ExpandoMetaClass - Interfaces](#) — Adding methods on interfaces
- [ExpandoMetaClass - Methods](#) — Adding or overriding instance methods
- [ExpandoMetaClass - Overriding static invokeMethod](#) — Overriding `invokeMethod` for static methods
- [ExpandoMetaClass - Properties](#) — Adding or overriding properties
- [ExpandoMetaClass - Runtime Discovery](#) — Overriding `invokeMethod` for static methods
- [ExpandoMetaClass - Static Methods](#) — Adding or overriding static methods

## ExpandoMetaClass - Borrowing Methods

### *ExpandoMetaClass - Borrowing Methods from other classes*

With `ExpandoMetaClass` you can also use Groovy's method pointer syntax to borrow methods from other classes. For example:

```
class Person {
  String name
}
class MortgageLender {
  def borrowMoney() {
    "buy house"
  }
}

def lender = new MortgageLender()

Person.metaClass.buyHouse =
lender.&borrowMoney

def p = new Person()

assert "buy house" == p.buyHouse()
```

## ExpandoMetaClass - Constructors

### *ExpandoMetaClass - Adding constructors*

Adding constructors is a little different to adding a method with ExpandoMetaClass. Essentially you use a special "constructor" property and either use the << or = operator to assign a closure. The arguments to the closure are of course the constructor arguments

```
class Book {  
    String title  
}  
Book.metaClass.constructor << { String title  
-> new Book(title:title) }  
  
def b = new Book("The Stand")
```

Be careful when adding constructors however, as it is very easy to get into stack overflow troubles. For example this code which overrides the default constructor:

```
class Book {  
    String title  
}  
Book.metaClass.constructor = { new Book() }  
  
def b = new Book("The Stand")
```

The above would produce a `StackOverflowError` as it recursively keeps calling the same constructor through Groovy's MetaClass system. You can get around this by writing helper code to instantiate an instance outside of Groovy. For example this uses Spring's `BeanUtils` class and does not cause a `StackOverflow`:

```
class Book {  
    String title  
}  
Book.metaClass.constructor = {  
    BeanUtils.instantiateClass(Book) }  
  
def b = new Book("The Stand")
```

### **ExpandoMetaClass Domain-Specific Language**

Initially developed under the Grails umbrella and integrated back into Groovy 1.5, ExpandoMetaClass is a very handy way for changing the runtime behavior of your objects and classes, instead of writing full-blown MetaClass classes. Each time, we want to add / change several properties or methods of an existing type, there is too much of a repetition of `Type.metaClass.xxx`. Take for example this extract of a Unit manipulation DSL dealing with operator overloading:

```
Number.metaClass.multiply = { Amount amount
-> amount.times(delegate) }
Number.metaClass.div =      { Amount amount
-> amount.inverse().times(delegate) }

Amount.metaClass.div =      { Number factor
-> delegate.divide(factor) }
Amount.metaClass.div =      { Amount factor
-> delegate.divide(factor) }
Amount.metaClass.multiply = { Number factor
-> delegate.times(factor) }
Amount.metaClass.power =    { Number factor
-> delegate.pow(factor) }
Amount.metaClass.negative = { ->
delegate.opposite() }
```

The repetition, here, looks obvious. But with the `ExpandoMetaClass` DSL, we can streamline the code by regrouping the operators per type:

```

Number.metaClass {
    multiply { Amount amount ->
amount.times(delegate) }
    div      { Amount amount ->
amount.inverse().times(delegate) }
}

Amount.metaClass {
    div <<  { Number factor ->
delegate.divide(factor) }
    div <<  { Amount factor ->
delegate.divide(factor) }
    multiply { Number factor ->
delegate.times(factor) }
    power   { Number factor ->
delegate.pow(factor) }
    negative { -> delegate.opposite() }
}

```

A metaClass() method takes a closure as single argument, containing the various definitions of the methods and properties, instead of repeating the Type.metaClass on each line. When there is just one method of a given name, use the pattern methodName { /\* closure \*/ }, but when there are several, you should use the append operator and follow the pattern methodName << { /\* closure \*/ }. Static methods can also be added through this mechanism, so instead of the classical approach:

```
// add a fqn() method to Class to get the
fully
// qualified name of the class (ie. simply
Class#getName)
Class.metaClass.static.fqn = { delegate.name
}

assert String.fqn() == "java.lang.String"
```

You can now do:

```
Class.metaClass {
    'static' {
        fqn { delegate.name }
    }
}
```

Note here that you have to quote the static keyword, to avoid this construct to look like a static initializer. For one off method addition, the classical approach is obviously more concise, but when you have several methods to add, the EMC DSL makes sense.

The usual approach for adding properties to existing classes through `ExpandoMetaClass` is to add a getter and a setter as methods. For instance, say you want to add a method that counts the number of words in a text file, you could try this:

```
File.metaClass.getWordCount = {
    delegate.text.split(/\w/).size()
}

new File('myFile.txt').wordCount
```

When there is some logic inside the getter, this is certainly the best approach, but when you just want to have new

properties holding simple values, through the `ExpandoMetaClass` DSL, it is possible to define them. In the following example, a `lastAccessed` property is added to a `Car` class — each instance will have its property. Whenever a method is called on that car, this property is updated with a newer timestamp.

```
class Car {  
    void turnOn() {}  
    void drive() {}  
    void turnOff() {}  
}  
  
Car.metaClass {  
    lastAccessed = null  
    invokeMethod = { String name, args ->  
        def metaMethod =  
delegate.metaClass.getMetaMethod(name, args)  
        if (metaMethod) {  
            delegate.lastAccessed = new  
Date()  
  
metaMethod.doMethodInvoke(delegate, args)  
        } else {  
            throw new  
MissingMethodException(name, delegate.class,  
args)  
        }  
    }  
}  
  
def car = new Car()  
println "Last accessed: ${car.lastAccessed  
?: 'Never'}"
```

```
car.turnOn()  
println "Last accessed: ${car.lastAccessed  
?: 'Never'}"
```

```
car.drive()  
sleep 1000  
println "Last accessed: ${car.lastAccessed  
?: 'Never'}"
```

```
sleep 1000  
car.turnOff()
```

```
println "Last accessed: ${car.lastAccessed  
?: 'Never'}"
```

In our example, in the DSL, we access that property through the delegate of the closure, with `delegate.lastAccessed = new Date()`. And we intercept any method call thanks to `invokeMethod()`, delegating to the original method for the call, and throwing an exception in case the method doesn't exist. Later on, you can see by executing this script that `lastAccessed` is updated as soon as we call a method on our instance.

## **ExpandoMetaClass - Dynamic Method Names**

### *ExpandoMetaClass - Dynamic method/property name creation*

Since Groovy allows you to use Strings as property names this in turns allows you to dynamically create method and property names at runtime.

#### *The Basics*

To create a method with a dynamic name simply use Groovy's feature of reference property names as strings. You can combine this with Groovy's string interpolation (Gstrings) to create method and property names on the fly:

```
class Person {  
    String name = "Fred"  
}  
  
def methodName = "Bob"  
  
Person.metaClass."changeNameTo${methodName}"  
= {-> delegate.name = "Bob" }  
  
def p = new Person()  
  
assert "Fred" == p.name  
  
p.changeNameToBob()  
  
assert "Bob" == p.name
```

The same concept can be applied to static methods and properties.

*A more elaborate example*

In Grails we have a concept of dynamic codecs, classes that can encode and decode data.

These classes are called HTMLCodec, JavaScriptCodec etc. an example of which can be seen below:

```
import
org.springframework.web.util.HtmlUtils
class HTMLCodec {
    static encode = { theTarget ->
HtmlUtils.htmlEscape(theTarget.toString())
}
    static decode = { theTarget ->
HtmlUtils.htmlUnescape(theTarget.toString())
}
}
```

So what we do with these classes is to evaluate the convention and add "encodeAsXXX" methods to every object based on the first part of the name of the codec class such as "encodeAsHTML". The pseudo code to achieve this is below:

```

def codecs = classes.findAll {
it.name.endsWith('Codec') }

codecs.each { codec ->

Object.metaClass."encodeAs${codec.name-'Code
c'}" = {
codec.newInstance().encode(delegate) }

Object.metaClass."decodeFrom${codec.name-'Co
dec'}" = {
codec.newInstance().decode(delegate) }
}

def html = '<html><body>hello</body></html>'

assert
'&lt;html&gt;&lt;body&gt;hello&lt;/body&gt;&
lt;/html&gt;' == html.encodeAsHTML()

```

As you can see from the above we dynamically construct the names of the methods using GString expressions!

## ExpandoMetaClass - GroovyObject Methods

### *ExpandoMetaClass - Overriding invokeMethod, getProperty and setProperty*

It is also possible to override the methods invokeMethod, getProperty and setProperty on ExpandoMetaClass thus allowing even more dynamic behaviour.

#### *Overriding invokeMethod*

As an example of overring invokeMethod, take this simple example:

```

class Stuff {
    def invokeMe() { "foo" }
}

Stuff.metaClass.invokeMethod = { String
name, args ->
    def metaMethod =
Stuff.metaClass.getMetaMethod(name, args)
    def result
    if(metaMethod) result =
metaMethod.invoke(delegate, args)
    else {
        result = "bar"
    }
    result
}

def stf = new Stuff()

assert "foo" == stf.invokeMe()
assert "bar" == stf.doStuff()

```

So what is happening here? Well firstly we've overridden invokeMethod by assigning it an appropriate closure, but in addition we first look-up a [MetaMethod](#) with the line:

```

def metaMethod =
delegate.class.metaClass.getMetaMethod(name)

```

A [MetaMethod](#) in Groovy is a method that is known to exist on the MetaClass whether added at runtime or whatever, thus we check if there is an existing [MetaMethod](#) and if there isn't we simply return "bar", hence the behaviour of the assert statements is correct.

### Overriding `getProperty` and `setProperty`

Again overriding `getProperty` and `setProperty` is similar to the above:

```
class Person {
    String name = "Fred"
}

Person.metaClass.getProperty = { String name
->
    def metaProperty =
Person.metaClass.getMetaProperty(name)
    def result
    if(metaProperty) result =
metaProperty.getProperty(delegate)
    else {
        result = "Flintstone"
    }
    result
}

def p = new Person()

assert "Fred" == p.name
assert "Flintstone" == p.other
```

The important thing to note here is that instead of a `MetaMethod` we look-up a [MetaProperty](#) instance if that exists we call the `getProperty` method of the [MetaProperty](#) passing the delegate (ie the instance of the class).

The only different with `setProperty` is you need the value in the method signature and to call `setProperty` on the `MetaProperty`:

```

Person.metaClass.setProperty = { String
name, value ->
    ...
    if(metaProperty)
metaProperty.setProperty(delegate, value)
    ...
}

```

#### *Useful References*

- [MetaMethod](#)
- [MetaProperty](#)

#### **ExpandoMetaClass - Interfaces**

##### *ExpandoMetaClass - Adding methods to interfaces*

It is possible to add methods onto interfaces with ExpandoMetaClass. To do this however, it MUST be enabled globally using the `ExpandoMetaClass.enableGlobally()` method before application start-up.

As an example this code adds a new method to all implementors of `java.util.List`:

```

List.metaClass.sizeDoubled = {->
delegate.size() * 2 }

def list = []

list << 1
list << 2

assert 4 == list.sizeDoubled()

```

Another example taken from Grails, this code allows access to session attributes using Groovy's subscript operator to all implementors of the `HttpSession` interface:

```

HttpSession.metaClass.getAt = { String key
->
    delegate.getAttribute(key)
}
HttpSession.metaClass.putAt = { String
key, Object val ->
    delegate.setAttribute(key, val)
}

def session = new MockHttpSession()

session.foo = "bar"

```

ExpandoMetaClass - Methods

## ExpandoMetaClass - Adding & Overriding instance methods

Once you have an ExpandoMetaClass to add new methods to it is trivial:

```

class Book {
    String title
}

Book.metaClass.titleInUpperCase << {->
title.toUpperCase() }

def b = new Book(title:"The Stand")

assert "THE STAND" == b.titleInUpperCase()

```

Note that in this case the left shift << operator is used to "append" the new method. If the method already exists an

exception will be thrown. If you want to replace an instance method you can use the = operator:

```
Book.metaClass.toString = {->  
title.toUpperCase() }
```

### **ExpandoMetaClass - Overriding static invokeMethod**

#### ***ExpandoMetaClass - Overriding invokeMethod for static methods***

It is also possible to override invokeMethod for static methods.

#### ***Overriding invokeMethod for static***

As an example of overriding invokeMethod for static methods, take this simple example:

```

class Stuff {
    static invokeMe() { "foo" }
}

Stuff.metaClass.'static'.invokeMethod = {
String name, args ->
    def metaMethod =
Stuff.metaClass.getStaticMetaMethod(name,
args)
    def result
    if(metaMethod) result =
metaMethod.invoke(delegate, args)
    else {
        result = "bar"
    }
    result
}

assert "foo" == Stuff.invokeMe()
assert "bar" == Stuff.doStuff()

```

So what is happening here? Well firstly we've overridden `invokeMethod` using the 'static' qualifier and by assigning it an appropriate closure, but in addition we first look-up a [MetaMethod](#) with the line:

```

def metaMethod =
delegate.class.metaClass.getStaticMetaMethod
(name)

```

A [MetaMethod](#) in Groovy is a method that is known to exist on the MetaClass whether added at runtime or whatever, thus we check if there is an existing [MetaMethod](#) and if there isn't we simply return "bar", hence the behaviour of the assert statements is correct.

#### Useful References

- [MetaMethod](#)
- [MutableMetaClass](#)
- [MetaObjectProtocol](#)

#### ExpandoMetaClass - Properties

## Adding properties

Properties can be added in a couple of ways. Firstly you can use the instance method syntax seen previously:

```
class Book {  
  String title  
}  
Book.metaClass.getAuthor << {-> "Stephen  
King" }  
  
def b = new Book("The Stand")  
  
assert "Stephen King" == b.author
```

In this case the property is dictated by the closure and is a read-only property. You can add the equivalent setter but then remember you will have to store the property somewhere for retrieval later so make sure you use thread safe code. For example you could store values in a synchronized Map using the object identity as the key:

```

def properties =
Collections.synchronizedMap([:])

Book.metaClass.setAuthor = { String value ->
properties[System.identityHashCode(delegate)
+ "author"] = value
}
Book.metaClass.getAuthor = {->
properties[System.identityHashCode(delegate)
+ "author"]
}

```

This is not the only technique however. For example in a servlet container you may store the values in the currently executing request as request attributes (as is done in some cases in Grails).

Alternatively you can simply assign a value as follows:

```

Book.metaClass.author = "Stephen King"
def b = new Book("The Stand")

assert "Stephen King" == b.author

```

In this case the property is mutable and has both a setter and getter.

## ExpandoMetaClass - Runtime Discovery

### *ExpandoMetaClass - Runtime MetaClass Analysis*

At runtime it is often useful to know what other methods or properties exist at the time the method is executed. To this end ExpandoMetaClass provides the following methods as of this writing:

- getMetaMethod
- hasMetaMethod

- getMetaProperty
- hasMetaProperty

Why can't you just use reflection? Well because Groovy is different, it has the methods that are "real" methods and methods that are available only at runtime. These are sometimes (but not always) represented as MetaMethods. The MetaMethods tell you what methods are available at runtime, thus your code can adapt.

This is of particular use when [overriding invokeMethod, getProperty and/or setProperty](#) for example:

```
class Stuff {
    def invokeMe() { "foo" }
}

Stuff.metaClass.invokeMethod = { String
name, args ->
    def metaMethod =
Stuff.metaClass.getMetaMethod(name, args)
    def result
    if(metaMethod) result =
metaMethod.invoke(delegate, args)
    else {
        result = "bar"
    }
    result
}

def stf = new Stuff()

assert "foo" == stf.invokeMe()
assert "bar" == stf.doStuff()
```

Here we are using the getMetaMethod method to obtain a reference to a method that may or may not exist. If it doesn't exist the getMetaMethod method will return null and the code can adapt to this fact.

#### *Useful References*

- [MetaMethod](#)

- [MetaProperty](#)

## ExpandoMetaClass - Static Methods

### *ExpandoMetaClass - Adding static methods*

Static methods can also be added using the same technique as instance methods with the addition of the "static" qualifier before the method name:

```
class Book {  
    String title  
}  
  
Book.metaClass.static.create << { String  
title -> new Book(title:title) }  
  
def b = Book.create("The Stand")
```

## Global AST Transformations

Groovy 1.6 offers several approaches to transforming the AST of code within the compiler. You can write a [custom AST visitor](#), you can use annotations and a [local AST transformation](#), or you can use a global AST transformation.

This page explains how to write and debug a global AST transformation.

Sticking with the naive and simple example from the [local transformation](#) page, consider wanting to provide console output at the start and stop of method calls within your code. The following "Hello World" example would actually print "Hello World" along with a start and stop message:

```
def greet() {  
    println "Hello World"  
}  
  
greet()
```

Not a great use case, but it is useful to explain the mechanics of global transformations.

A global transformation requires four steps: 1) write an ASTTransformation subclass, 2) create a Jar metadata file containing the name of your ASTTransformation, 3) create a Jar containing the class and metadata, and 4) invoke groovy with that Jar on your classpath.

### Writing an ASTTransformation

This is almost exactly the same step you'll need if writing a local transformation. You must define an [ASTTransformation](#) subclass that reads, and possibly rewrites, the syntax tree of the compiling code. Here is the transformation that will add a console start message and end message to all method invocations:

```
@GroovyASTTransformation(phase=CompilePhase.
CONVERSION)
public class LoggingASTTransformation
implements ASTTransformation {

    static final def TARGET =
WithLogging.getName()

    public void visit(ASTNode[] astNodes,
SourceUnit sourceUnit) {
        List methods =
sourceUnit.getAST()?.getMethods()
        methods?.each { MethodNode method ->
            Statement startMessage =
createPrintlnAst("Starting $method.name")
            Statement endMessage =
createPrintlnAst("Ending $method.name")

            List existingStatements =
method.getCode().getStatements()
            existingStatements.add(0,
startMessage)

            existingStatements.add(endMessage)
        }
    }
}
```

```
private Statement
createPrintlnAst(String message) {
    return new ExpressionStatement(
        new MethodCallExpression(
            new
VariableExpression("this"),
            new
ConstantExpression("println"),
            new ArgumentListExpression(
                new
ConstantExpression(message)
            )
        )
    )
}
```

```
}  
    }  
}
```

The first line ([@GroovyASTTransformation](#)) line tells the Groovy compiler that this is an AST transformation that should occur in the conversion [CompilePhase](#). Unlike local transformations, global transformations can occur in any phase.

The public `visit(ASTNode[], SourceUnit)` method is invoked for each source unit compiled. In this example, I'm just pulling out all the methods defined in the source. A method to the compiler is simply a list of [Statement](#) objects, so I'm adding a statement zero logging the start message and appending a statement to the end of the list with the end message.

Notice the complexity in creating a simple `println` Statement in the `createPrintInAst` method. A method call has a target(`this`), a name(`println`), and an argument list(the message). An easy way to create AST is to write the Groovy code you expect to create, then observe what AST the compiler generates within the IDE's debugger. This requires a test harness with a custom `GroovyClassLoader` and an [AST Visitor](#).

#### ***Writing Jar Metadata***

The Groovy compiler discovers your `ASTTransformation` through a file named "org.codehaus.groovy.transform.ASTTransformation". This file must contain the fully qualified package and name of your transformation. In my example, the file simply has one line:

```
gep.LoggingASTTransformation
```

#### ***Creating the Jar***

The `ASTTransformation` and the metadata must be packaged into a single Jar file. The `org.codehaus.groovy.transform.ASTTransformation` file must be in the `META-INF/services` directory. The Jar layout for this example follows:

```
LogMethodTransform.jar  
--gep  
----LoggingASTTransformation.class  
----LoggingASTTransformation$_visit_closure1  
.class  
--META-INF  
----services  
-----org.codehaus.groovy.transform.ASTTrans  
formation
```

#### *Compiling the Example*

The new Jar must be put on the groovyc classpath for the transformation to be invoked. If the sample script at the top of the post is in a file named "LoggingExample.groovy", then the command line to compile this is:

```
groovyc -cp LogMethodTransform.jar  
LoggingExample.groovy
```

This generates a LoggingExample.class that, when run with Java, produces:

```
Starting greet  
Hello World  
Ending greet
```

#### *Debugging Global Transformations*

Local transformations are simple to debug: the IDE (at least IDEA) supports it with no extra effort. Global transformations are not so easy. To test this you might write a test harness that invoked LoggingASTTransformation on a file explicitly. The test harness source is [available](#) and could easily be modified to fit your needs. Let me know if you know an easier way to debug this!

#### *Applying your transformation on custom file extensions*

Since Groovy 1.7.5, you are allowed to add your own file types to the groovy compiler. This can be especially useful when you have written a DSL which requires extensive AST transformations and that you want to avoid your transformation to be applied on regular groovy files. For example, imagine your DSL specifies user stories :

```
story "user enters invalid postcode" {  
  
    given ....  
  
    expect ...  
  
}
```

Then you could allow this syntax in .story files only. To do this, you must first have registered your global AST transformation into the '*META-INF/services/org.codehaus.groovy.transform.ASTTransformation*' file like explained before. The second step is to register your custom file extension into the '*META-INF/services/org.codehaus.groovy.source.Extensions*' file :

```
story
```

The last step is to update your AST transformation so that it checks the file extension :

```
void visit(ASTNode[] nodes, SourceUnit  
source) {  
    if (!source.name.endsWith('.story')) {  
        return  
    }  
    // start transforming the AST  
  
}
```

Now you have a global AST transformation that will only apply on .story files.

### Local AST Transformations

Groovy 1.6 provides two options for hooking into the Groovy compiler for compile-time metaprogramming: local and global AST transformations. This page explains how to write and debug a local AST transformation.

As a naive and simple example, consider wanting to write a `@WithLogging` annotation that would add console

messages at the start and end of a method invocation. So the following "Hello World" example would actually print "Hello World" along with a start and stop message:

```
@WithLogging  
def greet() {  
    println "Hello World"  
}  
  
greet()
```

A poor man's aspect oriented programming, if you will.

A local AST transformation is an easy way to do this. It requires two things: a definition of the `@WithLogging` annotation and an implementation of [ASTTransformation](#) that adds the logging expressions to the method.

An `ASTTransformation` is a callback that gives you access to the [SourceUnit](#), through which you can get a reference to the [AST](#). The AST is a tree structure of [Expression](#) objects that the source code has been transformed into. An easy way to learn about the AST is to explore it in a debugger, which will be shown shortly. Once you have the AST, you can analyze it to find out information about the code or rewrite it to add new functionality.

The local transformation annotation is the simple part. Here is the `@WithLogging` one:

```
import  
org.codehaus.groovy.transform.GroovyASTTransformationClass  
  
@Retention(RetentionPolicy.SOURCE)  
@Target([ElementType.METHOD])  
@GroovyASTTransformationClass(["gep.LoggingASTTransformation"])  
public @interface WithLogging {  
}
```

The annotation retention can be `SOURCE`, you won't need the annotation past that. The element type here is `METHOD`, the `@WithLogging` annotation applies to methods. But the most important part is the

@GroovyASTTransformationClass annotation. This links the @WithLogging annotation to the ASTTransformation subclass you will write. gep.LoggingTransformation is the full package and class of my ASTTransformation. This line wires the annotation to the transformation.

With this in place, the Groovy compiler is going to try to invoke gep.LoggingASTTransformation every time an @WithLogging is found in a source unit. Any breakpoint set within LoggingASTTransformation will now be hit within the IDE when running the sample script.

The ASTTransformation subclass is a little more complex. Here is the very simple, and very naive, transformation to add a method start and stop message for @WithLogging:

```
@GroovyASTTransformation(phase=CompilePhase.  
SEMANTIC_ANALYSIS)  
public class LoggingASTTransformation  
implements ASTTransformation {  
  
    public void visit(ASTNode[] nodes,  
SourceUnit sourceUnit) {  
        List methods =  
sourceUnit.getAST()?.getMethods()  
        // find all methods annotated with  
@WithLogging  
        methods.findAll { MethodNode method  
->  
            method.getAnnotations(new  
ClassNode(WithLogging))  
        }.each { MethodNode method ->  
            Statement startMessage =  
createPrintlnAst("Starting $method.name")  
            Statement endMessage =  
createPrintlnAst("Ending $method.name")  
  
            List existingStatements =  
method.getCode().getStatements()  
            existingStatements.add(0,
```

```
startMessage)
```

```
existingStatements.add(endMessage)
```

```
    }
```

```
  }
```

```
    private Statement
```

```
createPrintlnAst(String message) {
```

```
    return new ExpressionStatement(
```

```
        new MethodCallExpression(
```

```
            new
```

```
VariableExpression("this"),
```

```
            new
```

```
ConstantExpression("println"),
```

```
                new ArgumentListExpression(
```

```
                    new
```

```
ConstantExpression(message)
```

```
                )
```

```
            )
```

```
    )  
  }  
}
```

Starting at the top...

The `@GroovyASTTransformation(phase=CompilePhase.SEMANTIC_ANALYSIS)` line tells the Groovy compiler that this is a local transformation that applies to the SEMANTIC\_ANALYSIS [CompilePhase](#). Local transformations can only be applied at semantic analysis or later phases, and this line is required!

The public `visit(ASTNode[], SourceUnit)` method is called once per annotated node (class, method, or field; method parameters don't seem to be supported). The first element in the ASTNode array holds the annotation, the second one the annotated node. The AST you receive is not for the `@WithLogging` annotated method, it is for the entire file that contains `@WithLogging`. This example is just using `findAll` to locate methods that are annotated with `@WithLogging`, then using an `each` statement to wrap any annotated method with `println` lines. A method to the compiler is simply a list of Statement objects, so the example adds a statement zero logging the start message and appending a statement to the list with the end message.

Note the creation of the new `println` statements in the `createPrintlnAst(String)` method. Creating AST for code is not always simple. In this case we need to construct a new method call, passing in the receiver/variable, the name of the method, and an argument list. When creating AST, it might be helpful to write the code you're trying to create in a Groovy file and then inspect the AST of that code in the debugger to learn what to create. Then write a function like `createPrintlnAst` using what you learned through the debugger.

The final result:

```
@WithLogging  
def greet() {  
    println "Hello World"  
}  
  
greet()
```

Produces:

```
Starting greet  
Hello World  
Ending greet
```

## Per-Instance MetaClass

### Adding Methods to an Instance

Normally when you add a `MetaMethod`, it is added for *all instances* of that class. However, you can dynamically add methods to individual instances by giving that instance its own `MetaClass`.

*For Groovy version 1.6 and higher - any Groovy or Java object*

```
def str = "hello"
str.metaClass.test = { "test" }
assert "test" == str.test()
```

*For Groovy prior to version 1.6 - GroovyObjects only!*

```
def test = "test"
def gstr = "hello $test"      // this is a
                              GString, which implements GroovyObject

def emc = new ExpandoMetaClass( gstr.class,
                                false )
emc.test = { println "test" }
emc.initialize()

gstr.metaClass = emc
gstr.test()                  // prints
"test"
```

Note that you **cannot** do this:

```
gstr.metaClass = new ExpandoMetaClass(
gstr.class )
gstr.metaClass.test = { println "test" }
```

because you *must* call `emc.initialize()` before making any method calls on the instance. But you can't add MetaMethods after calling `initialize()`! This is bit of a catch 22 because the `ExpandoMetaClass` is intercepting methods to itself. The solution is (as shown in the first example) to simply add the MetaMethods before assigning the new MetaClass to your instance.

The other option is to set the set `emc.allowChangesAfterInit = true`. This will allow you to add additional methods on the MetaClass after it is in use.

**Note**

Be sure to use the proper constructor, `new ExpandoMetaClass(MyClass, false)`. The `false` parameter keeps the MetaClass from being inserted into the Registry. Otherwise your new MetaClass will be used for all instances of `MyClass`, not just the instance it is assigned to.

**Compatibility**

Only works in Groovy 1.1-beta-3 and above. Use the `Proxy` class in older versions to achieve a per-instance behaviour change.

***If your Instance is not a GroovyObject***

If your instance is a plain Java type, it will not implement `GroovyObject`, and consequently, will not have a `metaClass` property. In this case you must wrap your instance in a [groovy.util.Proxy](#):

```
ExpandoMetaClass emc = new ExpandoMetaClass(  
Object, false )  
emc.boo = { "Surprise!" }  
emc.initialize()  
  
def obj = new groovy.util.Proxy().wrap( new  
Object() )  
obj.setMetaClass( emc )  
assert obj.boo() == "Surprise!"
```

Note that this example is calling the `setMetaClass(..)` method rather than using the property notation in the previous example. This is because `Proxy` intercepts *method calls* only, not property access.

**Runtime mixins**

Last metaprogramming feature we'll cover today: runtime mixins. `@Mixin` allowed you to mixin new behavior to classes you owned and were designing. But you could not mixin anything to types you didn't own. Runtime mixins

propose to fill that gap by letting you add a mixin on any type at runtime. If we think again about our example of vehicles with some mixed-in capabilities, if we didn't *own* James Bond's vehicle and give it some diving ability, we could use this mechanism:

```
// provided by a third-party
interface Vehicle {
    String getName()
}

// provided by a third-party
class JamesBondVehicle implements Vehicle {
    String getName() { "James Bond's
vehicle" }
}

JamesBondVehicle.mixin DivingAbility,
FlyingAbility

assert new JamesBondVehicle().fly() ==
    "I'm the James Bond's vehicle and I
fly!"
assert new JamesBondVehicle().dive() ==
    "I'm the James Bond's vehicle and I
dive!"
```

One or more mixins can be passed as argument to the static `mixin()` method added by Groovy on `Class`.

### Using `invokeMethod` and `getProperty`

#### Using `invokeMethod` & `getProperty`

Since 1.0, Groovy supports the ability to intercept *all* method and property access via the `invokeMethod` and `get/setProperty` hooks. If you only want to intercept failed method/property access take a look at [Using `methodMissing` and `propertyMissing`](#).

#### *Overriding `invokeMethod`*

In any Groovy class you can override `invokeMethod` which will essentially intercept all method calls (to intercept calls to existing methods, the class additionally has to implement the `GroovyInterceptable` interface). This makes it possible to construct some quite interesting DSLs and builders.

For example a trivial `XmlBuilder` could be written as follows (note Groovy ships with much richer XML APIs and this just serves as an example):

```
class XmlBuilder {
    def out
    XmlBuilder(out) { this.out = out }
    def invokeMethod(String name, args) {
        out << "<$name>"
        if(args[0] instanceof Closure) {
            args[0].delegate = this
            args[0].call()
        }
        else {
            out << args[0].toString()
        }
        out << "</$name>"
    }
}

def xml = new XmlBuilder(new StringBuffer())
xml.html {
    head {
        title "Hello World"
    }
    body {
        p "Welcome!"
    }
}
```

Another simple usage of `invokeMethod` is to provide simple AOP style around advice to existing methods. Here is a simple logging example implemented with `invokeMethod`:

```
class MyClass implements GroovyInterceptable
{
    def invokeMethod(String name, args) {

        System.out.println ("Beginning
$name")
        def metaMethod =
metaClass.getMetaMethod(name, args)
        def result = metaMethod.invoke(this,
args)
        System.out.println ("Completed
$name")
        return result
    }
}
```

#### *Overriding getProperty and setProperty*

You can also override property access using the `getProperty` and `setProperty` property access hooks. For example it is possible to write a trival "Expandable" object using this technique:

```
class Expandable {
    def storage = [:]
    def getProperty(String name) {
storage[name] }
    void setProperty(String name, value) {
storage[name] = value }
}

def e = new Expandable()
e.foo = "bar"
println e.foo
```

## Using methodMissing and propertyMissing

### Using methodMissing & propertyMissing

Since 1.5, Groovy supports the concept of "methodMissing". This differs from [invokeMethod](#) in that it is only invoked in the case of failed method dispatch.

There are a couple of important aspects to this behaviour:

1. Since method/propertyMissing only occur in the case of failed dispatch, they are expensive to execute
2. Since method/propertyMissing aren't intercepting EVERY method call like [invokeMethod](#) they can be more efficient with a few meta-programming tricks

### *Using methodMissing with dynamic method registration*

Typically when using methodMissing the code will react in some way that makes it possible for the next time the same method is called, that it goes through the regular Groovy method dispatch logic.

For example consider [dynamic finders](#) in GORM. These are implemented in terms of methodMissing. How does it work? The code resembles something like this:

```

class GORM {

    def dynamicMethods = [...] // an array of
dynamic methods that use regex
    def methodMissing(String name, args) {
        def method = dynamicMethods.find {
it.match(name) }
        if(method) {
            GORM.metaClass."$name" = {
Object[] varArgs ->
                method.invoke(delegate, name,
varArgs)
            }
            return
method.invoke(delegate, name, args)
        }
        else throw new
MissingMethodException(name, delegate, args)
    }
}

```

Notice how, if we find a method to invoke then we dynamically register a new method on the fly using [ExpandoMeta Class](#). This is so that the next time the same method is called it is more efficient. This way methodMissing doesn't have the overhead of invokeMethod AND is not expensive for the second call

#### *Using propertyMissing*

Groovy also supports propertyMissing for dealing with property resolution attempts. For a getter you use a propertyMissing definition that takes a String argument:

```
class Foo {
    def propertyMissing(String name) { name }
}
def f = new Foo()

assertEquals "boo", f.boo
```

For a setters you add a second propertyMissing definition that takes a value argument:

```
class Foo {
    def storage = [:]
    def propertyMissing(String name, value) {
storage[name] = value }
    def propertyMissing(String name) {
storage[name] }
}
def f = new Foo()
f.foo = "bar"

assertEquals "bar", f.foo
```

As with methodMissing you will likely want to dynamically register new properties at runtime to improve the performance of you code.

#### *Static methods and properties*

You can add methodMissing and propertyMissing that deals with static methods and properties via [ExpandoMetaClasses](#)

## Ant Integration with Groovy

### Introduction

[Ant](#) is the predominant build environment for Java projects. Groovy is a leading Scripting language for the JVM. The good news is that you can use them together easily and with many benefits.

★ **Existing Ant Users (Java Projects):** If you are already familiar with using Ant with a traditional `build.xml` file, then you can continue to do so with almost no changes. If you want you can begin to use the [<groovy>](#) Ant task to bring the full power of a scripting language to your build scripts.

★ **Existing Ant Users (Groovy or mixed Groovy/Java Projects):** If you wish to pre-compile your Groovy scripts as part of your build process (it can help catch syntactic errors earlier if you do) then you need to know about the [<groovyc>](#) Ant task. You will find it almost identical to the `javac` task which you are probably already familiar with.

★ **Existing Groovy Users:** You probably want to consider how you can [use AntBuilder](#) to leverage the many available Ant tasks directly in your code using a DSL-style notation. You can [leverage Ant Libraries](#) in the same way.

## Further Information

See also:

- [Ant Task Troubleshooting](#)
- [Ant Manual](#)

A few articles related to this topic:

- [Build scripts with Groovy and Ant](#)
- [Practically Groovy: Ant scripting with Groovy](#)
- [Scripting a Groovy Ant](#)
- [Using Groovy to Send Emails: AntBuilder](#)
- [Using WebTest with AntBuilder](#)

## The groovy Ant Task

`<groovy>`

### Description

Executes a series of Groovy statements. Statements can either be read in from a text file using the `src` attribute or from between the enclosing Groovy tags.

### Required taskdef

Assuming `groovy-all-VERSION.jar` is in `my.classpath` you will need to declare this task at some point in the `build.xml` prior to using this task.

```
<taskdef name="groovy"  
  
  classname="org.codehaus.groovy.ant.Groovy"  
  classpathref="my.classpath" />
```

`<groovy>` attributes

Attribute	Description	Required
-----------	-------------	----------

src	File containing Groovy statements. The directory containing the file is added to the classpath	Yes, unless statements enclosed within tags
classpath	the classpath to use	No
classpathref	the classpath to use, given as reference to a PATH defined elsewhere	No

#### Parameters specified as nested elements

##### <classpath>

Groovy's classpath attribute is a PATH like structure and can also be set via a nested classpath element.

##### <arg> (since 1.1)

Arguments can be set via one or more nested <arg> elements using the standard Ant [command line conventions](#).

#### Available bindings

A number of bindings are in scope for use within your Groovy statements.

Name	Description
ant	an instance of AntBuilder that knows about the current ant project
project	the current ant project
properties	a Map of ant properties
target	the owning target that invoked this groovy script
task	the wrapping task, can access anything needed in <a href="#">org.apache.tools.ant.Task</a>
args	command line arguments, if any

#### Examples

Hello world, version 1:

```
<groovy>
println "Hello World"
</groovy>
```

Hello world, version 2:

```
<groovy>  
ant.echo "Hello World"  
</groovy>
```

List all xml files in the current directory:

```
<groovy>  
xmlfiles = new  
File(".").listFiles().findAll{ it =~  
"\.xml$" }  
xmlfiles.sort().each { println it.toString()  
}  
</groovy>
```

List all xml files within a jar:

```
<zipfileset id="found" src="foobar.jar"  
includes="**/*.xml" />  
<groovy>  
    project.references.found.each {  
        println it.name  
    }  
</groovy>
```

To run a script:

```
<groovy  
src="/some/directory/some/file.groovy">  
  <classpath>  
    <pathelement  
location="/my/groovy/classes/directory" />  
  </classpath>  
</groovy>
```

To find all the 'Builder' classes having an 'org.\*' package within a directory of jars:

```

<property name="local.target"
value="C:/Projects/GroovyExamples"/>
<groovy>
import java.util.jar.JarFile
def classes = []
def resourceNamePattern =
/org\/*.*/.*Builder.class/
def jarNamePattern =
/.*(beta|commons).*jar$/

def libdir = new
File("${properties['local.target']}/lib")
libdir.listFiles().grep(~jarNamePattern).each {
  candidate ->
    new JarFile(candidate).entries().each {
      entry ->
        if (entry.name =~
resourceNamePattern) classes += entry.name
    }
}
properties["builder-classes"] =
classes.join(' ')
</groovy>
<echo message='${builder-classes}' />

```

Which might result in something like:

```
org/apache/commons/cli/PatternOptionBuilder.  
class  
org/apache/commons/cli/OptionBuilder.class  
org/codehaus/groovy/tools/groovydoc/GroovyRo  
otDocBuilder.class  
org/custommonkey/xmlunit/HTMLDocumentBuilder  
.class  
org/custommonkey/xmlunit/TolerantSaxDocument  
Builder.class
```

FileScanner version of above (with a slight variation on collecting the names):

```

<groovy>
import java.util.jar.JarFile
def resourceNamePattern =
/org\/*.*/.*Builder.class/
def candidates = ant.fileScanner {
    fileset(dir: '${local.target}/lib') {
        include(name: '*beta*.jar')
        include(name: '*commons*.jar')
    }
}
def classes = candidates.collect {
    new JarFile(it).entries().collect {
it.name }.findAll {
    it =~~ resourceNamePattern
}
}.flatten()
properties["builder-classes"] =
classes.join(' ')
</groovy>

```

Here is an example of calling out to a web service from your Ant script:

```

<?xml version="1.0" encoding="UTF-8"?>
<project name="SOAP example" default="main"
basedir=".">
    <property environment="env"/>
    <property name="celsius" value="0"/>
    <target name="main">
        <taskdef name="groovy"
classname="org.codehaus.groovy.ant.Groovy">

```

```

        <classpath>
            <fileset
dir="{env.GROOVY_HOME}"
includes="embeddable/groovy-all-*.jar,lib/ivy*.jar"/>
        </classpath>
    </taskdef>
    <groovy>

@Grab('org.codehaus.groovy.modules.groovyws:
0.5.1')

        import groovyx.net.ws.WSClient
        def url =
'http://www.w3schools.com/webservices/tempco
nvert.asmx?WSDL'
        def proxy = new WSClient(url,
this.class.classLoader)
        proxy.initialize()
        ant.echo "I'm freezing at
${properties.celsius} degrees Celsius"
        properties.result =
proxy.CelsiusToFahrenheit(properties.celsius
)

        </groovy>
        <antcall target="results"/>
    </target>
    <target name="results">
        <echo message="I'm freezing at

```

```
${result} degrees Fahrenheit"/>
  </target>
</project>
```

which will output the following (along with some informational messages):

```
main:
    ...
    [echo] I'm freezing at 0 degrees
Celsius
results:
    [echo] I'm freezing at 32 degrees
Fahrenheit

BUILD SUCCESSFUL
```

Setting arguments

```
<target name="run">
  <groovy>
    <arg line="1 2 3"/>
    <arg value="4 5"/>
    println args.size()
    println args[2]
    args.each{ ant.echo(message:it)
  }
  </groovy>
</target>
```

```
Buildfile: GROOVY-2087.xml
```

```
run:
```

```
[groovy] 4  
[groovy] 3  
[echo] 1  
[echo] 2  
[echo] 3  
[echo] 4 5
```

```
BUILD SUCCESSFUL
```

#### Forking Groovy

Since 1.5.7 and 1.6-beta-2, `<groovy>` also supports a `fork="true"` attribute. In fact, many of the attributes from the `<java>` Ant task are supported. More details to come ...

#### More examples

- [Ant, Groovy and the Database](#)

### The groovyc Ant Task

```
<groovyc>
```

#### Description

Compiles Groovy source files and, if the joint compilation option is used, Java source files.

#### Required taskdef

Assuming `groovy-all-VERSION.jar` is in `my.classpath` you will need to declare this task at some point in the `build.xml` prior to the `groovyc` task being invoked.

```
<taskdef name="groovyc"  
  
  classname="org.codehaus.groovy.ant.Groovyc"  
  classpathref="my.classpath" />
```

`<groovyc>` Attributes

Attribute	Description	Required
srcdir	Location of the Groovy (and possibly Java) source files.	Yes
destdir	Location to store the class files.	Yes
classpath	The classpath to use.	No
classpathref	The classpath to use given as a path references.	No
sourcepath	The sourcepath to use.	No
sourcepathref	The sourcepath to use given as a path reference.	No
encoding	Encoding of source files.	No
verbose	Asks the compiler for verbose output; defaults to no.	No
includeAntRuntime	Whether to include the Ant run-time libraries in the classpath; defaults to yes.	No
includeJavaRuntime	Whether to include the default run-time libraries from the executing VM in the classpath; defaults to no.	No
fork	Whether to execute groovyc using a spawned instance of the JVM; defaults to no.	No
memoryInitialSize	The initial size of the memory for the underlying VM, if using fork mode; ignored otherwise. Defaults to the standard VM memory setting. (Examples: 83886080, 81920k, or 80m)	No
memoryMaximumSize	The maximum size of the memory for the underlying VM, if using fork mode; ignored otherwise. Defaults to the standard VM memory setting. (Examples: 83886080, 81920k, or 80m)	No

failonerror	Indicates whether compilation errors will fail the build; defaults to true.	No
listfiles	Indicates whether the source files to be compiled will be listed; defaults to no.	No
stacktrace	if true each compile error message will contain a stacktrace	No
jointCompilationOptions*	Enable joint compilation, specifying the command line options. (Using a nested javac task is preferred.)	No
indy	Enable compilation with the "invoke dynamic" support when using Groovy 2.0 and beyond and running on JDK 7	No

**Notes:** Joint compilation is only available since 1.1-beta-2, jointCompilationOptions is no longer supported, use the nested javac instead

**<groovyc> Nested Elements**

element	kind	Required	Replaces Attribute
src	a path structure	Yes (unless srcdir is used)	srcdir
classpath	a path structure	No	classpath
javac	javac task	No	jointCompilationOptions

**Notes:**

- For path structures see for example <http://ant.apache.org/manual/using.html#path>
- For usages of the javac task see <http://ant.apache.org/manual/CoreTasks/javac.html>
- The nested javac task behaves more or less as documented for the top-level javac task. srcdir, destdir, classpath, encoding for the nested javac task are taken from the enclosing groovyc task. If these attributes are specified then they are added, they do not replace. In fact, you should not attempt to overwrite the destination. Other attributes and nested elements are unaffected, for example fork, memoryMaximumSize, etc. may be used freely.

**Joint Compilation**

Joint compilation means that the Groovy compilation will parse the Groovy source files, create stubs for all of them, invoke the Java compiler to compile the stubs along with Java sources, and then continue compilation in the normal Groovy compiler way. This allows mixing of Java and Groovy files without constraint.

To invoke joint compilation with the jointCompilationOptions attribute, you have to simulate the command line with compiler switches. -j enables the joint compilation mode of working. Flags to the Java compiler are presented to the Groovy compiler with the -F option. So, for example, flags like nowarn are specified with -Fnowarn. Options to the

Java compiler that take values are presented to the Groovy compiler using -J options. For example -Jtarget=1.4 -Jsource=1.4 is used to specify the target level and source level. So a complete joinCompilationOptions value may look like: "-j -Fnowarn -Jtarget=1.4 -J-source=1.4". Clearly, using this way of specifying things is a real nuisance and not very Ant-like. In fact there are thoughts to deprecate this way of working and remove it as soon as is practical.

The right way of working is, of course, to use a nested tag and all the attributes and further nested tags as required. It is rare to specify srcdir and destdir, the nested javac task is provided with the srcdir and destdir values from the enclosing groovyc task, and it is invariable the right thing to do just to leave this as is. Here is an example:

```
<groovyc srcdir="${testSourceDirectory}"
  destdir="${testClassesDirectory}">
  <classpath>
    <pathelement
  path="${mainClassesDirectory}" />
    <pathelement
  path="${testClassesDirectory}" />
    <path refid="testPath" />
  </classpath>
  <javac source="1.4" target="1.4"
  debug="on" />
</groovyc>
```

To restate: the javac task gets the srcdir, destdir and classpath from the enclosing groovyc task.

## The groovydoc Ant task

`<groovydoc>`

### Description

Generates documentation from Groovy and Java source files.

### Required taskdef

Assuming groovy-all-VERSION.jar is in *my.classpath* you will need to declare this task at some point in the build.xml prior to the groovydoc task being invoked.

```

<taskdef name          = "groovydoc"
          classname     =
"org.codehaus.groovy.ant.Groovydoc"
          classpathref = "my.classpath"/>

```

#### <groovydoc> Attributes

Attribute	Description	Required
destdir	Location to store the class files.	Yes
sourcepath	The sourcepath to use.	No
packagenames	Comma separated list of package files (with terminating wildcard).	No
use	Create class and package usage pages.	No
windowtitle	Browser window title for the documentation (text).	No
doctitle	Include title for the package index(first) page (html-code).	No
header	Include header text for each page (html-code).	No
footer	Include footer text for each page (html-code).	No
overview	Read overview documentation from HTML file.	No
private	Show all classes and members (i.e. including private ones) if set to "true".	No

#### <groovydoc> Nested Elements

##### *link*

Create link to groovydoc/javadoc output at the given URL.

Attribute	Description	Required
packages	Comma separated list of package prefixes	Yes
href	Base URL of external site	Yes

Example #1 - <groovydoc> Ant task

```

<taskdef name          = "groovydoc"
          classname     =
"org.codehaus.groovy.ant.Groovydoc"
          classpathref  =
"path_to_groovy_all"/>

<groovydoc destdir     =
"${docsDirectory}/gapi"
          sourcepath   =
"${mainSourceDirectory}"
          packagenames = "**.*"
          use           = "true"
          windowtitle  = "${title}"
          doctitle     = "${title}"
          header       = "${title}"
          footer       = "${docFooter}"
          overview     =
"src/main/overview.html"
          private      = "false">
  <link
packages="java.,org.xml.,javax.,org.xml."
href="http://download.oracle.com/javase/6/do
cs/api"/>
  <link

```

```
packages="org.apache.tools.ant."  
href="http://evgeny-goldin.org/javadoc/ant/api"/>
```

```
<link
```

```
packages="org.junit.,junit.framework."  
href="http://kentbeck.github.com/junit/javadoc/latest"/>
```

```
<link
```

```
packages="groovy.,org.codehaus.groovy."  
href="http://groovy.codehaus.org/api"/>
```

```
<link
```

```
packages="org.codehaus.gmaven."
```

```
href="http://evgeny-goldin.org/javadoc/gmave  
n" />  
</groovydoc>
```

Example #2 - Executing <groovydoc> from Groovy

```
def ant = new AntBuilder()  
ant.taskdef(name: "groovydoc", classname:  
"org.codehaus.groovy.ant.Groovydoc")  
ant.groovydoc(  
    destdir      : "${docsDirectory}/gapi",  
    sourcepath   : "${mainSourceDirectory}",  
    packagenames : "**.*",  
    use          : "true",  
    windowtitle : "${title}",  
    doctitle     : "${title}",  
    header       : "${title}",  
    footer       : "${docFooter}",  
    overview     : "src/main/overview.html",  
    private      : "false") {  
  
    link(packages: "java.,org.xml.,javax.,org.xml.",  
        href: "http://download.oracle.com/javase/6/  
docs/api")  
  
    link(packages: "groovy.,org.codehaus.groovy.",  
        href: "http://groovy.codehaus.org/api")  
  
    link(packages: "org.apache.tools.ant.",  
        href: "http://evgeny-goldin.org/javadoc/ant/a  
pi")
```

```
link(packages:"org.junit.,junit.framework.",  
href:"http://kentbeck.github.com/junit/javad  
oc/latest")
```

```
link(packages:"org.codehaus.gmaven.",
```

```
href:"http://evgeny-goldin.org/javadoc/gmave  
n" )  
}
```

## Using Ant from Groovy

If ever you've been working with a build.xml file or some Jelly script and found yourself a little restricted by all those pointy brackets, or found it a bit wierd using XML as a scripting language and wanted something a little cleaner and more straight forward, then maybe Ant scripting with Groovy might be what you're after.

Groovy has a helper class called `AntBuilder` which makes the scripting of Ant tasks really easy; allowing a real scripting language to be used for programming constructs (variables, methods, loops, logical branching, classes etc). It still looks like a neat concise version of Ant's XML without all those pointy brackets; though you can mix and match this markup inside your script. Ant itself is a collection of jar files. By adding them to your classpath, you can easily use them within Groovy as is. We believe using `AntBuilder` leads to more concise and readily understood syntax.

Below are some examples (most taken from Groovy's own `AntBuilder` tests) which demonstrate:

- the use of Ant inside Groovy using the `AntBuilder` DSL notation
- a demo of iterating through an Ant `FileSet` using `fileScanner`
- that normal variables can be used to pass state into the Ant tasks and that Groovy code can be embedded anywhere in the markup.

```
def ant = new AntBuilder()

// lets just call one task
ant.echo("hello")

// here is an example of a block of Ant
inside GroovyMarkup
ant.sequential {
    echo("inside sequential")
    myDir = "target/AntTest/"
    mkdir(dir:myDir)
    copy(todir:myDir) {
        fileset(dir:"src/test") {
            include(name:"**/*.groovy")
        }
    }
    echo("done")
}

// now lets do some normal Groovy again
file = new
File("target/AntTest/groovy/util/AntTest.groovy")
assert file.exists()
```

```
def ant = new AntBuilder()

// lets create a scanner of filesets
scanner = ant.fileScanner {
    fileset(dir:"src/test") {
        include(name:"**/Ant*.groovy")
    }
}

// now lets iterate over
def found = false
for (f in scanner) {
    println("Found file $f")
    found = true
    assert f instanceof File
    assert f.name.endsWith(".groovy")
}
assert found
```

```
def ant = new AntBuilder()

ant.junit {

test(name: 'groovy.util.SomethingThatDoesNotExist')
}
}
```

```
def ant = new AntBuilder()

value = ant.path {
    fileset(dir:"xdocs") {
        include(name:"*.wiki")
    }
}

assert value != null

println "Found path of type
${value.class.name}"
println value
```

```
def ant = new AntBuilder()
def taskContainer = ant.parallel(){ //
"Parallel" serves as a sample TaskContainer
    ant.echo() //
"Echo" without message to keep tests silent
}
// not very elegant, but the easiest way to
get the ant internals...
assert taskContainer.dump() =~
/nestedTasks=\[org.apache.tools.ant.taskdefs
.Echo@\w+\]/
```

Compiling and running a Java file:

```
def ant = new AntBuilder()
ant.echo(file:'Temp.java', '''
class Temp { public static void
main(String[] args) {
System.out.println("Hello"); }}
''')
ant.javac(srcdir:'.', includes:'Temp.java',
fork:'true')
ant.java(classpath:'.', classname:'Temp',
fork:'true')
ant.echo('Done')
// =>
//      [javac] Compiling 1 source file
//      [java] Hello
//      [echo] Done
```

Sniffing around ...

```

def ant = new AntBuilder()
SpooftaskContainer.spoof.length = 0
def PATH = 'task.path'
ant.path(id:PATH){ant.pathelement(location:'
classes' )}
['spooftaskcontainer':'SpooftaskContainer',
'spoof':'Spooftask'].each{ pair ->
    ant.taskdef(name:pair.key,
classname:'groovy.util.'+pair.value,
classpathref:PATH)
}
ant.spooftaskcontainer(){
    ant.spoof()
}
expectedSpooftask =
    "SpooftaskContainer ctor\n"+
    "Spooftask ctor\n"+
    "in addTask\n"+
    "begin SpooftaskContainer execute\n"+
    "begin Spooftask execute\n"+
    "end Spooftask execute\n"+
    "end SpooftaskContainer execute\n"
assertEquals expectedSpooftask,
SpooftaskContainer.spoof.toString()

```

#### *Using the joint compiler*

Here is a small build file which uses the joint compiler to compile Groovy and Java source files together, and put them in WEB-INF/classes:

```

def ant = new AntBuilder().sequential {
  webinf = "deploy/WEB-INF"
  taskdef name: "groovyc", classname:
"org.codehaus.groovy.ant.Groovyc"
  groovyc srcdir: "src", destdir:
"${webinf}/classes", {
  classpath {
    fileset dir: "${webinf}/lib", {
      include name: "*.jar"
    }
    pathelement path: "${webinf}/classes"
  }
  javac source: "1.5", target: "1.5", debug:
"on"
}
}

```

## Using Ant Libraries with AntBuilder

### Introduction

Recent version of Ant have included a mechanism called [Antlibs](#). These allow you to define your own custom tasks, group them together with the appropriate definitions needed by Ant and use them in your Ant environment without nameclashes. Nameclashes are avoided by using namespaces. Numerous Antlibs are now available from both Apache (the developers of Ant) and other sources. Using these libraries with Groovy is fairly easy - though you have to be careful with some of the details.

### AntUnit

The [AntUnit](#) antlib includes predefined <assert> tasks corresponding to the most common kind of checks you want to do within your build files. They are using throughout the Ant codebase to test many of the ant tasks but you can use these assertions in your own build files (or any Groovy code) too.

Here is an example the uses the `assertFileDoesntExist` and `assertFileExists` checks.

First, we'll consider the traditional way of incorporating this antlib, by using namespaces (you'll need the `antunit.jar` in your classpath before you begin - as we are relying on Ant's autodiscovery of antlibs mechanism here):

```
def ant = new AntBuilder()

ant.'antlib:org.apache.ant.antunit:assertFileDoesntExist'(file:'copytest1.tmp')
ant.copy(file:'src/antunit.groovy',
tofile:'copytest1.tmp')
ant.'antlib:org.apache.ant.antunit:assertFileExists'(file:'copytest1.tmp')
ant.delete(file:'copytest1.tmp')
ant.'antlib:org.apache.ant.antunit:assertFileDoesntExist'(file:'copytest1.tmp')
```

Notice that the antunit assertions all exist within their own namespace. That's OK for now, Groovy allows special symbols in method names so long as you include the method name in quotes.

We can also incorporate the antlib directly into the default namespace as follows:

```
import org.apache.tools.ant.taskdefs.Antlib
def ant = new AntBuilder()
def url =
this.class.getResource('org/apache/ant/antunit/antlib.xml')
Antlib.createAntlib(ant.antProject, url,
'antlib:org.apache.ant.antunit').execute()

ant.assertFileDoesntExist(file:'copytest1.tmp')
ant.copy(file:'src/antunit.groovy',
tofile:'copytest1.tmp')
ant.assertFileExists(file:'copytest1.tmp')
ant.delete(file:'copytest1.tmp')
ant.assertFileDoesntExist(file:'copytest1.tmp')
```

This makes our code look simpler for this example but be careful with this approach though as you need to avoid name clashes. The preferred way is to use the NamespaceBuilder. Using this, our code becomes:

```
import groovy.xml.NamespaceBuilder
def ant = new AntBuilder()
def antunit =
NamespaceBuilder.newInstance(ant,
'antlib:org.apache.ant.antunit')
def destfile = 'copytest1.tmp'

antunit.assertFileDoesntExist(file:destfile)
    ant.copy(file:'src/antunit.groovy',
tofile:destfile)
antunit.assertFileExists(file:destfile)
    ant.delete(file:destfile)
antunit.assertFileDoesntExist(file:destfile)
```

#### Maven Ant Tasks

Another useful antlib is the [Maven Ant Tasks](#). They allow you to use Maven's artifact handling features from within Ant including:

- Dependency management - including transitive dependencies, scope recognition and SNAPSHOT handling
- Artifact deployment - file and SSH based deployment to a Maven repository
- POM processing - for reading a Maven 2.0.x pom.xml file

Here is how you could use these tasks to download some required jars into your local maven repository cache (~/.m2 directory).

```

import groovy.xml.NamespaceBuilder
def ant = new AntBuilder()

items = [[groupId:'jfree',
artifactId:'jfreechart', version:'1.0.5'],
         [groupId:'jfree',
artifactId:'jcommon', version:'1.0.9']]

def mvn = NamespaceBuilder.newInstance(ant,
'antlib:org.apache.maven.artifact.ant')

// download artifacts
mvn.dependencies(filesetId:'artifacts') {
items.each { dependency(it) } }
// print out what we downloaded
ant.fileScanner { fileset(refid:'artifacts')
}.each { println it }

```

When run, this produces a log of the maven ant task activity, such as:

```

Downloading:
jfree/jfreechart/1.0.5/jfreechart-1.0.5.pom
...
Transferring 298K
C:\Users\Paul\.m2\repository\jfree\jcommon\1
.0.9\jcommon-1.0.9.jar
C:\Users\Paul\.m2\repository\jfree\jfreechar
t\1.0.5\jfreechart-1.0.5.jar

```

We can take this example further and show how to create the JFreeChart example from [Plotting graphs with](#)

[JFreeChart](#) without having the JFreeChart jars statically defined in our classpath.

First another helper class:

```
class MavenDependency {  
    static void require(params) {  
  
    MavenDependencyHelper.getInstance().require(  
params)  
        }  
        static MavenDependencyHelper  
using(classLoader) {  
  
    MavenDependencyHelper.getInstance(classLoade  
r)  
        }  
    }  
  
    private class MavenDependencyHelper {  
        private classLoader  
        private  
MavenDependencyHelper(classLoader) {  
            this.classLoader = classLoader  
        }  
  
        static MavenDependencyHelper  
getInstance(classLoader) {  
            return new  
MavenDependencyHelper(classLoader)  
        }  
  
        static MavenDependencyHelper
```

```
getInstance() {
    return new
MavenDependencyHelper(MavenDependencyHelper.
classLoader)
}

MavenDependencyHelper require(params) {
    def ant = new AntBuilder()
    def mvn =
groovy.xml.NamespaceBuilder.newInstance(ant,
'antlib:org.apache.maven.artifact.ant')

mvn.dependencies(filesetId:"artifact_${param
s.groupId}_${params.artifactId}_${params.ver
sion}") { dependency(params) }
    ant.fileScanner {
fileset(refid:"artifact_${params.groupId}_${
params.artifactId}_${params.version}")
}.each {

classLoader.addClasspath(it.toString())
}
```

```
        this
    }
}
```

Now, here is the code we require to dynamically download the JFreeChart jars and add them to our classpath then run the script:

```
// no jfreechart imports required (we'll
find them programmatically)
import groovy.swing.SwingBuilder
import static
javax.swing.WindowConstants.EXIT_ON_CLOSE

def classLoader =
Thread.currentThread().contextClassLoader

// load jars and add to classpath
def maven =
MavenDependency.using(classLoader)
maven.require(groupId: 'jfree',
artifactId: 'jfreechart', version: '1.0.5')
maven.require(groupId: 'jfree',
artifactId: 'jcommon', version: '1.0.9')

// define used classes/instances
programmatically
def factoryClass =
classLoader.loadClass('org.jfree.chart.Chart
Factory')
def orientationClass =
classLoader.loadClass('org.jfree.chart.plot.
```

```
PlotOrientation')
def dataset =
classLoader.loadClass('org.jfree.data.category.DefaultCategoryDataset').newInstance()

// normal code below here
dataset.addValue 150, "no.1", "Jan"
dataset.addValue 210, "no.1", "Feb"
dataset.addValue 390, "no.1", "Mar"
dataset.addValue 300, "no.2", "Jan"
dataset.addValue 400, "no.2", "Feb"
dataset.addValue 200, "no.2", "Mar"

def labels = [ "Bugs", "Month", "Count" ]
def options = [true, true, true]
def chart =
factoryClass.createLineChart(*labels,
dataset,
orientationClass.VERTICAL, *options)
def swing = new SwingBuilder()
def frame = swing.frame(title:'Groovy
LineChart',
defaultCloseOperation:EXIT_ON_CLOSE)
{
panel(id:'canvas') {
rigidArea(width:400, height:400) }
}
frame.pack()
```

```
frame.show()  
chart.draw(swing.canvas.graphics,  
swing.canvas.bounds)
```

#### Ivy Tasks

We can also download jars using [Ivy](#). In this case we use MarkupBuilder to build an XML file that the Ivy retrieve task will use:

```

import groovy.xml.NamespaceBuilder

def ant = new AntBuilder()
def ivyfile = 'ivy.xml' // default file used
by Ivy
ant.delete(file:ivyfile, quiet:true)

new File(ivyfile).withWriter { writer ->
    def builder = new
groovy.xml.MarkupBuilder(writer)
    builder.'ivy-module'(version:'1.0') {
        info(organisation:"codehaus",
module:"GroovyExamples")
        dependencies {
            dependency(org:'jfree',
name:'jfreechart', rev:'1.0.5')
            dependency(org:'jfree',
name:'jcommon', rev:'1.0.9')
        }
    }
}

def ivy = NamespaceBuilder.newInstance(ant,
'antlib:org.apache.ivy.ant')
ivy.retrieve()
ivy.report(toDir:'reports') // optional

```

When run, this results in the files being downloaded:

```
[antlib:org.apache.ivy.ant:retrieve] :: Ivy
```



```
-----  
[antlib:org.apache.ivy.ant:retrieve] ::  
retrieving :: [ codehaus | grails ]  
[antlib:org.apache.ivy.ant:retrieve]  confs:  
[default]  
[antlib:org.apache.ivy.ant:retrieve]  2  
artifacts copied, 0 already retrieved
```

If you included the optional `report` step (and add `ant-trax.jar` from your ant distribution to our classpath), then you would have some additional log information and it would produce the following pretty report on dependencies:



default

### Dependencies Stats

Modules	2
Revisions	2 (0 searched  , 0 downloaded  , 0 evicted  , 0 errors  )
Artifacts	2 (0 downloaded, 0 failed)
Artifacts size	1456 kB (0 kB downloaded, 1456 kB in cache)

### Dependencies Overview

Module	Revision	Status	Resolver	Default	Licenses	Size
jfreechart by jfree	1.0.5	Integration	public	false		1157 kB
--- jcommon by jfree	1.0.9	Integration	public	false		299 kB
jcommon by jfree	1.0.9	Integration	public	false		299 kB

### Details

#### jfreechart by jfree resolved by default

Revision: 1.0.5

Status	Integration
Publication	20070607031318
Resolver	public
Configurations	system, default, compile, provided, runtime, master
Artifacts size	1157 kB (0 kB downloaded, 1157 kB in cache)

#### Required by

Organisation	Name	In Configurations	Asked Revision
codehaus	GroovyExamples	default	1.0.5

#### Dependencies

Module	Revision	Status	Resolver	Default	Licenses	Size
jcommon by jfree	1.0.9	Integration	public	false		299 kB

#### Artifacts

Name	Type	Ext	Download	Size
jfreechart	jar	jar	no	1157 kB

#### jcommon by jfree resolved by default

Revision: 1.0.9

Status	Integration
Publication	20070607031314
Resolver	public
Configurations	system, default, compile, provided, runtime, master
Artifacts size	299 kB (0 kB downloaded, 299 kB in cache)

#### Required by

Organisation	Name	In Configurations	Asked Revision
jfree	jfreechart	compile, runtime	1.0.9
codehaus	GroovyExamples	default	1.0.9

#### Dependencies

No dependency

#### Artifacts

Name	Type	Ext	Download	Size
jcommon	jar	jar	no	299 kB

## Bean Scripting Framework

Groovy integrates cleanly with [BSF](#) (the Bean Scripting Framework) which allows you to embed any scripting engine into your Java code while keeping your Java code decoupled from any particular scripting engine specifics.

The BSF engine for Groovy is implemented by the [GroovyEngine](#) class; however, that fact is normally hidden away by the BSF APIs. You just treat Groovy like any of the other scripting languages via the BSF API.

Note: Groovy has its own native support for integration with Java. See [Embedding Groovy](#) for further details. So you only need to worry about BSF if you want to also be able to call other languages from Java, e.g. [JRuby](#) or if you want to remain very loosely coupled from your scripting language.

## Getting started

Provided you have Groovy and BSF jars in your classpath, you can use the following Java code to run a sample Groovy script:

```
String myScript = "println('Hello World')\nreturn [1, 2, 3]";
BSFManager manager = new BSFManager();
List answer = (List) manager.eval("groovy",
    "myScript.groovy", 0, 0, myScript);
assertEquals(3, answer.size());
```

### Passing in variables

BSF lets you pass beans between Java and your scripting language. You can *register/unregister* beans which makes them known to BSF. You can then use BSF methods to *lookup* beans as required. Alternatively, you can *declare/undeclare* beans. This will register them but also make them available for use directly in your scripting language. This second approach is the normal approach used with Groovy. Here is an example:

```
manager.declareBean("xyz", new Integer(4),
    Integer.class);
Object answer = manager.eval("groovy",
    "test.groovy", 0, 0, "xyz + 1");
assertEquals(new Integer(5), answer);
```

### Other calling options

The previous examples used the *eval* method. BSF makes multiple methods available for your use (see the [BSF documentation](#) for more details). One of the other available methods is *apply*. It allows you to define an anonymous function in your scripting language and apply that function to arguments. Groovy supports this function using closures. Here is an example:

```
Vector ignoreParamNames = null;
Vector args = new Vector();
args.add(new Integer(2));
args.add(new Integer(5));
args.add(new Integer(1));
Integer actual = (Integer)
manager.apply("groovy", "applyTest", 0, 0,
    "def summer = { a, b, c -> a * 100 + b *
10 + c }", ignoreParamNames, args);
assertEquals(251, actual.intValue());
```

#### Access to the scripting engine

Although you don't normally need it, BSF does provide a hook that lets you get directly to the scripting engine. One of the functions which the engine can perform is to invoke a single method call on an object. Here is an example:

```
BSFEngine bsfEngine =
manager.loadScriptingEngine("groovy");
manager.declareBean("myvar", "hello",
String.class);
Object myvar = manager.lookupBean("myvar");
String result = (String)
bsfEngine.call(myvar, "reverse", new
Object[]{});
assertEquals("olleh", result);
```

#### Legacy points of interest

If you must integrate with early version of BSF (i.e. prior to **bsf 2.3.0-rc2**) then you'll need to manually register the Groovy language with BSF using the following snippet of code:

```
BSFManager.registerScriptingEngine(  
    "groovy",  
    "org.codehaus.groovy.bsf.GroovyEngine",  
    new String[] { "groovy", "gy" }  
);
```

## Compile-time Metaprogramming - AST Transformations

### AST Transformations

Although at times, it may sound like a good idea to extend the syntax of Groovy to implement new features (like this is the case for instance for multiple assignments), most of the time, we can't just add a new keyword to the grammar, or create some new syntax construct to represent a new concept. However, with the idea of AST (Abstract Syntax Tree) Transformations, we are able to tackle new and innovative ideas without necessary grammar changes.

When the Groovy compiler compiles Groovy scripts and classes, at some point in the process, the source code will end up being represented in memory in the form of a Concrete Syntax Tree, then transformed into an Abstract Syntax Tree. The purpose of AST Transformations is to let developers hook into the compilation process to be able to modify the AST before it is turned into bytecode that will be run by the JVM.

**AST Transformations provides Groovy with improved compile-time metaprogramming capabilities** allowing powerful flexibility at the language level, without a runtime performance penalty.

### Annotation based AST Transformations

One hook for accessing this capability is via annotations. In your Groovy code you can make use of one of more annotations to mark a class for receiving an AST transformation during compilation. Behind the scenes, an AST processor relevant to the annotation you are using is inserted into the compiler phases at the appropriate point.

There are a number of standard Annotations that cause AST transformations to be performed.

#### *Standard Annotations*

- [Bindable and Vetoable transformation](#)
- [Building AST Guide](#)
- [Category and Mixin transformations](#)
- [Compiler Phase Guide](#)
- [Delegate transformation](#)
- [Immutable AST Macro](#)
- [Immutable transformation](#)
- [Lazy transformation](#)
- [Newify transformation](#)
- [PackageScope transformation](#)
- [Singleton transformation](#)

[Grape](#) also provides its own transformation with `@Grab`.

## Implementing your own AST Transformations

There are two kinds of transformations: global and local transformations.

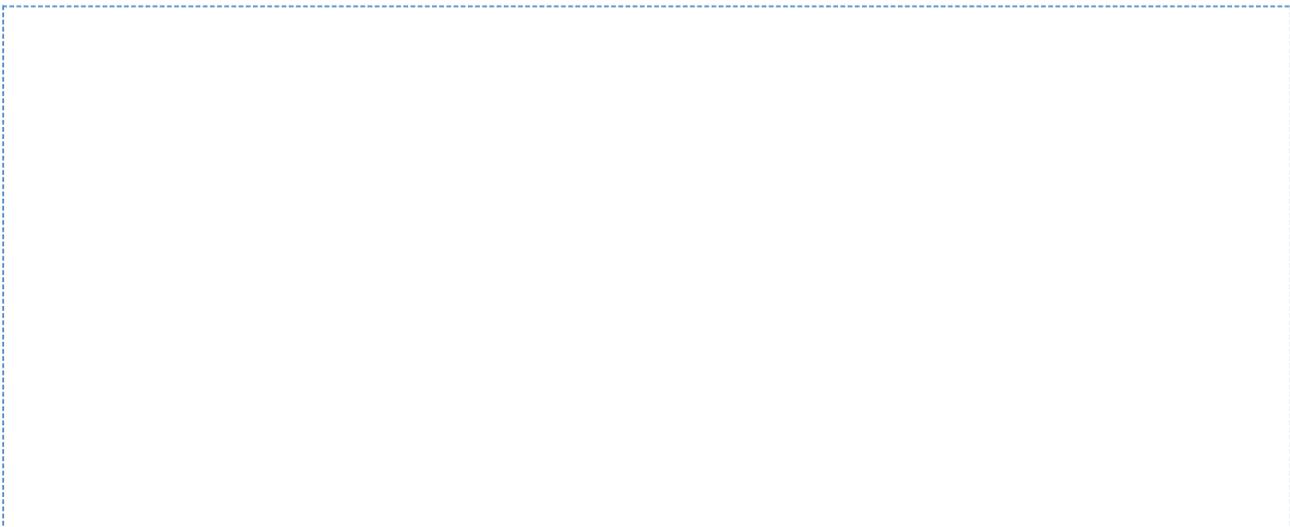
- Global transformations are applied to by the compiler on the code being compiled, wherever the transformation apply. Compiled classes that implement global tranasformations are in a JAR added to the classpath of the compiler and contain service locator file META-INF/services/org.codehaus.groovy.transform.ASTTransformation with a line with the name of the transformation class. The transformation class must have a no-args constructor and implement the org.codehaus.groovy.transform.ASTTransformationinterface. It will be run against every source in the compilation, so be sure to not create transformations which scan all the AST in an expansive and time-consuming manner, to keep the compiler fast.
- Local transformations are transformations applied locally by annotating code elements you want to transform. For this, we reuse the annotation notation, and those annotations should implement org.codehaus.groovy.transform.ASTTransformation. The compiler will discover them and apply the transformation on these code elements.

### **See Also**

- [Building AST Guide](#) - decide how best to create AST
- [Compiler Phase Guide](#) - decide on the compiler phase in which to perform the work
- implementing a [Global AST Transformations](#)
- implementing a [local AST transformation](#)

## Bindable and Vetoable transformation

To wrap up our overview of AST transformations, let's finish by speaking about two transformations very useful to Swing developers: @Bindable and @Vetoable. When creating Swing UIs, you're often interested in monitoring the changes of value of certain UI elements. For this purpose, the usual approach is to use JavaBeans PropertyChangeListeners to be notified when the value of a class field changes. You then end up writing this very common boiler-plate code in your Java beans:



```
import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;

public class MyBean {
    private String prop;

    PropertyChangeSupport pcs = new
PropertyChangeSupport(this);

    public void
addPropertyChangeListener(PropertyChangeList
ener l) {
        pcs.add(l);
    }

    public void
removePropertyChangeListener(PropertyChangeL
istener l) {
        pcs.remove(l);
    }

    public String getProp() {
        return prop;
    }

    public void setProp(String prop) {
        pcs.firePropertyChanged("prop",
this.prop, this.prop = prop);
    }
}
```

Fortunately, with Groovy and the `@Bindable` annotation, this code can be greatly simplified:

```
class MyBean {  
    @Bindable String prop  
}
```

Now pair that with Groovy's Swing builder new `bind()` method, define a text field and bind its value to a property of your data model:

```
textField text: bind(source: myBeanInstance,  
sourceProperty: 'prop')
```

Or even:

```
textField text: bind { myBeanInstance.prop }
```

The binding also works with simple expressions in the closure, for instance something like this is possible too:

```
bean location: bind { pos.x + ', ' + pos.y }
```

You may also be interested in having a look at `ObservableMap` and `ObservableList`, for a similar mechanism on maps and lists.

Along with `@Bindable`, there's also a `@Vetoable` transformation for when you need to be able to veto some property change. Let's consider a `Trompetist` class, where the performer's name is not allowed to contain the letter 'z':

```

import java.beans.*
import groovy.beans.Vetoable

class Trumpetist {
    @Vetoable String name
}

def me = new Trumpetist()
me.vetoableChange = { PropertyChangeEvent
pce ->
    if (pce.newValue.contains('z'))
        throw new PropertyVetoException("The
letter 'z' is not allowed in a name", pce)
}

me.name = "Louis Armstrong"

try {
    me.name = "Dizzy Gillespie"
    assert false: "You should not be able to
set a name with letter 'z' in it."
} catch (PropertyVetoException pve) {
    assert true
}

```

Looking at a more thorough Swing builder example with binding:

```

import groovy.swing.SwingBuilder
import groovy.beans.Bindable
import static
javax.swing.JFrame.EXIT_ON_CLOSE

class TextModel {
    @Bindable String text
}

def textModel = new TextModel()

SwingBuilder.build {
    frame( title: 'Binding Example
(Groovy)', size: [240,100], show: true,
        locationRelativeTo: null,
defaultCloseOperation: EXIT_ON_CLOSE ) {
        gridLayout cols: 1, rows: 2
        textField id: 'textField'
        bean textModel, text: bind{
textField.text }
        label text: bind{ textModel.text }
    }
}

```

Running this script shows up a frame with a text field and a label below, and the label's text is bound on the text field's content.

SwingBuilder has evolved so nicely in the past year that the Groovy Swing team decided to launch a new project based on it, and on the Grails foundations: project Griffon was born. Griffon proposes to bring the *Convention over Configuration* paradigm of Grails, as well as all its project structure, plugin system, gant scripting capabilities, etc.

If you are developing Swing rich clients, make sure to have a look at [Griffon](#).

**Building AST Guide**

## Building AST in Groovy 1.6 and Prior

In Groovy 1.6 (and prior) there is one way to build Abstract Syntax Trees (AST) in code: using the constructors on the ASTNode subclasses.

Here is an example of building a block of code that returns the String 'Hello'. A use case for this would be to create a method body implementation that simply returns 'Hello':

```
AstNode node = new BlockStatement(  
    [new ReturnStatement(  
        new ConstantExpression("Hello")  
    )],  
    new VariableScope())
```

### Advantages

- Documentation is available in [Javadoc/Groovydoc](#)
- Supports being invoked from Java
- Supported in all Groovy versions
- Some IDEs support code completion and parameter lookup

### Disadvantages

- It can be difficult to determine what AST you need to write
- Verbose - does not communicate the source being created
- Fragile - AST may need to change between major releases
- Author must know what AST looks like in a particular CompilePhase

## Building AST in Groovy 1.7

Groovy 1.7 introduces three new ways to build AST:

- From Strings
- From Code
- From a DSL-like Specification

### *AstBuilder.buildFromString*

The AstBuilder object provides an API to build AST from Strings of Groovy source code. The original example using buildFromString is:

```
List<ASTNode> nodes = new  
AstBuilder().buildFromString("\nHello\n")
```

### Advantages

- Does not require author to understand ASTNode subtypes

- Allows author to target a CompilePhase
- Communicates source code being generated
- Robust - Should need no changes even if AST is updated in a release

#### Disadvantages

- IDE cannot check syntax or grammar
- IDE cannot refactor across String
- Some entities cannot be created, like the AST for a field declaration

#### *AstBuilder.buildFromCode*

The AstBuilder object also provides an API to create AST from source code. The original example using buildFromCode is:

```
List<ASTNode> nodes = new  
AstBuilder().buildFromCode { "Hello" }
```

#### Advantages

- Clearly communicates source being generated
- Does not require author to understand ASTNode subtypes
- Allows author to target a CompilePhase
- Robust - Should need no changes even if AST is updated in a release
- IDE supports syntax checking and refactoring in Closure

#### Disadvantages

- Some entities cannot be created, like the AST for a field declaration
- buildFromCode requires that the left hand side of the invocation be of type AstBuilder. The best way to ensure this is to invoke it with:

```
new AstBuilder().buildFromCode { ... }
```

rather than having a local variable or field of type AstBuilder.

#### *AstBuilder.buildFromSpec*

The AstBuilder object also provides a DSL like API for building AST. The original example using buildFromSpec is:

```
List<ASTNode> nodes = new
AstBuilder().buildFromSpec {
    block {
        returnStatement {
            constant "Hello"
        }
    }
}
```

#### Advantages

- Allows conditionals (or any Groovy code) to be executed during the AST building process.
- Allows any ASTNode subtype to be created
- Fully documented with lengthy examples in [TestCase](#)

#### Disadvantages

- It can be difficult to determine what AST you need to write
- Verbose - does not always communicate the source being created
- Fragile - AST may need to change between major releases
- Author must know what AST looks like in a particular CompilePhase
- IDE does not *yet* provide code tips

#### Mixing Methods

Sometimes the best solution is to mix several types of the AST Builders. For instance, consider the following method:

```
public String myMethod(String parameter) {
    println 'Hello from a synthesized
method!'
    println "Parameter value: $parameter"
}
```

It might be best to use `buildFromSpec` to build the method declaration and `buildFromCode` to create the method body:

```

List<ASTNode> result = new
AstBuilder().buildFromSpec {
    method('myMethod', Opcodes.ACC_PUBLIC,
String) {
    parameters {
        parameter 'parameter':
String.class
    }
    exceptions {}
    block {
        owner.expression.addAll new
AstBuilder().buildFromCode {
            println 'Hello from a
synthesized method!'
            println "Parameter value:
$parameter"
        }
    }
    annotations {}
}
}
}

```

### Further Resources

The test cases shipping with Groovy are an excellent resource.

More examples can be found in GEP-2, the original proposal. <http://docs.codehaus.org/display/GroovyJSR/GEP+2+-+AST+Builder+Support>

Examples and questions can be found on the groovy-user and groovy-dev mailing lists.

### Category and Mixin transformations

If you've been using Groovy for a while, you're certainly familiar with the concept of Categories. It's a mechanism to extend existing types (even final classes from the JDK or third-party libraries), to add new methods to them. This is also a technique which can be used when writing Domain-Specific Languages. Let's consider the example below:

```

final class Distance {
    def number
    String toString() { "${number}m" }
}

class NumberCategory {
    static Distance getMeters(Number self) {
        new Distance(number: self)
    }
}

use(NumberCategory) {
    def dist = 300.meters

    assert dist instanceof Distance
    assert dist.toString() == "300m"
}

```

We have a simplistic and fictive Distance class which may have been provided by a third-party, who had the bad idea of making the class final so that nobody could ever extend it in any way. But thanks to a Groovy Category, we are able to decorate the Distance type with additional methods. Here, we're going to add a getMeters() method to numbers, by actually decorating the Number type. By adding a getter to a number, you're able to reference it using the nice property syntax of Groovy. So instead of writing 300.getMeters(), you're able to write 300.meters.

The downside of this category system and notation is that to add instance methods to other types, you have to create static methods, and furthermore, there's a first argument which represents the instance of the type we're working on. The other arguments are the normal arguments the method will take as parameters. So it may be a bit less intuitive than a normal method definition we would have added to Distance, should we have had access to its source code for enhancing it. Here comes the @Category annotation, which transforms a class with instance methods into a Groovy category:

```

@Category(Number)
class NumberCategory {
    Distance getMeters() {
        new Distance(number: this)
    }
}

```

No need for declaring the methods static, and the `this` you use here is actually the number on which the category will apply, it's not the real `this` of the category instance should we create one. Then to use the category, you can continue to use the `use(Category) {}` construct. What you'll notice however is that these kind of categories only apply to one single type at a time, unlike classical categories which can be applied to any number of types.

Now, pair `@Category` extensions to the `@Mixin` transformation, and you can mix in various behavior in a class, with an approach similar to multiple inheritance:

```

@Category(Vehicle) class FlyingAbility {
    def fly() { "I'm the ${name} and I fly!" }
}

@Category(Vehicle) class DivingAbility {
    def dive() { "I'm the ${name} and I dive!" }
}

interface Vehicle {
    String getName()
}

@Mixin(DivingAbility)
class Submarine implements Vehicle {
    String getName() { "Yellow Submarine" }
}

```

```
}
```

```
@Mixin(FlyingAbility)
```

```
class Plane implements Vehicle {  
    String getName() { "Concorde" }  
}
```

```
@Mixin([DivingAbility, FlyingAbility])
```

```
class JamesBondVehicle implements Vehicle {  
    String getName() { "James Bond's  
vehicle" }  
}
```

```
assert new Plane().fly() ==  
    "I'm the Concorde and I fly!"
```

```
assert new Submarine().dive() ==  
    "I'm the Yellow Submarine and I  
dive!"
```

```
assert new JamesBondVehicle().fly() ==  
    "I'm the James Bond's vehicle and I  
fly!"
```

```
assert new JamesBondVehicle().dive() ==  
       "I'm the James Bond's vehicle and I  
dive!"
```

You don't inherit from various interfaces and inject the same behavior in each subclass, instead you mixin the categories into your class. Here, our marvelous James Bond vehicle gets the flying and diving capabilities through mixins.

An important point to make here is that unlike `@Delegate` which can *inject* interfaces into the class in which the delegate is declared, `@Mixin` just does runtime mixing — as we shall see in the metaprogramming enhancements further down in this article.

## Compiler Phase Guide

Groovy AST transformations must be performed in one of the nine defined [compiler phases](#).

Global transformations may be applied in any phase, but local transformations may only be applied in the semantic analysis phase or later. Briefly, the compiler phases are:

- Initialization: source files are opened and environment configured
- Parsing: the grammar is used to produce tree of tokens representing the source code
- Conversion: An abstract syntax tree (AST) is created from token trees.
- Semantic Analysis: Performs consistency and validity checks that the grammar can't check for, and resolves classes.
- Canonicalization: Complete building the AST
- Instruction Selection: instruction set is chosen, for example java5 or pre java5
- Class Generation: creates the binary output in memory
- Output: write the binary output to the file system
- Finalization: Perform any last cleanup

Generally speaking, there is more type information available later in the phases. If your transformation is concerned with reading the AST, then a later phase where information is more plentiful might be a good choice. If your transformation is concerned with writing AST, then an earlier phase where the tree is more sparse might be more convenient.

### ***Static Property Transformations***

As a particular example, Groovy aggressively and statically types the static properties of a class, as demonstrated here:



```
class Foo {
  static doLog() { log.info("This won't even
compile!") }
}
Foo.metaClass.static.log = [info:{ println
it }] // Intended to enable the above code
to work (but doesn't work)
/*
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed,
/Users/robert/dev/workspace/Groovy-WithLog/test/Bogus.groovy: 2: Apparent variable 'log'
was found in a static scope but doesn't
refer to a local variable, static field or
class. Possible causes:
You attempted to reference a variable in the
binding or an instance variable from a
static context.
You misspelled a classname or statically
imported field. Please check the spelling.
You attempted to use a method 'log' but left
out brackets in a place not allowed by the
grammar.
@ line 2, column 19.
    static doLog() { log.info("This won't
even compile!") }
                        ^

1 error
*/
```

So if you want to create an AST transform that creates a static property, you have to hook in before this check is performed: in practice, this means the Conversion phase (Semantic Analysis is where you get busted). In using a phase that early, though, most types won't be resolved, and so insofar as type resolution is significant (e.g. in checking for annotations of a particular class), it will have to be hand-rolled.

### ***Further Information***

To learn more about what AST is produced in each phase, you can use the AST viewer in Groovy console to explore the output in different phases.

### **Delegate transformation**

Java doesn't provide any built-in delegation mechanism, and so far Groovy didn't either. But with the `@Delegate` transformation, a class field or property can be annotated and become an object to which method calls are delegated. In the following example, an Event class has a date delegate, and the compiler will delegate all of Date's methods invoked on the Event class to the Date delegate. As shown in the latest assert, the Event class has got a `before(Date)` method, and all of Date's methods.

```

import java.text.SimpleDateFormat

class Event {
    @Delegate Date when
    String title, url
}

def df = new SimpleDateFormat("yyyy/MM/dd")

def gr8conf = new Event(title: "GR8
Conference",
                        url:
"http://www.gr8conf.org",
                        when:
df.parse("2009/05/18"))
def javaOne = new Event(title: "JavaOne",
                        url:
"http://java.sun.com/javaone/",
                        when:
df.parse("2009/06/02"))

assert gr8conf.before(javaOne.when)

```

The Groovy compiler adds all of Date's methods to the Event class, and those methods simply delegate the call to the Date field. If the delegate is not a final class, it is even possible to make the Event class a subclass of Date simply by extending Date, as shown below. No need to implement the delegation ourselves by adding each and every Date methods to our Event class, since the compiler is friendly-enough with us to do the job itself.

```

class Event extends Date {
    @Delegate Date when
    String title, url
}

```

In the case you are delegating to an interface, however, you don't even need to explicitly say you implement the interface of the delegate. The @Delegate transformation will take care of this and implement that interface. So the instances of your class will automatically be instances of the delegate's interface. `import java.util.concurrent.locks.*`

```

class LockableList {
    @Delegate private List list = \[\]
    @Delegate private Lock lock = new
    ReentrantLock()
}

def list = new LockableList()

list.lock()
try {
    list << 'Groovy'
    list << 'Grails'
    list << 'Griffon'
} finally {
    list.unlock()
}

assert list.size() == 3
assert list instanceof Lock
assert list instanceof List

```

In this example, our LockableList is now a composite of a list and a lock and is instanceof of List and Lock. However, if you didn't intend your class to be implementing these interfaces, you would still be able to do so by specifying a parameter on the annotation: @Delegate(interfaces = false) private List list = []

Let's have a look at another simple usage of @Delegate, for wrapping an existing class, delegating all calls to the delegate:

```
class Photo {
    int width
    int height
}

class PhotoSelection {
    @Delegate Photo photo

    String title
    String caption
}

def photo = new Photo(width: 640, height:
480)
def selection = new PhotoSelection(title:
"Groovy", caption: "Groovy", photo: photo)

assert selection.title == "Groovy"
assert selection.caption == "Groovy"

assert selection.width == 640
assert selection.height == 480
```

## Immutable AST Macro

### *The @Immutable Annotation*

Immutable objects are ones which don't change after initial creation. Such objects are frequently desirable because they are simple and can be safely shared even in multi-threading contexts. This makes them great for functional and

concurrent scenarios. The rules for creating such objects are well-known:

- No mutators (methods that modify internal state)
- Class must be final
- Fields must be private and final
- Defensive copying of mutable components
- equals, hashCode and toString must be implemented in terms of the fields if you want to compare your objects or use them as keys in e.g. maps

Writing classes that follow these rules is not hard but does involve a fair bit of boiler plate code and is prone to error. Here is what such a class might look like in Java:

```
// Java  
public final class Punter {  
    private final String first;  
    private final String last;  
  
    public Punter(String first, String last)  
{  
        this.first = first;  
        this.last = last;  
}  
  
    public String getFirst() {  
        return first;  
}  
  
    public String getLast() {  
        return last;  
}  
  
    @Override  
    public int hashCode() {  
        final int prime = 31;  
        int result = 1;  
        result = prime * result + ((first ==
```

```

null)
        ? 0 : first.hashCode());
result = prime * result + ((last ==
null)
        ? 0 : last.hashCode());
return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Punter other = (Punter) obj;
    if (first == null) {
        if (other.first != null)
            return false;
    } else if
(!first.equals(other.first))
        return false;
    if (last == null) {
        if (other.last != null)
            return false;
    } else if (!last.equals(other.last))
        return false;
    return true;
}

```

```
@Override  
public String toString() {  
    return "Punter(first:" + first  
        + ", last:" + last + ")";  
}
```

```
    }  
  
}
```

Groovy makes it easier to create such classes using the `@Immutable` annotation. You only need this:

```
@Immutable final class Punter {  
    String first, last  
}
```

The "other code" shown above is added at compile time. All of the methods you see above will be there (and you can use them from Java of course). You just don't need to develop and maintain them.

#### The Details

A class created using `@Immutable` has the following characteristics:

- Properties automatically have private, final backing fields with getters.
- Attempts to update the property will result in a `ReadOnlyPropertyException`.
- A map-based constructor is provided which allows you to set properties by name.
- A tuple-style constructor is provided which allows you to set properties in the same order as they are defined.
- Default `equals`, `hashCode` and `toString` methods are provided based on the property values.
- Date objects, `Cloneable` objects and arrays are defensively copied on the way in (constructor) and out (getters).
- Arrays and cloneable objects use the `clone` method. For your own classes, it is up to you to define this method and use deep cloning if appropriate.
- `Collection` objects and `Map` objects are wrapped by immutable wrapper classes (but not deeply cloned!).
- Attempts to update them will result in an `UnsupportedOperationException`.
- Fields that are enums or other `@Immutable` classes are allowed but for an otherwise possible mutable property type, an error is thrown.
- You don't have to follow Groovy's normal property conventions, e.g. you can create an explicit private field and then you can write explicit get and set methods. Such an approach, isn't currently prohibited (to give you some wiggle room to get around these conventions) but any fields created in this way are deemed not to be part of the significant state of the object and aren't factored into the `equals` or `hashCode` methods. Use at your own risk!

### Immutable transformation

Immutable objects are ones which don't change after initial creation. Such objects are frequently desirable because they are simple and can be safely shared even in multi-threading contexts. This makes them great for functional and concurrent scenarios. The rules for creating such objects are well-known:

- No mutators (methods that modify internal state)
- Class must be final
- Fields must be private and final
- Defensive copying of mutable components

- equals(), hashCode() and toString() must be implemented in terms of the fields if you want to compare your objects or use them as keys in e.g. maps

Instead of writing a very long Java or Groovy class mimicking this immutability behavior, Groovy lets you just write an immutable class as follow:

```
@Immutable final class Coordinates {  
    Double latitude, longitude  
}  
  
def c1 = new Coordinates(latitude:  
48.824068, longitude: 2.531733)  
def c2 = new Coordinates(48.824068,  
2.531733)  
  
assert c1 == c2
```

All the boiler-plate code is generated at compile-time for you! The example shows that to instantiate such immutable coordinates, you can use one of the two constructors created by the transformation, one taking a map whose keys are the properties to set to the values associated with those keys, and the other taking the values of the properties as parameters. The assert also shows that equals() was implemented and allows us to properly compare such immutable objects.

You can have a look at the [details of the implementation](#) of this transformation. For the record, the Groovy example above using the @Immutable transformation is over 50 lines of equivalent Java code.

### **Lazy transformation**

Another transformation is @Lazy. Sometimes, you want to handle the initialization of a field of your class lazily, so that its value is computed only on first use, often because it may be time-consuming or memory-expensive to create. The usual approach is to customize the getter of said field, so that it takes care of the initialization when the getter is called the first time. But in Groovy 1.6, you can now use the @Lazy annotation for that purpose:

```

class Person {
    @Lazy pets = ['Cat', 'Dog', 'Bird']
}

def p = new Person()
assert !(p.dump().contains('Cat'))

assert p.pets.size() == 3
assert p.dump().contains('Cat')

```

In the case of complex computation for initializing the field, you may need to call some method for doing the work, instead of a value like our pets list. It is then possible to have the lazy evaluation being done by a closure call, as the following example shows:

```

class Person {
    @Lazy List pets = { /* complex
computation here */ }()
}

```

There is also an option for leveraging Soft references for garbage collection friendliness for expensive data structures that may be contained by such lazy fields:

```

class Person {
    @Lazy(soft = true) List pets = ['Cat',
'Dog', 'Bird']
}

def p = new Person()
assert p.pets.contains('Cat')

```

The internal field created by the compiler for pets will actually be a Soft reference, but accessing p.pets directly will

return the value (ie. the list of pets) held by that reference, making the use of the soft reference transparent to the user of that class.

### **Newify transformation**

The @Newify transformation proposes two new ways of instantiating classes. The first one is providing Ruby like approach to creating instances with a new() class method:

```
@Newify rubyLikeNew() {  
    assert Integer.new(42) == 42  
}  
  
rubyLikeNew()
```

But it is also possible to follow the Python approach with omitting the new keyword. Imagine the following tree creation:

```

class Tree {
    def elements
    Tree(Object... elements) { this.elements
= elements as List }
}

class Leaf {
    def value
    Leaf(value) { this.value = value }
}

def buildTree() {
    new Tree(new Tree(new Leaf(1), new
Leaf(2)), new Leaf(3))
}

buildTree()

```

The creation of the tree is not very readable because of all those new keywords spread across the line. The Ruby approach wouldn't be more readable, since a `new()` method call for creating each element is needed. But by using `@Newify`, we can improve our tree building slightly to make it easier on the eye:

```

@Newify([Tree, Leaf]) buildTree() {
    Tree(Tree(Leaf(1), Leaf(2)), Leaf(3))
}

```

You'll also notice that we just allowed `Tree` and `Leaf` to be *newified*. By default, under the scope which is annotated, all instantiations are *newified*, but you can limit the reach by specifying the classes you're interested in. Also, note that for our example, perhaps a Groovy builder may have been more appropriate, since its purpose is to indeed create any kind of hierarchical / tree structure.

If we take another look at our coordinates example from a few sections earlier, using both `@Immutable` and `@Newify` can be interesting for creating a path with a concise but type-safe manner:

```

@Immutable final class Coordinates {
    Double latitude, longitude
}

@Immutable final class Path {
    Coordinates[] coordinates
}

@Newify([Coordinates, Path])
def build() {
    Path(
        Coordinates(48.824068, 2.531733),
        Coordinates(48.857840, 2.347212),
        Coordinates(48.858429, 2.342622)
    )
}

assert build().coordinates.size() == 3

```

A closing remark here: since a `Path(Coordinates[] coordinates)` was generated, we can use that constructor in a *var args* way in Groovy, just as if it had been defined as `Path(Coordinates... coordinates)`.

### PackageScope transformation

Groovy's convention for properties is that any *field* without any visibility modifier is exposed as a property, with a getter and a setter transparently generated for you. For instance, this `Person` class exposes a getter `getName()` and a setter `setName()` for a private `name` field:

```

class Person {
    String name
}

```

Which is equivalent to this Java class:

```
public class Person {
    private String name;
    public String getName() { return name; }
    public void setName(name) { this.name =
name; }
}
```

That said, this approach has one drawback in that you don't have the possibility to define a field with package-scope visibility. To be able to expose a field with package-scope visibility, you can now annotate your field with the `@PackageScope` annotation.

### Singleton transformation

Whether the singleton is pattern or an anti-pattern, there are still some cases where we need to create singletons. We're used to create a private constructor, a `getInstance()` method for a static field or even an initialized public static final field. So instead of writing code like this in Java:

```
public class T {
    public static final T instance = new
T();
    private T() {}
}
```

You just need to annotate your type with the `@Singleton` annotation:

```
@Singleton class T {}
```

The singleton instance can then simply be accessed with `T.instance` (direct public field access).

You can also have the lazy loading approach with an additional annotation parameter:

```
@Singleton(lazy = true) class T {}
```

Would become more or less equivalent to this Groovy class:

```

class T {
    private static volatile T instance
    private T() {}
    static T getInstance () {
        if (instance) {
            instance
        } else {
            synchronized(T) {
                if (instance) {
                    instance
                } else {
                    instance = new T ()
                }
            }
        }
    }
}

```

Lazy or not, once again, to access the instance, simply do T.instance (property access, shortcut for T.getInstance()).

## GPath

GPath is a path expression language integrated into Groovy which allows parts of nested structured data to be identified. In this sense, it has similar aims and scope as XPath does for XML. The two main places where you use GPath expressions is when dealing with nested POJOs or when dealing with XML.

As an example, you can specify a path to an object or element of interest:

a.b.c -> for XML, yields all the <c> elements inside <b> inside <a>

a.b.c -> all POJOs, yields the <c> properties for all the <b> properties of <a> (sort of like a.getB().getC() in JavaBeans)

For XML, you can also specify attributes, e.g.:

a["@href"] -> the href attribute of all the a elements

a.'@href' -> an alternative way of expressing this

a.@href -> an alternative way of expressing this when using XmlSlurper

### Example

The best example of GPath for xml is test-new/groovy/util/XmlSlurperTest.groovy.

```
package groovy.util

class XmlSlurperTest extends GroovyTestCase
{

void testXmlParser() {
    def text = """
<characters>
  <props>
    <prop>dd</prop>
  </props>
  <character id="1" name="Wallace">
    <likes>cheese</likes>
  </character>
  <character id="2" name="Gromit">
    <likes>sleep</likes>
  </character>
</characters>
"""

    def node = new
    XmlSlurper().parseText(text);

    assert node != null
    assert node.children().size() == 3 //,
    "Children ${node.children()}"

    def characters = node.character
    println "node:" + node.children().size()
    println "characters:" +
```

```
node.character.size()
  for (c in characters) {
    println c['@name']
  }

assert characters.size() == 2

assert node.character.likes.size() == 2
//, "Likes ${node.character.likes}"

// lets find Gromit
def gromit = node.character.find {
it['@id'] == '2' }
  assert gromit != null //, "Should have
found Gromit!"
  assert gromit['@name'] == "Gromit"

// lets find what Wallace likes in 1
query
  def answer = node.character.find {
it['@id'] == '1' }.likes.text()
```

```
    assert answer == "cheese"  
  }  
}
```

## Outline

### 1. Accessing element as property

```
def characters = node.character  
def gromit = node.character[1]
```

### 2. Accessing attributes

```
println gromit['@name']  
or  
println gromit.@name
```

### 3. Accessing element body

```
println gromit.likes[0].text()  
println node.text()
```

If the element is a father node, it will print all children's text.

### 3. Explore the DOM use children() and parent()

```
def characters = node.children()  
for (c in characters) {  
  println c.@name  
}
```

### 4. Find elements use expression

```
def gromit = node.character.find { it.@id ==  
'2' }
```

#### *Another Example*

Here is a two line example of how to get a list of all the links to .xml files listed on a web page. The Neko parser is used to parse non-well formed html. It no longer ships as part of the standard Groovy distribution but can be [downloaded](#) and dropped into the lib directory of your Groovy distribution. You'll need to also add a copy of xercesImpl.jar to the groovy lib directory.

```
def myDocument = new XmlParser( new  
org.cyberneko.html.parsers.SAXParser()  
) .parse("http://myUrl.com")  
def links =  
myDocument.depthFirst().A[ '@href' ].findAll{  
it.endsWith(".xml") }
```

#### *More Information*

See also: [Processing XML](#)

## **Groovy and JMX**

### **Introduction**

Given that Groovy sits directly on top of Java, Groovy can leverage the tremendous amount of work already done for [JMX](#) with Java. In addition, Groovy provides a `GroovyMBean` class which makes an *MBean* look like a normal Groovy object. This simplifies Groovy code for interacting with MBeans. For example, the following code:

```
println server.getAttribute(beanName, 'Age')
server.setAttribute(beanName, new
Attribute('Name', 'New name'))
Object[] params = [5, 20]
String[] signature = [Integer.TYPE,
Integer.TYPE]
println server.invoke(beanName, 'add',
params, signature)
```

can be simplified to:

```
def mbean = new GroovyMBean(server,
beanName)
println mbean.Age
mbean.Name = 'New name'
println mbean.add(5, 20)
```

The remainder of this page shows you how to:

- Monitor the JVM using MXBeans
- Monitor Apache Tomcat and display statistics
- Monitor Oracle OC4J and display information
- Monitor BEA WebLogic and display information
- Leverage Spring's MBean annotation support to export your Groovy beans as MBeans

Note: many of the examples on this page use Java 5 which incorporates JMX 1.2 (more recent versions if JMX or Java will also work). In some cases, you can run some of these examples using Java 1.4 by including a version of JMX on your CLASSPATH. MX4J is bundled with the full distribution of Groovy. In most cases, you can delete this jar from your distribution `lib` directory if you are running with Java 5 or above (in fact you might have to - see the Troubleshooting section below).

## Monitoring the JVM

MBeans are not accessed directly by an application but are managed by a repository called an *MBean server*. Java 5 and above includes a special MBean server called the *platform MBean server*, which is built into the JVM. Platform MBeans are registered in this server using unique names.

You can monitor the JVM through its platform MBeans with the following code:

```
import java.lang.management.*

def os =
ManagementFactory.operatingSystemMXBean
println ""OPERATING SYSTEM:
\tarchitecture = $os.arch
\tname = $os.name
\tversion = $os.version
\tprocessors = $os.availableProcessors
""

def rt = ManagementFactory.runtimeMXBean
println ""RUNTIME:
\tname = $rt.name
\tspec name = $rt.specName
\tvendor = $rt.specVendor
\tspec version = $rt.specVersion
\tmanagement spec version =
$rt.managementSpecVersion
""

def cl =
ManagementFactory.classLoadingMXBean
println ""CLASS LOADING SYSTEM:
\tisVerbose = ${cl.isVerbose()}
\tloadedClassCount = $cl.loadedClassCount
\ttotalLoadedClassCount =
$cl.totalLoadedClassCount
\tunloadedClassCount =
$cl.unloadedClassCount
```

```
"""
```

```
def comp =  
ManagementFactory.compilationMXBean  
println ""COMPILATION:  
\ttotalCompilationTime =  
$comp.totalCompilationTime  
"""
```

```
def mem = ManagementFactory.memoryMXBean  
def heapUsage = mem.heapMemoryUsage  
def nonHeapUsage = mem.nonHeapMemoryUsage  
println ""MEMORY:  
HEAP STORAGE:  
\tcommitted = $heapUsage.committed  
\tinit = $heapUsage.init  
\tmax = $heapUsage.max  
\tused = $heapUsage.used  
NON-HEAP STORAGE:  
\tcommitted = $nonHeapUsage.committed  
\tinit = $nonHeapUsage.init  
\tmax = $nonHeapUsage.max  
\tused = $nonHeapUsage.used  
"""
```

```
ManagementFactory.memoryPoolMXBeans.each{ mp  
->  
    println "\tname: " + mp.name  
    String[] mmnames = mp.memoryManagerNames  
    mmnames.each{ mmname ->  
        println "\t\tManager Name: $mmname"
```

```
    }
    println "\t\tmtype = $mp.type"
    println "\t\tUsage threshold supported =
" + mp.isUsageThresholdSupported()
  }
println()
```

```
def td = ManagementFactory.threadMXBean
println "THREADS:"
td.allThreadIds.each { tid ->
    println "\tThread name =
${td.getThreadInfo(tid).threadName}"
}
println()
```

```
println "GARBAGE COLLECTION:"
ManagementFactory.garbageCollectorMXBeans.ea
ch { gc ->
    println "\tname = $gc.name"
    println "\t\tcollection count =
$gc.collectionCount"
    println "\t\tcollection time =
$gc.collectionTime"
    String[] mpoolNames = gc.memoryPoolNames
    mpoolNames.each { mpoolName ->
        println "\t\tmpool name =
```

```
$mpoolName "  
    }  
}
```

When run, you will see something like this:

#### **OPERATING SYSTEM:**

```
architecture = x86  
name = Windows XP  
version = 5.1  
processors = 2
```

#### **RUNTIME:**

```
name = 620@LYREBIRD  
spec name = Java Virtual Machine  
Specification  
vendor = Sun Microsystems Inc.  
spec version = 1.0  
management spec version = 1.0
```

#### **CLASS LOADING SYSTEM:**

```
isVerbose = false  
loadedClassCount = 919  
totalLoadedClassCount = 919  
unloadedClassCount = 0
```

#### **COMPILATION:**

```
totalCompilationTime = 91
```

#### **MEMORY:**

**HEAP STORAGE:**

**committed = 3108864**

**init = 0**

**max = 66650112**

**used = 1994728**

**NON-HEAP STORAGE:**

**committed = 9240576**

**init = 8585216**

**max = 100663296**

**used = 5897880**

**name: Code Cache**

**Manager Name: CodeCacheManager**

**mtype = Non-heap memory**

**Usage threshold supported = true**

**name: Eden Space**

**Manager Name: MarkSweepCompact**

**Manager Name: Copy**

**mtype = Heap memory**

**Usage threshold supported = false**

**name: Survivor Space**

**Manager Name: MarkSweepCompact**

**Manager Name: Copy**

**mtype = Heap memory**

**Usage threshold supported = false**

**name: Tenured Gen**

**Manager Name: MarkSweepCompact**

**mtype = Heap memory**

**Usage threshold supported = true**

**name: Perm Gen**

**Manager Name: MarkSweepCompact**

**mtype = Non-heap memory**  
**Usage threshold supported = true**

**THREADS:**

**Thread name = Monitor Ctrl-Break**  
**Thread name = Signal Dispatcher**  
**Thread name = Finalizer**  
**Thread name = Reference Handler**  
**Thread name = main**

**GARBAGE COLLECTION:**

**name = Copy**  
**collection count = 60**  
**collection time = 141**  
**mpool name = Eden Space**  
**mpool name = Survivor Space**  
**name = MarkSweepCompact**  
**collection count = 0**  
**collection time = 0**  
**mpool name = Eden Space**

```
mpool name = Survivor Space  
mpool name = Tenured Gen  
mpool name = Perm Gen
```

## Monitoring Tomcat

First start up [Tomcat](#) with JMX monitoring enabled by setting the following:

```
set JAVA_OPTS=-Dcom.sun.management.jmxremote  
-Dcom.sun.management.jmxremote.port=9004  
-Dcom.sun.management.jmxremote.authenticate=  
false  
-Dcom.sun.management.jmxremote.ssl=false
```

You can do this in your startup script and may choose any available port - we used 9004.

The following code uses JMX to discover the available MBeans in the running Tomcat, determine which are web modules, extract the processing time for each web module and displays the result in a graph using JFreeChart:

```
import javax.management.ObjectName  
import  
javax.management.remote.JMXConnectorFactory  
as JmxFactory  
import javax.management.remote.JMXServiceURL  
as JmxUrl  
import org.jfree.chart.ChartFactory  
import  
org.jfree.data.category.DefaultCategoryDataset  
as Dataset  
import org.jfree.chart.plot.PlotOrientation  
as Orientation
```

```
import groovy.swing.SwingBuilder
import javax.swing.WindowConstants as WC

def serverUrl =
    'service:jmx:rmi:///jndi/rmi://localhost:9004/jmxrmi'
def server = JmxFactory.connect(new
    JmxUrl(serverUrl)).MBeanServerConnection
def serverInfo = new GroovyMBean(server,
    'Catalina:type=Server').serverInfo
println "Connected to: $serverInfo"

def query = new ObjectName('Catalina:*')
String[] allNames = server.queryNames(query,
    null)
def modules = allNames.findAll{ name ->
    name.contains('j2eeType=WebModule')
}.collect{ new GroovyMBean(server, it) }

println "Found ${modules.size()} web
modules. Processing ..."
def dataset = new Dataset()

modules.each{ m ->
    println m.name()
    dataset.addValue m.processingTime, 0,
m.path
}

def labels = ['Time per Module', 'Module',
    'Time']
```

```
def options = [false, true, true]
def chart =
ChartFactory.createBarChart(*labels,
dataset,
                Orientation.VERTICAL,
*options)
def swing = new SwingBuilder()
def frame = swing.frame(title:'Catalina
Module Processing Time',

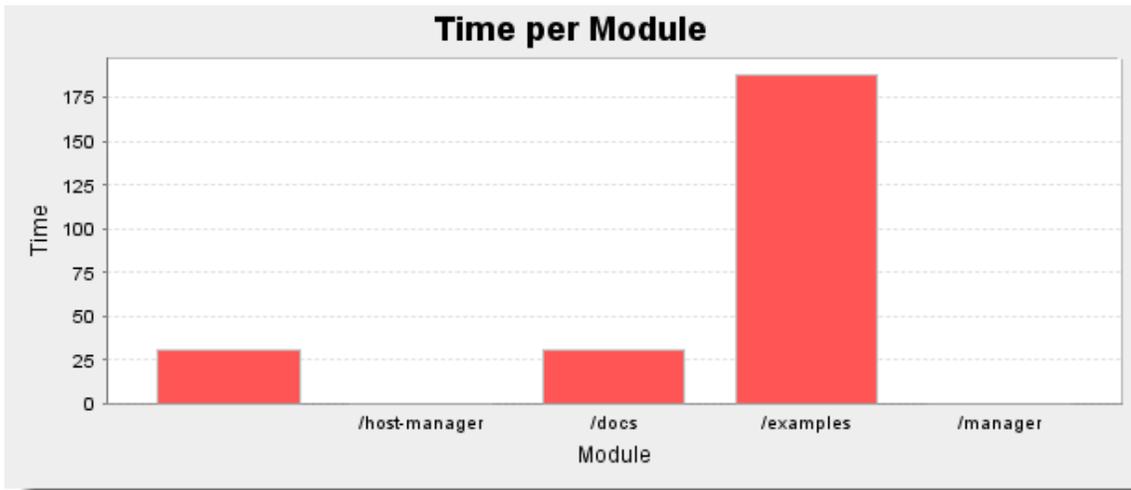
defaultCloseOperation:WC.EXIT_ON_CLOSE) {
    panel(id:'canvas') {
rigidArea(width:600, height:250) }
}
frame.pack()
```

```
frame.show()  
chart.draw(swing.canvas.graphics,  
swing.canvas.bounds)
```

When run, we will see a trace of progress being made:

```
Connected to: Apache Tomcat/6.0.13  
Found 5 web modules. Processing ...  
Catalina:j2eeType=WebModule,name=//localhost  
/,J2EEApplication=none,J2EEServer=none  
Catalina:j2eeType=WebModule,name=//localhost  
/host-manager,J2EEApplication=none,J2EEServe  
r=none  
Catalina:j2eeType=WebModule,name=//localhost  
/docs,J2EEApplication=none,J2EEServer=none  
Catalina:j2eeType=WebModule,name=//localhost  
/examples,J2EEApplication=none,J2EEServer=no  
ne  
Catalina:j2eeType=WebModule,name=//localhost  
/manager,J2EEApplication=none,J2EEServer=non  
e
```

The output will look like this:



Note: if you get errors running this script, see the **Troubleshooting** section below.

### OC4J Example

Here is a script to access [OC4J](#) and print out some information about the server, its runtime and (as an example) the configured JMS destinations:

```
import javax.management.remote.*
import
oracle.oc4j.admin.jmx.remote.api.JMXConnecto
rConstant

def serverUrl = new
JMXServiceURL('service:jmx:rmi://localhost:2
3791')
def serverPath =
'oc4j:j2eeType=J2EEServer,name=standalone'
def jvmPath =
'oc4j:j2eeType=JVM,name=single,J2EEServer=st
andalone'
def provider =
'oracle.oc4j.admin.jmx.remote'
def credentials = [
```

```

(JMXConnectorConstant.CREDENTIALS_LOGIN_KEY)
: 'oc4jadmin',

(JMXConnectorConstant.CREDENTIALS_PASSWORD_KEY): 'admin'
]
def env = [

(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES): provider,
    (JMXConnector.CREDENTIALS): credentials
]
def server = JmxFactory.connect(serverUrl,
env).MBeanServerConnection
def serverInfo = new GroovyMBean(server,
serverPath)
def jvmInfo = new GroovyMBean(server,
jvmPath)
println """Connected to $serverInfo.node. \
Server started ${new
Date(serverInfo.startTime)}.
OC4J version:  $serverInfo.serverVersion
from $serverInfo.serverVendor
JVM version:  $jvmInfo.javaVersion from
$jvmInfo.javaVendor
Memory usage:  $jvmInfo.freeMemory bytes
free, \
$jvmInfo.totalMemory bytes total
""

def query = new

```

```
javax.management.ObjectName('oc4j:*')
String[] allNames = server.queryNames(query,
null)
def dests = allNames.findAll{ name ->

name.contains('j2eeType=JMSDestinationResour
ce')}
}.collect{ new GroovyMBean(server, it) }

println "Found ${dests.size()} JMS
```

```
destinations. Listing ..."  
dests.each{ d -> println "$d.name:  
$d.location" }
```

Here is the result of running this script:

```
Connected to LYREBIRD. Server started Thu  
May 31 21:04:54 EST 2007.  
OC4J version: 11.1.1.0.0 from Oracle Corp.  
JVM version: 1.6.0_01 from Sun  
Microsystems Inc.  
Memory usage: 8709976 bytes free, 25153536  
bytes total
```

```
Found 5 JMS destinations. Listing ...  
Demo Queue: jms/demoQueue  
Demo Topic: jms/demoTopic  
jms/Oc4jJmsExceptionQueue:  
jms/Oc4jJmsExceptionQueue  
jms/RAExceptionQueue: jms/RAExceptionQueue  
OracleASRouter_store: OracleASRouter_store
```

As a slight variation, this script displays a pie chart of memory usage using JFreeChart:

```
import org.jfree.chart.ChartFactory  
import javax.swing.WindowConstants as WC  
import javax.management.remote.*  
import  
oracle.oc4j.admin.jmx.remote.api.JMXConnecto  
rConstant
```

```
def url =
  'service:jmx:rmi://localhost:23791'
def credentials = [:]
credentials[JMXConnectorConstant.CREDENTIALS
_LOGIN_KEY] = "oc4jadmin"
credentials[JMXConnectorConstant.CREDENTIALS
_PASSWORD_KEY] = "password"
def env = [:]
env[JMXConnectorFactory.PROTOCOL_PROVIDER_PA
CKAGES] = "oracle.oc4j.admin.jmx.remote"
env[JMXConnector.CREDENTIALS] = credentials
def server = JMXConnectorFactory.connect(new
JMXServiceURL(url),
env).MBeanServerConnection
def jvmInfo = new GroovyMBean(server,
'oc4j:j2eeType=JVM,name=single,J2EEServer=st
andalone')

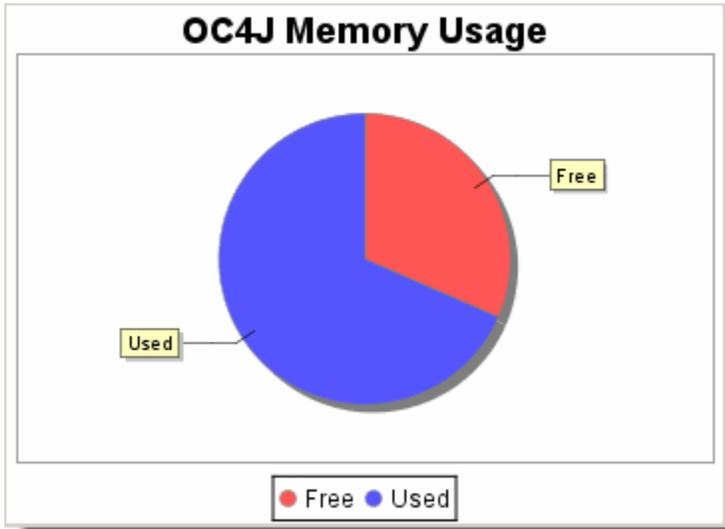
def piedata = new
org.jfree.data.general.DefaultPieDataset()
piedata.setValue "Free", jvmInfo.freeMemory
piedata.setValue "Used", jvmInfo.totalMemory
- jvmInfo.freeMemory

def options = [true, true, true]
def chart =
ChartFactory.createPieChart('OC4J Memory
Usage', piedata, *options)
chart.backgroundPaint = java.awt.Color.white
def swing = new groovy.swing.SwingBuilder()
def frame = swing.frame(title:'OC4J Memory
```

```
Usage',  
defaultCloseOperation:WC.EXIT_ON_CLOSE) {  
    panel(id: 'canvas') {  
        rigidArea(width:350, height:250) }  
    }  
    frame.pack()
```

```
frame.show()  
chart.draw(swing.canvas.graphics,  
swing.canvas.bounds)
```

Which looks like:



#### WebLogic Example

This script prints out information about the server followed by information about JMS Destinations (as an example). Many other mbeans are [available](#).

```
import javax.management.remote.*  
import javax.management.*  
import javax.naming.Context  
  
def urlRuntime =  
  '/jndi/weblogic.management.mbeanservers.runt  
ime'  
def urlBase =  
  'service:jmx:t3://localhost:7001'  
  
def serviceURL = new JMXServiceURL(urlBase +  
urlRuntime)
```

```

def h = new Hashtable()
h.put(Context.SECURITY_PRINCIPAL,
'weblogic')
h.put(Context.SECURITY_CREDENTIALS,
'weblogic')
h.put(JMXConnectorFactory.PROTOCOL_PROVIDER_
PACKAGES, 'weblogic.management.remote')
def server =
JMXConnectorFactory.connect(serviceURL,
h).MBeanServerConnection
def domainName = new
ObjectName('com.bea:Name=RuntimeService,' +

'Type=weblogic.management.mbeanservers.runti
me.RuntimeServiceMBean')
def rtName = server.getAttribute(domainName,
'ServerRuntime')
def rt = new GroovyMBean(server, rtName)
println "Server: name=$rt.Name,
state=$rt.State,
version=$rt.WeblogicVersion"
def destFilter =
Query.match(Query.attr('Type'),
Query.value('JMSDestinationRuntime'))
server.queryNames(new
ObjectName('com.bea:*'), destFilter).each{
name ->
    def jms = new GroovyMBean(server, name)
    println "JMS Destination:
name=$jms.Name, type=$jms.DestinationType"+

```

```
    " ,  
    messages=$jms.MessagesReceivedCount "  
}
```

Here is the output:

```
Server: name=examplesServer, state=RUNNING,  
version=WebLogic Server 10.0 Wed May 9  
18:10:27 EDT 2007 933139  
JMS Destination:  
name=examples-jms!exampleTopic, type=Topic,  
messages=0  
JMS Destination:  
name=examples-jms!exampleQueue, type=Queue,  
messages=0  
JMS Destination:  
name=examples-jms!jms/MULTIDATASOURCE_MDB_QU  
EUE, type=Queue, messages=0  
JMS Destination:  
name=examplesJMSServer!examplesJMSServer.Tem  
poraryQueue0, type=Queue, messages=68  
JMS Destination: name=examples-jms!quotes,  
type=Topic, messages=0  
JMS Destination:  
name=examples-jms!weblogic.wsee.wseeExamples  
DestinationQueue, type=Queue, messages=0  
JMS Destination:  
name=examples-jms!weblogic.examples.ejb30.Ex  
ampleQueue, type=Queue, messages=0
```

You can also use Spring to automatically register beans as JMX aware.

Here is an example class (Calculator.groovy):

```
import
org.springframework.jmx.export.annotation.*

@ManagedResource(objectName="bean:name=calcM
Bean", description="Calculator MBean")
public class Calculator {

    private int invocations

    @ManagedAttribute(description="The
Invocation Attribute")
    public int getInvocations() {
        return invocations
    }

    private int base = 10

    @ManagedAttribute(description="The Base
to use when adding strings")
    public int getBase() {
        return base
    }

    @ManagedAttribute(description="The Base
to use when adding strings")
    public void setBase(int base) {
        this.base = base
    }
```

```
    @ManagedOperation(description="Add two
numbers")
    @ManagedOperationParameters([
        @ManagedOperationParameter(name="x",
description="The first number"),
        @ManagedOperationParameter(name="y",
description="The second number")])
    public int add(int x, int y) {
        invocations++
        return x + y
    }
```

```
    @ManagedOperation(description="Add two
strings representing numbers of a particular
base")
    @ManagedOperationParameters([
        @ManagedOperationParameter(name="x",
description="The first number"),
        @ManagedOperationParameter(name="y",
description="The second number")])
    public String addStrings(String x,
String y) {
        invocations++
        def result = Integer.valueOf(x,
base) + Integer.valueOf(y, base)
        return Integer.toString(result,
```

```
base)
    }
}
```

Here is the Spring configuration file (beans.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema
/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/
spring-beans.xsd">
    <bean id="mbeanServer"

class="org.springframework.jmx.support.MBean
ServerFactoryBean">
        <property
name="locateExistingServerIfPossible"
value="true"/>
    </bean>

    <bean id="exporter"

class="org.springframework.jmx.export.MBeanE
xporter">
```

```
        <property name="assembler"
ref="assembler" />
        <property name="namingStrategy"
ref="namingStrategy" />
        <property name="beans">
            <map>
                <entry
key="bean:name=defaultCalcName"
value-ref="calcBean" />
            </map>
        </property>
        <property name="server"
ref="mbeanServer" />
        <property name="autodetect"
value="true" />
    </bean>

    <bean id="jmxAttributeSource"

class="org.springframework.jmx.export.annota
tion.AnnotationJmxAttributeSource" />

    <!-- will create management interface
using annotation metadata -->
    <bean id="assembler"

class="org.springframework.jmx.export.assemb
ler.MetadataMBeanInfoAssembler">
        <property name="attributeSource"
ref="jmxAttributeSource" />
    </bean>
```

```
<!-- will pick up the ObjectName from
the annotation -->
<bean id="namingStrategy"
class="org.springframework.jmx.export.naming
.MetadataNamingStrategy">
    <property name="attributeSource"
ref="jmxAttributeSource" />
</bean>

<bean id="calcBean"
class="Calculator">
```

```
        <property name="base" value="10" />
    </bean>
</beans>
```

Here is a script which uses this bean and configuration:

```
import
org.springframework.context.support.ClassPat
hXmlApplicationContext
import
java.lang.management.ManagementFactory
import javax.management.ObjectName
import javax.management.Attribute

// get normal bean
def ctx = new
ClassPathXmlApplicationContext("beans.xml")
def calc = ctx.getBean("calcBean")

Thread.start{
    // access bean via JMX, use a separate
thread just to
    // show that we could access remotely if
we wanted
    def server =
ManagementFactory.platformMBeanServer
    def mbean = new GroovyMBean(server,
'bean:name=calcMBean')
    sleep 1000
    assert 8 == mbean.add(7, 1)
```

```
mbean.Base = 8
assert '10' == mbean.addStrings('7',
'1')
mbean.Base = 16
sleep 2000
println "Number of invocations:
$mbean.Invocations"
println mbean
}

assert 15 == calc.add(9, 6)
```

```
assert '11' == calc.addStrings('10', '1')
sleep 2000
assert '20' == calc.addStrings('1f', '1')
```

And here is the resulting output:

```
Number of invocations: 5
MBean Name:
  bean:name=calcMBean

Attributes:
  (rw) int Base
  (r) int Invocations
Operations:
  int add(int x, int y)
  java.lang.String
  addStrings(java.lang.String x,
  java.lang.String y)
  int getInvocations()
  int getBase()
  void setBase(int p1)
```

You can even attach to the process while it is running with [jconsole](#). It will look something like:  
!jconsole.gif!We started the Groovy application with the `-Dcom.sun.management.jmxremote` JVM argument using a Java 5 JVM.

See also:

- [Dynamic language beans in Spring](#)
- [Using Spring Factories with Groovy](#)
- [Spring JMX Documentation](#)

## Troubleshooting

*groovy.lang.MissingMethodException or groovy.lang.GroovyRuntimeException*

If you get an error like this

```
groovy.lang.MissingMethodException: No  
signature of method:  
javax.management.remote.rmi.RMIConnector$Rem  
oteMBeanServerConnection.queryMBeans()  
is applicable for argument types:  
(javax.management.ObjectName, null)  
values: {Catalina:*, null}
```

or like this:

```
Caught: groovy.lang.GroovyRuntimeException:  
Could not find matching constructor for:  
groovy.util.GroovyMBean(javax.management.rem  
ote.rmi.RMIConnector$RemoteMBeanServerConnec  
tion, java.lang.String)
```

you have to move away or delete "mx4j-\*.jar" from "\$GROOVY\_HOME/lib". MX4J is designed to add javax.management classes to 1.4 JVMs. If you already have a newer JMX jar on your classpath or are using a Java 5 or higher JVM, the MX4J classes will be incompatible with the ones from the newer Sun JVMs or newer versions of JMX.

*java.lang.SecurityException*

If you get the following error, your container's JMX access is password protected:

```
java.lang.SecurityException: Authentication  
failed! Credentials required
```

To fix that, add an environment with the credentials when connecting, like this (password has to be set before that):

```
def jmxEnv = null
if (password != null)
    jmxEnv = [(JMXConnector.CREDENTIALS):
(String[])["monitor", password]]
def connector =
JMXConnectorFactory.connect(new
JMXServiceURL(serverUrl), jmxEnv)
```

Details for the software you are trying to monitor/manage may differ slightly. Check out the other examples using credentials above if appropriate (e.g. OC4J and WebLogic). If you still have troubles, you will have to consult the documentation for the software you are trying to monitor/manage for details on how to provide credentials.

#### Further JMX Information

- [Groovy JmxBuilder](#)
- [Monitoring the Java Virtual Machine](#)
- [Using Groovy for System Management](#)
- [JMX Scripts using JRuby - Part I](#)
- [JMX Scripts using JRuby - Part II](#)
- [Groovier jconsole!](#)
- [JMX Scripts with Eclipse Monkey](#)
- [Using JMX to monitor Apache ActiveMQ](#)
- [Jagger project \(JMX application monitoring with Groovy\)](#)

#### Groovy JmxBuilder

**JmxBuilder is a Groovy-based domain specific language for the Java Management Extension (JMX) API.** It uses the builder pattern (FactoryBuilder) to create an internal DSL that facilitates the exposure of POJO's and Groovy beans as management components via the MBean server. JmxBuilder hides the complexity of creating and exporting management beans via the JMX API and provides a set of natural Groovy constructs to interact with the JMX infrastructure.

#### Instantiating JmxBuilder

To start using JmxBuilder, simply make sure the jar file is on your class path. Then you can do the following in your code

```
def jmx = new JmxBuilder()
```

**That's it!** You are now ready to use the JmxBuilder.

**NOTE:**

- You can pass in an instance of **your own MBeanServer** to the builder (**JmxBuilder(MBeanServer)**)
- If no MBeanServer is specified, the builder instance will default to the underlying platform MBeanServer.

Once you have an instance of JmxBuilder, you are now ready to invoke any of its builder nodes.

## JMX Connectors

Remote connectivity is a crucial part of the JMX architecture. JmxBuilder facilitates the creation of connector servers and connector clients with minimal amount of coding.

### Connector Server

JmxBuilder.connectoServer() supports the full Connector api syntax and will let you specify properties, override the URL, specify your own host, etc.

### Syntax

```
jmx.connectorServer(  
    protocol:"rmi",  
    host:"...",  
    port:1099,  
    url:"...",  
    properties:[  
        "authenticate":true|false,  
        "passwordFile":"...",  
        "accessFile":"...",  
        "sslEnabled" : true | false  
        // any valid connector property  
    ]  
)
```

Note that the serverConnector node will accept four ServerConnector property aliases (authenticate, passwordFile, accessFile, and sslEnabled). You can use these aliases or provided any of the RMI-supported properties.

**Example - Connector Server (see correction below)**

```
jmx.connectorServer(port: 9000).start()
```

The snippet above returns an RMI connector that will start listening on port 9000. By default, the builder will internally generate URL "**service:jmx:rmi:///jndi/rmi://localhost:9000/jmxrmi**"

*NOTE: Sadly you are as likely to get something like the following when attempting to run the previous snippet of code (example is incomplete, see below):*

```
Caught: java.io.IOException: Cannot bind to
URL [rmi://localhost:9000/jmxrmi]:
javax.naming.ServiceUnavailableException
[Root exception is
java.rmi.ConnectException: Connection
refused to host: localhost; nested exception
is:
    java.net.ConnectException: Connection
refused]
```

*This occurs on Mac and Linux (CentOS 5) with Groovy 1.6 installed. Perhaps there were assumptions made about the configuration of the /etc/hosts file?*

*NOTE: The correct example is shown below.*

#### **Connector Example (Corrected) - Connector Server**

The example above does not create the RMI registry. So, in order to export, you have to first export the RMI object registry (make sure to import `java.rmi.registry.LocateRegistry`).

```
import java.rmi.registry.LocateRegistry
...

LocateRegistry.createRegistry(9000)
jmx.connectorServer(port: 9000).start()
```

#### **Connector Client**

`JmxBuilder.connectorClient()` node lets you create JMX connector client object to connect to a JMX MBean Server.

#### **Syntax**

```
jmx.connectorClient (
    protocol:"rmi",
    host:"...",
    port:1099,
    url:"...",
)
```

### Example - Client Connector

Creating a connector client can be done just as easily. With one line of code, you can create an instance of a JMX Connector Client as shown below.

```
def client = jmx.connectorClient(port: 9000)
client.connect()
```

You can then access the MBeanServerConnection associated with the connector using:

```
client.getMBeanServerConnection()
```

### JmxBuilder MBean Export

You can **export a Java object or a Groovy object** with minimal coding. JmxBuilder will even find and **export dynamic Groovy methods** injected at runtime.

#### *Implicit vs Explicit Descriptors*

When using the builder, you can **let JmxBuilder implicitly generate** all of your MBean descriptor info. This is useful when you want to write minimal code to quickly export your beans. You can also explicitly declare all descriptor info for the bean. This gives you total control on how you want to describe every piece of information that you want to export for the underlying bean.

#### The JmxBuilder.export() Node

The **JmxBuilder.export() node provides a container** where all management entities to be exported to the MBeanServer are placed. You can place one or more bean() or timer() nodes as children of the export() node. JmxBuilder will **automatically batch export the entities described** by the nodes to the MBean server for management (see example below).

```
def beans = jmx.export {
    bean(new Foo())
    bean(new Bar())
    bean(new SomeBar())
}
```

In the code snippet above, **JmxBuilder.export()** will export three management beans to the MBean server.

#### JmxBuilder.export() Syntax

JmxBuilder.export() node supports the **registrationPolicy** parameter to specify how JmxBuilder will behave to resolve bean name collision during MBean registration:

```
jmx.export(policy: "replace | ignore | error")
or
jmx.export(regPolicy: "replace | ignore | error")
```

**replace** - JmxBuilder.export() will replace any bean already registered with the MBean during export.

**ignore** - The bean being exported will be ignored if the same bean is already registered.

**error** - JmxBuilder.export() throws an error upon bean name collision during registration.

#### Integration with GroovyMBean Class

When you export an MBean to the MBeanServer, **JmxBuilder will return an instance of GroovyMBean** representing the management bean that have been exported by the builder. Nodes such as **bean()** and **timer()** will return an instances of GroovyMBean when they are invoked. The **export()** node returns an **array of all of GroovyMBean[]** representing all managed objects exported to the MBean server.

#### MBean Registration with JmxBuilder.bean()

This portion of this reference uses class **RequestController** to illustrate how to use JmxBuilder to export runtime management beans. The class is for illustration purpose and can be a POJO or a Groovy bean.

- RequestController

```

public class RequestController {
    // constructors
    public RequestCopntroller()
    public RequestController(Map resource)

    // attributes
    public boolean isStarted() { ... }
    public int getRequestCount(){ ... }
    public int getResourceCount() { ... }
    public void setRequestLimit(int limit){ ... }
    public int getRequestLimit() { ... }

    // operations
    public void start(){ ... }
    public void stop(){ ... }
    public void putResource(String name, Object resource){ ... }
    public void makeRequest(String res) { ... }
    public void makeRequest() { ... }
}

```

#### *Implicit Export*

As mentioned earlier, you can use JmxBuilder's flexible syntax to export any POJO/POGO with no descriptor. The builder can automatically describe all aspects of the management beans using implicit defaults. These default values can easily be overridden as we'll see in this in the next section.

The simplest way to export a POJO or POGO is listed below.

```

jmx.export {
    bean(new
RequestController(resource:"Hello World"))
}

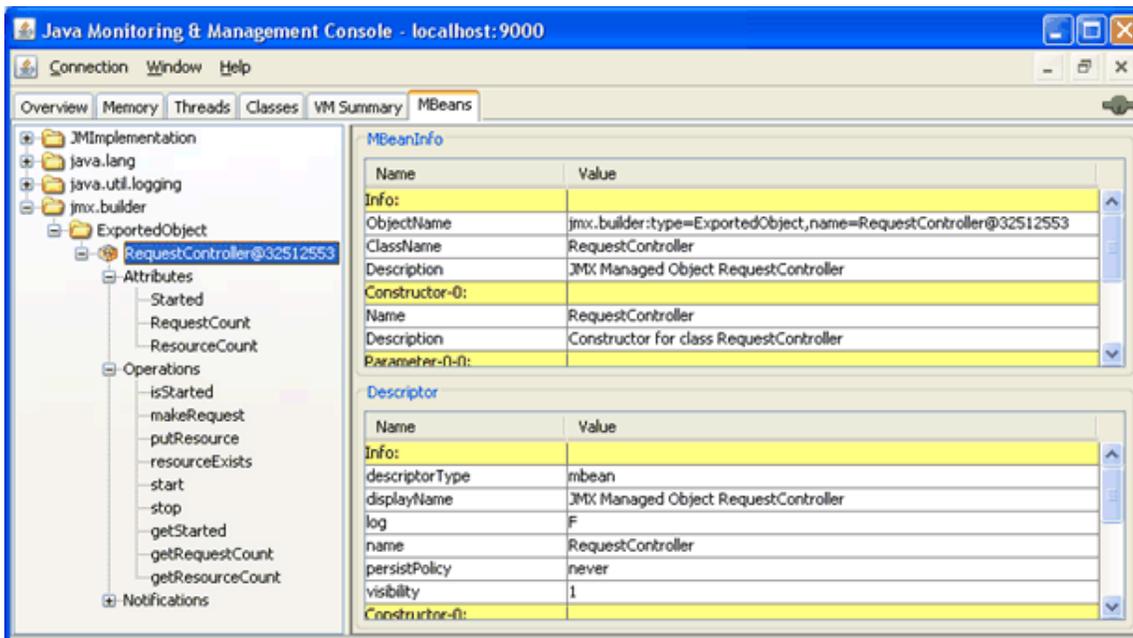
```

#### **What this does:**

- First, the **JmxBuilder.export()** node will export an MBean to the MBeanServer representing **the declared POJO** instance.
- The builder will **generate a default ObjectName** for the MBean and all other MBean descriptor information.
- **JmxBuilder will automatically export** all declared **attributes** (MBean getter/setters), **constructors**, and **operations** on the instance.
- The exported **attributes** will have **read-only** visibility.

Remember, **JmxBuilder.export()** returns an array of **GroovyMBean[]** objects for all exported instances. So, once you call **JmxBuilder.export()**, you have **immediate access to the underlying MBean proxy** (via **GroovyMBean**).

## **JConsole view of Exported Bean**



### JmxBuilder.bean() Syntax

The JmxBuilder.bean() node supports an extensive set of descriptors to describe your bean for management. The JMX MBeanServer uses these descriptors to expose meta data about the bean exposed for management.

```

jmx.export {
  bean(
    target:bean instance,
    name:ObjectName,
    desc:"...",
    attributes:"*",
    attributes:[
      "Attribute1", "Attribute2", ..., "Attribute_n" ]
    attributes:[
      "AttributeName": "*",
      "AttributeName": [
        desc:"...",
        defaultValue:value,
        writable:true|false,

```

```

        editable:true|false,
        onChange:{event-> // event
handler}
    ]
],

constructors:"*",
constructors:[
    "Constructor Name":[],
    "Constructor Name":[
"ParamType1", "ParamType2, ..., ParamType_n" ],
    "Constructor Name":[
        desc:"...",
        params:[
            "ParamType1":"*",

"ParamType2":[desc:"...", name:"..."], ...,

"ParamType_n":[desc:"...", name:"..."]
        ]
    ],

operations:"*",
operations:[ "OperationName1",
"OperationName2", ..., "OperationNameN" ],
operations:[
    "OperationName1":"*",
    "OperationName2":[
"type1", "type2", "type3" ]
    "OperationName3":[

```

```
        desc:"...",
        params:[
            "ParamType1": "*"
        ],
        "ParamType2": [desc:"...", name:"..."], ...,
        "ParamType_n": [desc:"...", name:"..."]
            ],
            onInvoked:{event->
JmxBuilder.send(event:"", to:"")
        }
    ],

    listeners:[
        "ListenerName1": [event: "...",
from:ObjectName, call:{event->}],
        "ListenerName2": [event: "...",
from:ObjectName, call:&methodPointer]
    ]
}
```

```
    )  
}
```

Instead of describing the entire node, the following section explore each attribute separately.

#### **Bean() Node - Specifying MBean ObjectName**

Using the bean() node descriptors, you can specify your own MBean ObjectName.

```
def ctrl = new  
RequestController(resource:"Hello World")  
def beans = jmx.export {  
    bean(target:ctrl,  
name:"jmx.tutorial:type=Object")  
}
```

The ObjectName can be specified as a String or an instance of the ObjectName.

#### **Bean() Node - Attribute Export**

JMX attributes are the setters and getters on the underlying bean. The JmxBuilder.bean() node provides several ways to flexibly describe and export MBean attributes. You can combine them however you want to achieve any level of attribute visibility. Let's take a look.

##### *Export All Attributes with Wildcard "\*"*

The following code snippet **will describe and export all attributes** on the bean as read-only. **JmxBuilder will use default values** to describe the attributes that exported for management.

```
def objName = new  
ObjectName("jmx.tutorial:type=Object")  
def beans = jmx.export {  
    bean(target: new RequestController(),  
name: objName,  
attributes: "*")  
}
```

### *Export Attribute List*

JmxBuilder will let you specify a list of attributes to export.

```
def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    attributes: [ "Resource",
"RequestCount" ]
  )
}
```

In the snippet above, **only the "Resource" and "RequestCount" attributes will be exported**. Again, since no descriptors are provided, **JmxBuilder will use sensible defaults** to describe the exported attributes.

### *Export Attribute with Explicit Descriptors*

One of the strengths of JmxBuilder is its flexibility in describing MBean. With the builder you can describe all aspects of the MBeans attribute that you want to export to the MBeanServer (see syntax above).

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    attributes: [
      "Resource":[desc: "The resource
to request.", readable: true, writable:
true, defaultValue:"Hello"],
      "RequestCount": "*"
    ]
  )
}

```

In the snippet above, attribute **"Resource"** is **fully-described** using all supported descriptors (i.e. desc, readable, writable, defaultValue) for a JMX attribute. However, we use the wildcard to describe attribute **RequestCount** and it will be exported and described using defaults.

#### Bean() Node - Constructor Export

JmxBuilder **supports the explicit description and export of constructors** defined in the underlying bean. There are several options available when exporting constructors. You can combine them however you want to achieve the desired level of manageability.

#### *Export all Constructors with "\*"*

You can use the builder's special `""` **notation to \*export all constructors** declared on the underlying bean. The builder will use default values to describe the MBean constructors.

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    constructors: "*"
  )
}

```

*Export Constructors using Parameter Descriptor*

JmxBuilder lets you **target specific constructor** to export by **describing the parameter signature**. This is useful when you have several constructors with different parameter signature and you want to export specific constructors.

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    constructors: [
      "RequestController": [ "Object" ]
    ]
  )
}

```

Here, JmxBuilder will **export a constructor that takes one parameter of type "Object"**. Again, JmxBuilder will use default values to fill in the description of the constructor and the parameters.

*Export Constructor with Explicit Descriptors*

JmxBuilder allows you to **fully-describe** the constructor that you want to target for export (see syntax above).

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
    bean(target: new RequestController(),
name: objName,
        constructors:[
            "RequestController":[
                desc:"Constructor takes
param",
                params:[ "Object" :
[name:"Resource", desc:"Resource for
controller"] ]
            ]
        ]
    )
}

```

In the code above, JmxBuilder will target a constructor that takes one parameter for export to the MBeanServer. Notice how the constructor can be fully-described using all optional descriptor keys including parameter descriptors.

#### Bean() Node - Operation Export

Similar to constructors, JmxBuilder supports the description and export of MBean operations using a flexible notation (see above for syntax). You can combine these notations however you want to achieve the level of operation manageability desired.

#### *Export All Operations with "\*"*

You can use the builder's special "\*" notation to **export all operations** defined on the bean to be exposed for management. The builder will use default descriptor values for the operations being exported.

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    operations: "*"
  )
}

```

In this snippet, JmxBuilder will **export all bean operations** and will use default values to describe them in the MBeanServer.

#### *Export Operation List*

JmxBuilder has a shorthand notation that lets you quickly target operations to be exported by providing a list of methods to export.

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    operations: [ "start", "stop" ]
  )
}

```

In the snippet above, the **builder will only export methods start() and stop()**. All other methods will be ignored. JmxBuilder will use default descriptor values to describe the operations being exported.

#### *Export Operations by Signature*

Using JmxBuilder, you can target methods to export for management using the methods's parameter signature. This is useful when you want to distinguish methods with the same name that you want to export (i.e. stop()) instead of

stop(boolean)).

```
def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
  bean(
    target: new RequestController(),
    name: objName,
    operations: [
      "makeRequest":[ "String" ]
    ]
  )
}
```

In the snippet above, JmxBuilder would **select method makeRequest(String)** to be exported instead of the other version makeRequest() which takes no parameter. In this shorthand context, the signature is specified as a list of type (i.e. "String").

#### *Export Operations with Explicit Descriptors*

JmxBuilder supports detailed descriptors for bean operations. You can supply deep descriptor info about any operation on your bean including a name, description, method parameters, parameter type, and parameter description.

```

def objName = new
ObjectName("jmx.tutorial:type=Object")
def beans = jmx.export {
    bean(target: new RequestController(),
name: objName,
    operations: [
        "start": [desc:"Starts request
controller"],
        "stop": [desc:"Stops the request
controller"],
        "setResource": [params: [ "Object"
]],
        "makeRequest": [
            desc:"Executes the
request.",
            params: [

"String": [name:"Resource", desc:"The resource
to request" ]
        ]
    ]
)
}

```

The snippet above shows all of the ways JmxBuilder allows you to describe an operation targeted for management:

- Operations **start()** and **stop()** are described by the "desc" key (this is enough since there are no params).
- In operation **setResource()** uses of a shorthand version of **params:** to describe the parameters for the method.
- **makeRequest()** uses the the extended descriptor syntax to describe all aspects of the operation.

## Embedding Descriptor

JmxBuilder supports the ability to **embed descriptors directly in your Groovy class**. So, instead of wrapping your description around the declared object (as we've seen here), you can embed your JMX descriptors directly in your class.

- RequestControllerGroovy \*

```
public class RequestController {
    // attributes
    boolean started
    int requestCount
    int resourceCount
    int requestLimit
    Map resources

    // operations
    void start(){ ... }
    void stop(){ ... }
    void putResource(String name, Object resource){ ... }
    void makeRequest(String res) { ... }
    void makeRequest() { ... }

    static descriptor = [
        name: "jmx.builder:type=EmbeddedObject",
        operations: [ "start", "stop", "putResource" ]
        attributes:"*"
    ]
}

....
// export
jmx.export(
    bean(new RequestControllerGroovy())
)
```

There are two things going on in the code above:

1. Groovy class RequestControllerGroovy is defined and includes a **static descriptor** member. That member is used to declare a JmxBuilder descriptor to describe member of the class targeted for JMX export.
2. The second part of the code shows how to use JmxBuilder to export that class for management.

## Timer Export

JMX standards mandate that the implementation of the API makes available a timer service. Since JMX is a component-based architecture, timers provide an excellent signaling mechanism to communicate to registered listener components in the MBeanServer. JmxBuilder supports the creation and export of timers using the same easy syntax we've seen so far.

## Timer Node Syntax

```

timer(
    name:ObjectName,
    event:"..." ,
    message:"..." ,
    data:dataValue
    startDate:"now" | dateValue
    period:"99d" | "99h" | "99m" | "99s" | 99
    occurrences:long
)

```

The timer() node supports several attributes:

- **name:** - [Required](#) The qualified JMX ObjectName instance (or String) for the timer.
- **event:** - The JMX event type string that will be broadcast with every timing signal (default "jmx.builder.event").
- **message:** - An optional string value that can be sent to listeners.
- **data:** - An optional object that can be sent to listeners of timing signal.
- **startDate:** - When to start timer. Set of valid values [ "now", date object ]. Default is "now"
- **period:** - A timer's period expressed as either a number of millisecond or time unit (day, hour, minute, second). See description below.
- **occurrences:** - A number indicating the number of time to repeat timer. Default is forever.

#### *Exporting a Timer*

```

def timer = jmx.timer(name:
"jmx.builder:type=Timer", event:
"heartbeat", period: "1s")
timer.start()

```

This snippet above **describes, creates, and exports a standard JMX Timer** component. Here, the **timer()** node **returns a GroovyMBean** that represents the registered timer MBean in the MBeanServer.

An **alternative way of exporting timers** is within the JmxBuilder.export() node.

```
def beans = jmx.export {
    timer(name: "jmx.builder:type=Timer1",
event: "event.signal", period: "1s")
    timer(name: "jmx.builder:type=Timer2",
event: "event.log", period: "1s")
}
beans[0].start()
beans[1].start()
```

#### *Timer Period*

The `timer()` node supports a flexible notation for specifying the **timer period values**. You can specify the time in second, minutes, hour, and day. The default is millisecond.

- `timer(period: 100)` = 100 millisecond
- `timer(period: "1s")` = 1 second
- `timer(period: "1m")` = 1 minute
- `timer(period: "1h")` = 1 hour
- `timer(period: "1d")` = 1 day

The node will automatically translate.

#### *JmxBuilder and Events*

An integral part of **JMX** is its **event model**. Registered management beans can **communicate with each other by broadcasting events** on the MBeanServer's event bus. **JmxBuilder provides several ways to easily listen and react to events** broadcasted on the MBeanServer's event bus. Developers can **capture any event on the bus or throw their own** to be consumed by other components registered on the MBeanServer.

#### *Event Handling Closures*

JmxBuilder leverages Groovy's use of closures to provide simple, yet elegant, mean of reacting to JMX events. JmxBuilder supports two closure signatures:

## Parameterless

```
callback = {->
    // event handling code here.
}
```

JmxBuilder executes the closure and passes no information about the event that was captured on the bus.

## With Event Parameter

```
callback = {event ->
    // event handling code
}
```

JmxBuilder will pass an "event" object to the closure using this format. The event object contains information about the event was intercepted so that it can be handled by the handler. The parameter will contain different set of info depending on the event that was captured.

### *Handling Attribute onChange Event*

When describing attributes (see bean() node section above), you can provide a closure (or method pointer) for callback to be executed when the value of the attribute is updated on the exported MBean. This gives developers an opportunity to listen to and react to state changes on the MBean.

```
jmx.export {
    bean(
        target: new RequestController(),
    name: "jmx.tutorial:type=Object",
        attributes: [
            "Resource": [
                readable: true, writable:
true,
                onChange: {e ->
                    println e.oldValue
                    println e.newValue
                }
            ]
        ]
    )
}
```

The sample snippet above shows how to specify an "onChange" callback closure when describing MBean

attributes. In this sample code, whenever attribute "Resource" is updated via the exported MBean, the **onChange event will be executed**.

*Attribute onChange Event Object*

When handling the attribute onChange event, the handler closure will receive an event object with the following info:

- event.**oldValue** - the previous attribute value before the change event.
- event.**newValue** - the new value of the attribute after the change.
- event.**attribute** - the name of the attribute on which the event occurred.
- event.**attributeType** - the data type of the attribute that causes the event.
- event.**sequenceNumber** - a numeric value representing the sequence number of event.
- event.**timeStamp** - a time stamp for the event occurrence.

*Handling Operation onCall Event*

Similar to mbean attributes, JmxBuilder affords developers the **ability to listen for operation invocation** on an MBean registered in the MBeaServer. JmxBuilder accepts a **callback closure that will be executed after the MBean method has invoked**.

```

class EventHandler {
    void handleStart(e){
        println e
    }
}

def handler = new EventHandler()

def beans = jmx.export {
    bean(target: new RequestController(),
name: "jmx.tutorial:type=Object",
    operations: [
        "start": [
            desc:"Starts request
controller"
            onCall:handler.&handleStart
        ]
    ]
)
}

```

The snippet above **shows how to declare an "onCall" closure to be used as listener** when operation "start()" is invoked on the MBean. This sample **uses the method pointer syntax** to illustrate the versatility of JmxBuilder.

#### *Operation onCall Event Object*

When handling the operation onCall event, the callback closure will receive an event object with the following info:

- event.**event** - the event type string that was broadcasted.
- event.**source** - The object on which the method was invoked.
- event.**data** - the data type of the attribute that causes the event.
- event.**sequenceNumber** - a numeric value representing the sequence number of event.
- event.**timeStamp** - a time stamp for the event occurrence.

#### **Listener MBean**

When you export an MBean with the bean() node, you can define events the MBean can listen and react to. The

bean() node provides a "listeners:" attribute that lets you define event listeners that your bean can react to.

```
def beans = jmx.export {
  timer(name: "jmx.builder:type=Timer",
  event: "heartbeat", period: "1s").start()
  bean(target: new RequestController(),
  name: "jmx.tutorial:type=Object",
  operations: "*",
  listeners: [
    heartbeat:[

from:"jmx.builder:type=Timer",
      call:{e ->
        println e
      }
    ]
  ]
)
}
```

In the sample above, we see the **syntax for adding listeners to an exported MBean**.

- First, a **timer is exported** and started.
- Then an **MBean is declared that will listen to the timer** event and do something meaningful.
- The **"heartbeat:"** name is arbitrary and has no correlation to the timer declared above.
- The **source** of the event **is specified using the "from:" attribute**.

You can also specify an **event type** you are interested in receiving from a broadcaster (since a broadcaster can be emitting multiple events).

#### *Listening to JMX Events*

In some cases, you will want to create stand-alone event listeners (not attached to exported MBeans). JmxBuilder provides the Listener() node to let you create JMX listeners that can listen to MBeanServer events. This is useful when creating JMX client applications to monitor/manage JMX agents on remote JMX MBeanServers.

#### *Listener Node Syntax*

```
jmx.listener(  
    event: "...",  
    from: "object name" | ObjectName,  
    call: {event->}  
)
```

Here is the description of the `listener()` node attributes:

- `event`: An optional string that identifies the JMX event type to listen for.
- `from` (required): The JMX ObjectName of the component to listen to. This can be specified as a string or an instance of ObjectName
- `call`: The closure to execute when the event is captured. This can also be specified as a Groovy method pointer.

Here is an example of JmxBuilder's listener node:

```
jmx.timer(name: "jmx.builder:type=Timer",  
period: "1s").start()  
  
jmx.listener(  
    from: "jmx.builder:type=Timer",  
    call: {e ->  
        println "beep..."  
    }  
)
```

This example shows how you can use a stand alone listener (outside of an MBean export). Here, we **export a timer with a 1 second** resolution. Then, we specify a listener to that timer that will print "beep" every second.

#### Emitting JMX Events

JmxBuilder provides the **tools needed to broadcast your own events** on the MBeanServer's event bus. There are no restrictions on the event type you can broadcast. You simply **declare your emitter** and the event type that you want to send, then **broadcast your event** at any time. Any registered component in the MBeanServer can register themselves to listen to your events.

#### Emitter Syntax

```
jmx.emitter(name: "ObjectName",  
event: "type")
```

The attributes for the node `Emitter()` can be summarized as follows:

- `name`: an optional JMX `ObjectName` used to register your emitter in the `MBeanServer`. Default is `jmx.builder:type=Emitter,name=Emitter@OBJECT_HASH_VALUE`
- `event`: an option string value that describes the JMX event type. Default is `"jmx.builder.event.emitter"`.

#### *Declare the Emitter*

```
def emitter = jmx.emitter()
```

The snippet **declares the emitter using implicit descriptor syntax**. `JmxBuilder` will do the followings:

- Create and register an emitter `MBean` with a default `ObjectName`.
- Setup a **default event type** with value `"jmx.builder.event.emitter"`.
- Return a `GroovyMBean` representing the emitter.

As with other nodes in the builder, **you can override all keys in the `emitter()` node**. You can specify the **ObjectName** and the **event type**.

#### *Broadcast Event*

Once you have declared your emitter, you can broadcast your event.

```
emitter.send()
```

The sample above shows the **emitter sending an event**, once it has been declared. Any JMX component registered in the `MBeanServer` can register to receive message from this emitter.

#### *Sending Event Objects*

You can optionally pass data to the receiver when you send the message.

```
...  
emitter.send("Hello!")  
...
```

If you use an **event listener closure** (see above) that **accepts a parameter**, you can access that value.

## Groovy Categories

There are many situations where you might find that it would be useful if a class not under your control had additional methods that you define. In order to enable this capability, Groovy implements a feature borrowed from [Objective-C](#), called Categories. There are a few categories that are included in the system for adding functionality to classes that make them more usable within the Groovy environment.

- [DOMCategory](#)
- [ServletCategory](#)

The first category allows you to treat DOM objects as arrays and maps so that you can use them in conjunction with the Groovy path expression language and treat them like JavaBeans. Here is an example from the tests of using the DOMCategory:

## DOMTest.groovy

```
import groovy.xml.*

def html = DOMBuilder.newInstance().html {
  head {
    title (class:'mytitle', 'Test')
  }
  body {
    p (class:'mystyle', 'This is a test.')
  }
}

use (groovy.xml.dom.DOMCategory) {
  assert html.head.title.text() == 'Test'
  assert html.body.p.text() == 'This is a
test.'
  assert html.find{ it.tagName == 'body'
}.tagName == 'body'
  assert
html.getElementsByTagName('*').grep{
it.'@class' }.size() == 2
}

try {
  html.head
} catch(MissingPropertyException mpe) {
  println "Categories wear off"
}
```

As you can see here we are treating DOM objects just as if they were JavaBeans and are accessing them with GPath. The ServletCategory is similarly used when we want to treat the attributes of Servlet API objects as if they were properties since they don't follow the typical conventions for JavaBeans or Maps either. In the GroovyServlet that lets you use scripts as servlets we call GroovyCategorySupport from Java in order to make it possible to use property accessors against the request:

## GroovyServlet.java

```
Closure closure = new Closure(gse) {  
    public Object call() {  
        try {  
            return ((GroovyScriptEngine)  
getDelegate()).run(scriptUri, binding);  
        } catch (ResourceException e) {  
            throw new RuntimeException(e);  
        } catch (ScriptException e) {  
            throw new RuntimeException(e);  
        }  
    }  
};  
GroovyCategorySupport.use(ServletCategory.cl  
ass, closure);
```

This allows users to access things like Session attributes and request Attributes by name instead of through the API within their Groovy servlet scripts. For example, without this you would have to do:

```
if (session.getAttribute("count") == null)  
then session.setAttribute("count", 1);
```

With this you can say it more tersely as:

```
if (session.count == null) session.count =
1;
```

In order to create your own Categories and extend classes yourself you'll need to understand what the "use" keyword expects to be defined within the class you pass to it. To add a method to a class T, simply define a new class with a static method whose first parameter is of type T. Here is a simple example from the tests:

### CategoryTest.groovy

```
class StringCategory {
    static String lower(String string) {
        return string.toLowerCase()
    }
}

use (StringCategory) {
    assert "test" == "TeSt".lower()
}
```

This code will print out the string "test". This facility is extremely powerful and essentially lets you change the way any class in the system works when it is called from Groovy code. Note though that you can't add methods to classes, pass them to Java code, and expect the Java code to be able to call them. Since most people use statically typed Java with little reflection I doubt this case would come up much anyway.

Here is an example of using this as an end user in order to add methods to Apple's own NSDictionary and NSArray class in order to manipulate their Cocoa objects as if they were native Groovy objects:

### bookmarks.groovy

```
#!/Users/sam/bin/groovy
// Put /System/Library/Java in your
CLASSPATH
import groovy.xml.*;
```

```
import groovy.xml.dom.*;
import java.io.*;
import com.apple.cocoa.foundation.*;

class PropertyListCategory {
    static Object get(NSDictionary
dictionary, String key) {
        return
dictionary.objectForKey(key);
    }
    static Object getAt(NSArray array,
int i) {
        return
array.objectAtIndex(i);
    }
    static void each(NSArray array,
Closure closure) {
        for (i in
0..array.count()-1) {

closure.call(array[i]);
        }
    }
}

filename =
"${System.getProperty("user.home")}/Library/
Safari/Bookmarks.plist";
data = new NSData(new File(filename));
errorString = new String[1];
format = new int[1];
```

```
plist =
NSPropertyListSerialization.propertyListFrom
Data(data,

NSPropertyListSerialization.PropertyListImmu
table, format, errorString);

if (errorString[0]) {
    println "Error: ${errorString[0]}";
    System.exit(1);
}

def getURLs(NSArray array, list) {
    array.each {
        getURLs(it, list);
    }
}

def getURLs(NSDictionary dict, list) {
    if (dict.Children != null)
getURLs(dict.Children, list);
    if (dict.URIDictionary != null) {

list.add([title:dict.URIDictionary.title,
url:dict.URLString]);
    }
}

def getURLs(NSDictionary dict) {
    use (PropertyListCategory) {
        def list = [];
```

```
        getURLs(dict, list);  
        return list;  
    }
```

```
}
```

```
println getURLs(plist);
```

Notice how we can even create Category classes in Groovy code. They essentially look just like built-in ones within `DefaultGroovyMethods`. Define them by creating a static method that takes the type you want to extend, then the additional parameters that the new method will take.

#### *Advanced Usage*

A category needs not to be directly exposed to the user code, the following will also do:

### Closure, Category and JPA example

```
class JPACategory{  
    // Let's enhance JPA EntityManager without  
getting into the JSR committee  
    static void persistAll(EntityManager em ,  
Object[] entities) { //add an interface to  
save all  
        entities?.each { em.persist(it) }  
    }  
}  
  
def transactionContext = {  
    EntityManager em, Closure c ->  
    def tx = em.transaction  
    try {  
        tx.begin()  
        use(JPACategory) {  
            c()  
        }  
        tx.commit()  
    } catch (e) {
```

```
    tx.rollback()
  } finally {
    //cleanup your resource here
  }
}
```

// user code, they always forget to close resource in exception, some even forget to commit, let's not rely on them.

```
EntityManager em; //probably injected
transactionContext (em) {
  em.persistAll(obj1, obj2, obj3)
  // let's do some logics here to make the
```



If you have a groovy script, you can edit and run the script immediately.

```
$ cat test.groovy  
println 'Hello Bonson'  
  
$ groovy test.groovy  
Hello Bonson
```

Here is an example with your own command line arguments.

```
$ cat test.groovy  
println 'Hello ' + args[0]  
  
$ groovy test.groovy Jeeves  
Hello Jeeves
```

However you can also run such a simple groovy program by providing the script in the command line arguments.

```
$ groovy -e "println 'Hello Bob' "  
Hello Bob
```

This may not look useful, but it fits in with the UNIX tradition of chaining simple programs together to build powerful commands. Tools like perl, sed, awk and grep do these jobs very well. But many users have limited experience with these tools' arcane syntax and will be more familiar with Java and therefore Groovy.

```
$ grep -i ^groov /usr/share/dict/words |  
groovy -e 'print  
System.in.text.toUpperCase() '  
GROOVE  
GROOVELESS  
GROOVELIKE  
GROOVER  
GROOVERHEAD  
GROOVINESS  
GROOVING  
GROOVY
```

Because looping through STDIN or input files tends to be a common thing to do, groovy (and ruby, perl etc) provide shortcuts for this. **currently broken, groovy not flushing output (still so 060927?)**

-n will loop through each line of the input, and provide it to your script in the *line* variable.

```
grep -i ^groov /usr/share/dict/words |  
groovy -n -e 'println line.toUpperCase() '
```

If we definitely want to print the output of each line we can use -p and shorten it to

```
grep -i ^groov /usr/share/dict/words |  
groovy -p -e 'line.toUpperCase() '
```

We can use the looping constructs along with -i, which writes the output back to the original files (and creates a backup copy with the given extension). And wreak havoc on our local file system, with wide-scale search and replace.

```
groovy -p -i .bak -e '(line =~  
"<h\\d>(.*?)</h\\d>").replaceAll("$1")'  
~/Desktop/cooluri.html
```

**TIP:** Never ever use the option `-i` without a backup extension.

Or to really get into groovy (literally)

```
find . -name \*.java | xargs groovy -p -i -e  
'(line =~ "@author James  
Strachan").replaceAll("@author Bobby  
Bonson")'
```

Additionally you have access to the line number in the current file you are reading via the variable `count`. This can be used for a number of convenient groovy one-liners.

Let us assume you want to prefix every line in a file with the line number. Doing this requires next to no work in Groovy (we additionally create a copy of the original file with the extension `.bak`).

```
groovy -pi .bak -e "count + ': ' + line"
```

Or let us create a grep-like command that prints the line number where it found matching strings for a regular expression.

```
groovy -p -e "if(line =~ /groovy/)count + ':  
' + line"
```

Print the first 50 lines of all files:

```
groovy -p -e "if(count < 50) line"
```

until one file is longer than 50 lines:

```
groovy -p -e "if(count >=
50)System.exit(0);line"
```

Add a Groovy-Shebang (the string `#!/usr/bin/groovy`) to all Groovy files:

```
groovy -i .bak -pe "if(count == 1) println
'#!/usr/bin/groovy'" *.groovy
```

Another very convenient option is `-a`, which splits the current input line into the array `split`. By default the split pattern is `" "` (one space). The option `-a` optionally takes another split pattern which is then used instead.

Print processes owned by root:

```
ps aux|groovy -a -ne "if(split[0] =~
'root')println split[10..-1]"
```

Print all logins from `/etc/passwd` that are not commented:

```
groovy -a':' -ne "if(!(split[0] =~
/^#/))println split[0]" /etc/passwd
```

Add the first and the penultimate column of a file:

```
groovy -a -pe
"split[0].toInteger()+split[-2].toInteger()"
accounts.txt
```

For more examples or inspiration browse through the search results for [Perl One Liners](#)

### listen mode

Another groovy command line option is the ability to startup groovy in listen mode, which will attach groovy to a TCP port on your machine (`-l <port>` with a default port of 1960).

For each connection that is made to this port, groovy executes the supplied script on a line by line basis.

This oneliner will reverse every line that is thrown at it, try telnet to your machine on port 1960 to interact with this script.

```
groovy -l -e "println line.reverse()"
```

you can combine the -p option from earlier, to automatically print the result of your script

The following one liner is equivalent to the one liner immediately above.

```
groovy -l -p -e "line.reverse()"
```

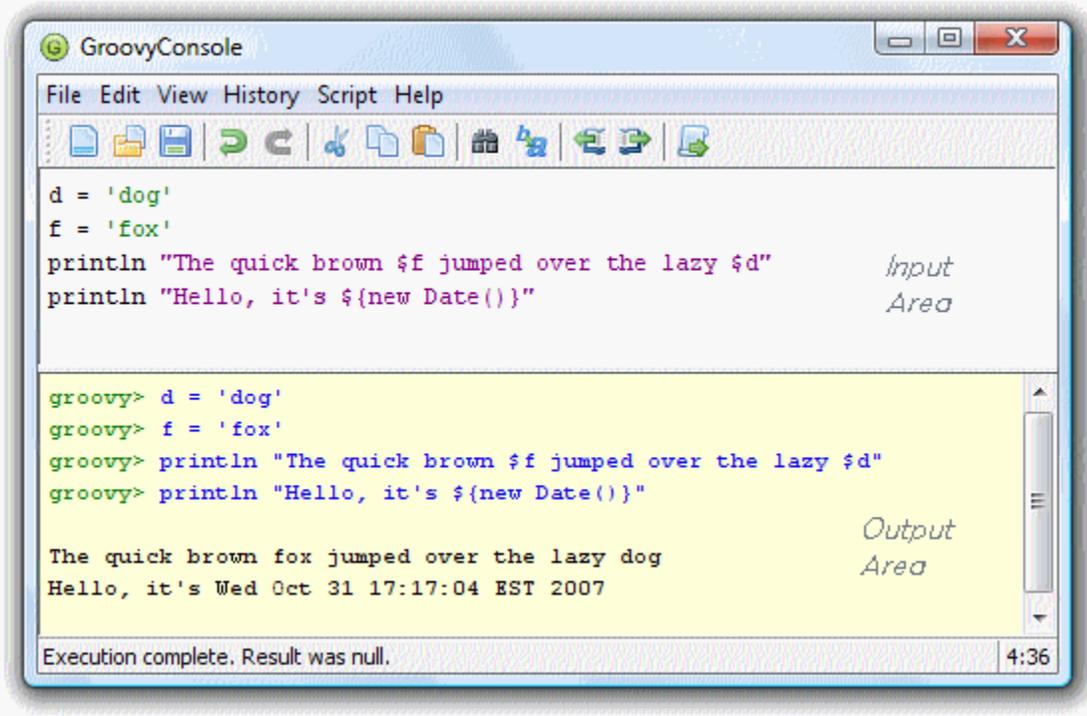
More examples of useful command line scripts in [SVN](#)

## Groovy Console

The Groovy Swing Console allows a user to enter and run Groovy scripts. This page documents the features of this user interface.

### Basics

The Groovy Console:



1. Groovy Console is launched via "groovyConsole" or "groovyConsole.bat", both located in "\$GROOVY\_HOME/bin"
2. The Console has an input area and an output area.
3. You type a Groovy script in the input area.
4. When you select "Run" from the "Actions" menu, the console compiles the script and runs it.
5. Anything that would normally be printed on System.out is printed in the output area.
6. If the script returns a non-null result, that result is printed.

## Features

### Running Scripts

Handy tips for running scripts:

- Ctrl+Enter and Ctrl+R are both shortcut keys for "Run Script".
- If you highlight just part of the text in the input area, then Groovy runs just that text.
- The result of a script is the value of the last expression executed.
- You can turn the System.out capture on and off by selecting "Capture System.out" from the "Actions" menu

### Editing Files

You can open any text file, edit it, run it (as a Groovy Script) and then save it again when you are finished.

- Select File -> Open (shortcut key ctrl+O) to open a file
- Select File -> Save (shortcut key ctrl+S) to save a file
- Select File -> New File (shortcut key ctrl+Q) to start again with a blank input area

### History and results

- You can pop-up a gui inspector on the last (non-null) result by selecting "Inspect Last" from the "Actions" menu. The inspector is a convenient way to view lists and maps.
- The console remembers the last ten script runs. You can scroll back and forth through the history by selecting "Next" and "Previous" from the "Edit" menu. Ctrl-N and ctrl-P are convenient shortcut keys.
- The last (non-null) result is bound to a variable named '\_' (an underscore).
- The last result (null and non-null) for every run in the history is bound into a list variable named '\_\_\_' (two underscores). The result of the last run is `__[-1]`, the result of the second to last run is `___[-2]` and so forth.

### And more

- You can attempt to interrupt a long running task by clicking the "interrupt" button on the small dialog box that pops up when a script is executing.
- You can change the font size by selecting "Smaller Font" or "Larger Font" from the "Actions menu"
- The console can be run as an Applet thanks to `groovy.ui.ConsoleApplet`
- Code is auto indented when you hit return
- You can drag'n drop a Groovy script over the text area to open a file
- You can modify the classpath with which the script in the console is being run by adding a new JAR or a directory to the classpath from the Script menu
- Error hyperlinking from the output area when a compilation error is expected or when an exception is thrown

### Embedding the Console

To embed a Swing console in your application, simply create the Console object, load some variables, and then launch it. The console can be embedded in either Java or Groovy code.

The Java code for this is:

```
import groovy.ui.Console;

...

Console console = new Console();
console.setVariable("var1",
getValueOfVar1());
console.setVariable("var2",
getValueOfVar2());
console.run();

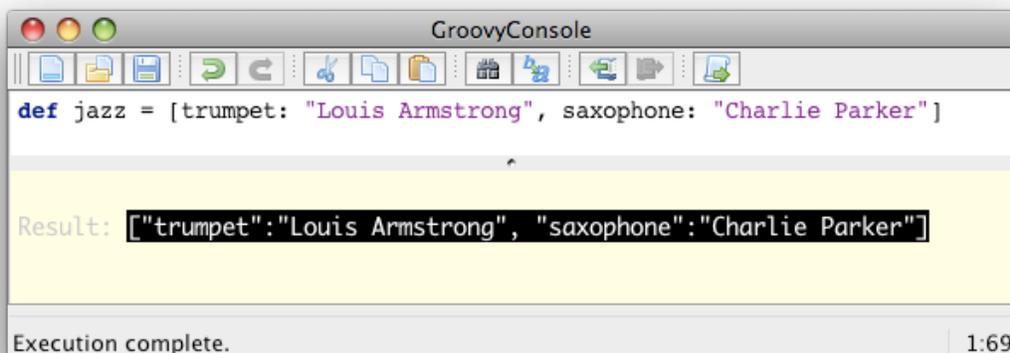
...
```

Once the console is launched, you can use the variable values in Groovy code.

An example of how to embed either the GroovyConsole or GroovyShell in a Spring Web application can be found at [Embedding a Groovy Console in a Java Server Application](#)

### Visualizing script output results

You can customize the way script output results are visualized. Let's see how we can customize this. For example, viewing a map result would show something like this:



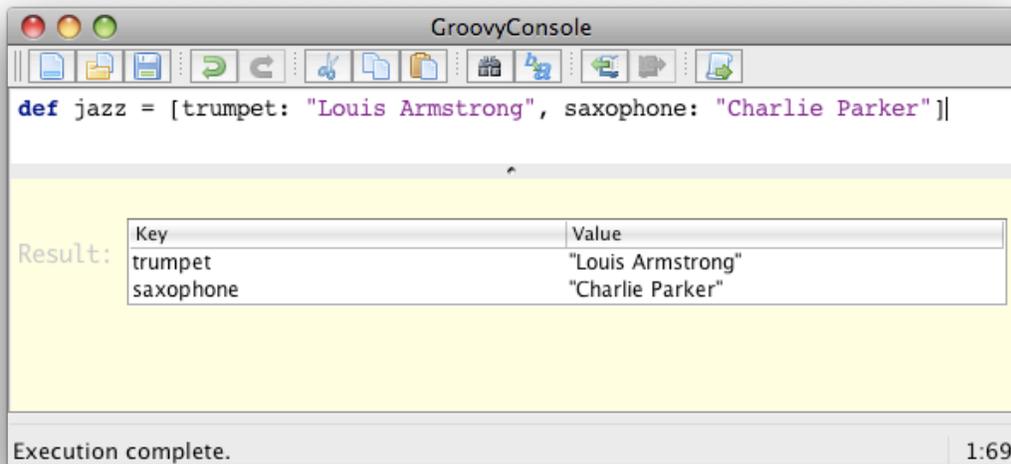
What you see here is the usual textual representation of a Map. But, what if we enabled custom visualization of certain results? The Swing console allows you to do just that. First of all, you have to ensure that the visualization

option is ticked: View -> Visualize Script Results — for the record, all settings of the Groovy Console are stored and remembered thanks to the Preference API. There are a few result visualizations built-in: if the script returns a `java.awt.Image`, a `javax.swing.Icon`, or a `java.awt.Component` with no parent, the object is displayed instead of its `toString()` representation. Otherwise, everything else is still just represented as text. Now, create the following Groovy script in `~/groovy/OutputTransforms.groovy`:

```
import javax.swing.*

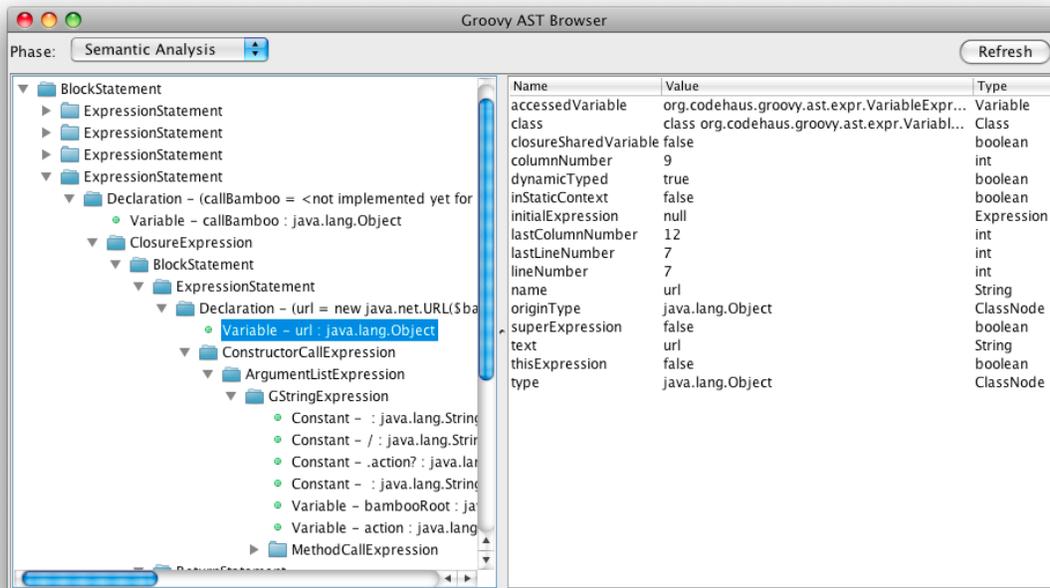
transforms << { result ->
    if (result instanceof Map) {
        def table = new JTable(
            result.collect{ k, v ->
                [k, v?.inspect()] as
Object[]
            } as Object[][],
            ['Key', 'Value'] as Object[])
        table.preferredViewportSize =
table.preferredSize
        return new JScrollPane(table)
    }
}
```

The Groovy Swing console will execute that script on startup, injecting a transforms list in the binding of the script, so that you can add your own script results representations. In our case, we transform the Map into a nice-looking Swing JTable. And we're now able to visualize maps in a friendly and attractive fashion, as the screenshot below shows:



## AST browser

Groovy Console can visualize the AST (Abstract Syntax Tree) representing the currently edited script, as shown by the screenshot below. This is particularly handy when you want to develop [AST transformations](#).



## Groovy Math

Groovy supports access to all Java math classes and operations. However, in order to make scripting math operations as intuitive as possible to the end user, the groovy math model supports a 'least surprising' approach to literal math operations for script programmers. To do this, groovy uses exact, or decimal math for default calculations.

This means that user computations like:

$$1.1 + 0.1 == 1.2$$

will return true rather than false (using float or double types in Java returns a result of 1.2000000000000002).

### Numeric literals

To support the 'least surprising' approach, groovy literals with decimal points are instantiated as *java.math.BigDecimal* types rather than binary floating point types (Float, Double). Float and Double types can of course be created explicitly or via the use of a suffix (see table below). Exponential notation is supported for decimal types (BigDecimal, Double Float) with or without a signed exponent (1.23e-23). Hexadecimal and octal literals are also supported. Hexadecimal numbers are specified in the typical format of "0x" followed by hex digits (e.g. 0x77).

Integral numeric literals (those without a decimal point) which begin with a 0 are treated as octal. Both octal and hexadecimal literals may have an integral suffix (G,L,I). Integral numeric literals without a suffix will be the smallest type into which the value will fit (Integer, Long, or BigInteger). See the numeric literal grammar at the end of this page for more details on syntax.

Type	Suffix
BigInteger	G
Long	L
Integer	I
BigDecimal	G
Double	D
Float	F

Examples:

```

assert 42I == new Integer("42")
assert 123L == new Long("123")
assert 2147483648 == new Long("2147483648")
//Long type used, value too large for an
Integer
assert 456G == new
java.math.BigInteger("456")
assert 123.45 == new
java.math.BigDecimal("123.45") //default
BigDecimal type used
assert 1.200065D == new Double("1.200065")
assert 1.234F == new Float("1.234")
assert 1.23E23D == new Double("1.23E23")

```

## Math operations

While the default behavior is to use decimal math, no attempt is made to preserve this if a binary floating point number is introduced into an expression (i.e. groovy never automatically promotes a binary floating point number to a BigDecimal). This is done for two reasons: First, doing so would imply a level of exactness to a result that is not guaranteed to be exact, and secondly, performance is slightly better under binary floating point math, so once it is introduced it is kept.

Finally, Groovy's math implementation is as close as practical to the Java 1.5 BigDecimal math model which implements precision based floating point decimal math (ANSI X3.274-1996 and ANSI X3.274-1996/AM 1-2000 (section 7.4)).

Therefore, binary operations involving subclasses of java.lang.Number automatically convert their arguments according to the following matrix (except for division, which is discussed below).

	<b>BigDecimal</b>	<b>BigInteger</b>	<b>Double</b>	<b>Float</b>	<b>Long</b>	<b>Integer</b>
<b>BigDecimal</b>	BigDecimal	BigDecimal	Double	Double	BigDecimal	BigDecimal
<b>BigInteger</b>	BigDecimal	BigInteger	Double	Double	BigInteger	BigInteger
<b>Double</b>	Double	Double	Double	Double	Double	Double
<b>Float</b>	Double	Double	Double	Double	Double	Double

<b>Long</b>	BigDecimal	BigInteger	Double	Double	Long	Long
<b>Integer</b>	BigDecimal	BigInteger	Double	Double	Long	Integer

Note - Byte, Character, and Short arguments are considered to be Integer types for the purposes of this matrix.

### **Division**

The division operators "/" and "/"= produce a Double result if either operand is either Float or Double and a BigDecimal result otherwise (both operands are any combination of Integer, Long, BigInteger, or BigDecimal). BigDecimal Division is performed as follows:

```
BigDecimal.divide(BigDecimal right, <scale>,
BigDecimal.ROUND_HALF_UP)
```

where <scale> is MAX(this.scale(), right.scale(), 10). Finally, the resulting BigDecimal is normalized (trailing zeros are removed).

For example:

```
1/2 == new java.math.BigDecimal("0.5")
1/3 == new
java.math.BigDecimal("0.3333333333")
2/3 == new
java.math.BigDecimal("0.6666666667")
```

Integer division can be performed on the integral types by casting the result of the division. For example:

```
assert (int)(3/2) == 1I
```

Future versions of Groovy may support an integer division operator such as div and/or ÷.

### **Power Operator**

Since groovy 1.0 beta 10 release, the power operator "\*\*" is supported for math calculation. For example, 5\*\*3 equals to Math.pow(5,3).

Java code:

```
// y = 2 x^3 + 5 x^2 - 3 x + 2
double x = 5.0;
double y = 2.0 * Math.pow(x,3) + 5.0 *
Math.pow(x,2) - 3.0*x + 2.0;
```

Groovy code:

```
// y = 2 x^3 + 5 x^2 - 3 x + 2
def x = 5.0
def y = 2.0*x**3 + 5.0*x**2 - 3.0*x + 2.0
```

#### More In-depth Information

Groovy and Java Math is explained in more depth in these pages:

[Integer Math](#)

[Decimal Math](#)

[Floating Point Math](#)

#### Numeric literal grammar

##### **IntegerLiteral:**

**Base10IntegerLiteral**

**HexIntegerLiteral**

**OctalIntegerLiteral**

##### **Base10IntegerLiteral:**

**Base10Numeral IntegerTypeSuffix (optional)**

##### **HexIntegerLiteral:**

**HexNumeral IntegerTypeSuffix (optional)**

**OctalIntegerLiteral:**

**OctalNumeral IntegerTypeSuffix (optional)**

**IntegerTypeSuffix: one of**

**i I l L g G**

**Base10Numeral:**

**0**

**NonZeroDigit Digits (optional)**

**Digits:**

**Digit**

**Digits Digit**

**Digit:**

**0**

**NonZeroDigit**

**NonZeroDigit: one of**

**\1 2 3 4 5 6 7 8 9**

**HexNumeral:**

**0 x HexDigits**

**0 X HexDigits**

**HexDigits:**

**HexDigit**

**HexDigit HexDigits**

**HexDigit: one of**

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

**OctalNumeral:**

0 OctalDigits

**OctalDigits:**

OctalDigit

OctalDigit OctalDigits

**OctalDigit: one of**

0 1 2 3 4 5 6 7

**DecimalPointLiteral:**

Digits . Digits ExponentPart (optional)

DecimalTypeSuffix (optional)

Digits ExponentPart DecimalTypeSuffix  
(optional)

Digits ExponentPart (optional)

DecimalTypeSuffix (optional)

**ExponentPart:**

ExponentIndicator SignedInteger

**ExponentIndicator: one of**

e E

**SignedInteger:**

Signopt Digits

**Sign: one of**

+ -

# DecimalTypeSuffix: one of f F d D g G

## Groovy Shell

### Groovy Shell

The Groovy Shell, aka. `groovysh` is a command-line application which allows easy access to evaluate Groovy expressions, define classes and run simple experiments.

- [Features](#)
- [Command-line Options and Arguments](#)
- [Evaluating Expressions](#)
  - [Simple Expressions](#)
  - [Evaluation Result](#)
  - [Multi-line Expressions](#)
  - [Variables](#)
  - [Functions](#)
- [Commands](#)
  - [Recognized Commands](#)
- [Preferences](#)
  - [Recognized Preferences](#)
  - [Listing Preferences](#)
  - [Clearing Preferences \(ie. Resetting to Defaults\)](#)
- [User Profile Scripts and State](#)
  - [Profile Scripts](#)
  - [State](#)
- [Screen Shots](#)
- [Troubleshooting](#)
  - [Platform Problems](#)

### Features

- No need for `go` command to execute buffer.
- Rich cross-platform edit-line editing, history and completion thanks to [JLine](#).
- ANSI colors (prompt, exception traces, etc).
- Simple, yet robust, command system with online help, user alias support and more.
- User profile support

### Command-line Options and Arguments

The shell supports several options to control verbosity, ANSI coloring and other features.

```

./bin/groovysh --help
usage: groovysh [options] [...]
  -C, --color[=FLAG]          Enable or
disable use of ANSI colors
  -D, --define=NAME=VALUE    Define a system
property
  -T, --terminal=TYPE        Specify the
terminal TYPE to use
  -V, --version              Display the
version
  -d, --debug                Enable debug
output
  -h, --help                 Display this
help message
  -q, --quiet                Suppress
superfluous output
  -v, --verbose              Enable verbose
output

```

In addition to options, commands or expressions can be given on the command-line which will invoke the shell in non-interactive mode. The commands or expressions will be evaluated and the shell will exit. If no additional arguments are given the shell will startup interactively.

*Execute a Command*

```

./bin/groovysh 'show preferences'
No preferences are set

```

*Evaluate an Expression*

```
./bin/groovysh 'System.properties.each { k,  
v -> println("$k = $v") }'  
java.runtime.name = Java(TM) 2 Runtime  
Environment, Standard Edition  
sun.boot.library.path =  
/System/Library/Frameworks/JavaVM.framework/  
Versions/1.5.0/Libraries  
java.vm.version = 1.5.0_07-87  
awt.nativeDoubleBuffering = true  
gopherProxySet = false  
...
```

## Evaluating Expressions

### *Simple Expressions*

```
println "Hello"
```

### *Evaluation Result*

When a complete expression is found, it is compiled and evaluated. The result of the evaluation is stored into the `_` variable.

### *Multi-line Expressions*

Multi-line/complex expressions (like closure or class definitions) may be defined over several lines. When the shell detects that it has a complete expression it will compile and evaluate it.

### *Define a Class*

```
class Foo {
    def bar() {
        println "baz"
    }
}
```

#### *Use the Class*

```
foo = new Foo()
foo.bar()
```

#### *Variables*

Shell variables are **all** untyped (ie. no `def` or other type information).

This **will** set a shell variable:

```
foo = "bar"
```

But, this will evaluate a local variable and will **not** be saved to the shell's environment:

```
def foo = "bar"
```

#### *Functions*

Functions can be defined in the shell, and will be saved for later use.

Defining a function is easy:

```
groovy:000> def hello(name) {
groovy:001> println("Hello $name")
groovy:002> }
```

And then using it is as one might expect:

```
hello("Jason")
```



Internally the shell creates a closure to encapsulate the function and then binds the closure to a variable. So variables and functions share the same namespace.

## Commands

The shell has a number of different commands, which provide rich access to the shell's environment.

Commands all have a *name* and a *shortcut* (which is something like `\h`). Commands may also have some predefined system *aliases*. Users may also create their own aliases.

### Recognized Commands

`help`

Display the list of commands (and aliases) or the help text for specific command.

### The Command List

```
groovy:000> help
```

```
For information about Groovy, visit:  
http://groovy.codehaus.org
```

### Available commands:

```
help      (\h ) Display this help message  
?         (\? ) Alias to: help  
exit     (\x ) Exit the shell  
quit     (\q ) Alias to: exit  
import   (\i ) Import a class into the  
namespace  
display  (\d ) Display the current buffer  
clear    (\c ) Clear the buffer  
show     (\S ) Show variables, classes or  
imports
```

**inspect** (\n ) Inspect a variable or the last result with the GUI object browser

**purge** (\p ) Purge variables, classes, imports or buffers

**edit** (\e ) Edit the current buffer

**load** (\l ) Load a file or URL into the buffer

**.** (\. ) Alias to: load

**save** (\s ) Save the current buffer to a file

**record** (\r ) Record the current session to a file

**history** (\H ) Display, manage and recall edit-line history

**alias** (\a ) Create an alias

**set** (\= ) Set (or list) preferences

```
For help on a specific command type:  
help <command>
```

#### *Help for a Command*

While in the interactive shell, you can ask for help for any command to get more details about its syntax or function. Here is an example of what happens when you ask for help for the `help` command:

```
groovy:000> help help  
  
usage: help [<command>]  
  
Display the list of commands or the help  
text for <command>.
```

#### **exit**

Exit the shell.

This is the **only** way to exit the shell. Well, you can still `CTRL-C`, but the shell will complain about an abnormal shutdown of the JVM.

#### **import**

Add a custom import which will be included for all shell evaluations.

This command can be given at any time to add new imports.

#### **display**

Display the contents of the current buffer.

This only displays the buffer of an incomplete expression. Once the expression is complete, the buffer is reset. The prompt will update to show the size of the current buffer as well.

#### *Example*

```
groovy:000> class Foo {
groovy:001> def bar
groovy:002> def baz() {
groovy:003> display
  001> class Foo {
  002> def bar
  003> def baz() {
```

**clear**

Clears the current buffer, resetting the prompt counter to 000. Can be used to recover from compilation errors.

**show**

Show variables, classes or preferences or imports.

*show variables*

```
groovy:000> show variables
Variables:
  _ = true
```

*show classes*

*show imports*

*show preferences*

*show all*

**inspect**

Opens the GUI object browser to inspect a variable or the result of the last evaluation.

**purge**

Purges objects from the shell.

*purge variables*

*purge classes*

*purge imports*

*purge preferences*

*purge all*

*edit*

Edit the current buffer in an external editor.

Currently only works on UNIX systems which have the `EDITOR` environment variable set, or have configured the `editor` preference.

*load*

Load one or more files (or urls) into the buffer.

*save*

Saves the buffer's contents to a file.

*record*

Record the current session to a file.

*record start*

*record stop*

*record status*

*history*

Display, manage and recall edit-line history.

*history show*

*history recall*

*history flush*

*history clear*

*alias*

Create an alias.

*set*

Set or list preferences.

## **Preferences**

Some of aspects of `groovysh` behaviors can be customized by setting preferences. Preferences are set using the `set` command or the `\=` shortcut.

### ***Recognized Preferences***

*verbosity*

Set the shell's verbosity level. Expected to be one of:

- DEBUG
- VERBOSE
- INFO
- QUIET

Default is `INFO`.

If this preference is set to an invalid value, then the previous setting will be used, or if there is none, then the preference is removed and the default is used.

**show-last-result**

Show the last result after an execution.

Default is `true`.

**sanitize-stack-trace**

Sanitize (trim-down/filter) stack traces.

Default is `true`.

**editor**

Configures the editor used by the `edit` command.

Default is the value of the system environment variable `EDITOR`.

#### Mac OS X

To use `TextEdit`, the default text editor on Mac OS X, configure:

```
set editor /Applications/TextEdit.app/Contents/MacOS/TextEdit
```

#### Setting a Preference

```
set verbosity DEBUG
```

#### Listing Preferences

To list the current `set` preferences (and their values):

```
show preferences
```

### **Limitation**

At the moment, there is no way to list all of the known/available preferences to be set.

#### *Clearing Preferences (ie. Resetting to Defaults)*

```
purge preferences
```

### **User Profile Scripts and State**

#### *Profile Scripts*

```
$HOME/.groovy/groovysh.profile
```

This script, if it exists, is loaded when the shell starts up.

```
$HOME/.groovy/groovysh.rc
```

This script, if it exists, is loaded when the shell enters interactive mode.

#### *State*

```
$HOME/.groovy/groovysh.history
```

Edit-line history is stored in this file.

### **Screen Shots**

These shots have been taken over the development of the new shell, so some of the content might look slightly different. Also, note the yellow colors here are the shell's *bold* color, so the colors might look different depending on how the enclosing shell has its colors setup.

```
bliss:~/ws/groovy/groovy-core/tmp/groovy-core jason$ NEWSHELL=t ./target/install/bin/groovysh
Groovy Shell (1.1-beta-3-SNAPSHOT, JVM: 1.5.0_07-87)
Type 'help' or '\h' for help.
-----
groovy:(0):000> 
```

```
bliss:~/ws/groovy/groovy-core/tmp/groovy-core jason$ NEWSHELL=t ./target/install/bin/groovysh
Groovy Shell (1.1-beta-3-SNAPSHOT, JVM: 1.5.0_07-87)
Type 'help' or '\h' for help.
-----
groovy:(0):000> help

For information about Groovy, visit:
http://groovy.codehaus.org

Available commands:
help      (\h) Display this help message
?         (? ) Alias to: help
exit      (\x) Exit the shell
quit      (\q) Alias to: exit
import    (\i) Import a class into the namespace
display   (\d) Display the current buffer
clear     (\c) Clear the buffer
show      (\s) Show variables, classes or imports
inspect   (\n) Inspect a variable or the last result with the GUI object browser
purge     (\p) Purge variables, classes, imports or buffers
edit      (\e) Edit the current buffer
load      (\l) Load a file or URL into the buffer
.         (.) Alias to: load
save      (\s) Save the current buffer to a file
buffer    (\b) Manage shell buffers
#         (#) Alias to: buffer
history   (\H) Show edit-line history
alias     (\a) Create an alias
s         (\a0) User defined alias to: shadow

For help on a specific command type:
help <command>

groovy:(0):000> 
```

```
Default (108,32)
Type 'help' or '\h' for help.
-----
groovy:(0):000> help show
usage: show [<variables|classes|imports|all>]
Show variables, classes or imports...
groovy:(0):000> help buffer
usage: buffer [+|-|?|<n>]
Manages the shell's buffers. With no argument displays the currently selected buffer.
+ Create a new buffer and select it
- Delete the current buffer
? Display information about the buffers
<n> Select buffer <n>
groovy:(0):000> help edit
usage: edit
Edit the current buffer.
groovy:(0):000> help purge
usage: purge [<variables|classes|imports|buffers|all>]+
Purges objects from the shell...
groovy:(0):000> █
```

## Troubleshooting

Please [report](#) any problems you run into. Please be sure to mark the JIRA issue with the Groovysh component.

### *Platform Problems*

#### Problems loading the JLine DLL

On Windows, [JLine](#) (which is used for the fancy shell input/history/completion fluff), uses a **tiny** DLL file to trick the **evil** Windows faux-shell (CMD.EXE or COMMAND.COM) into providing Java with unbuffered input. In some rare cases, this might fail to load or initialize.

One solution is to disable the frills and use the unsupported terminal instance. You can do that on the command-line using the `--terminal` flag and set it to one of:

- none
- false
- off
- `jline.UnsupportedTerminal`

```
groovysh --terminal=none
```

#### Problems with Cygwin on Windows

Some people have issues when running groovysh with cygwin. If you have troubles, the following may help:

```
stty -icanon min 1 -echo
groovysh --terminal=unix
stty icanon echo
```

wiki

## Groovy Utils

### Groovy Utils

This page documents the utility classes available in the `groovy.utils.*` package.

1. [ConfigSlurper](#)
2. [ObjectGraphBuilder](#)
3. [ObservableMap](#)

### ConfigSlurper

#### ConfigSlurper

[ConfigSlurper](#) is a utility class within Groovy for writing properties file like scripts for performing configuration. Unlike regular Java properties files ConfigSlurper scripts support native Java types and are structured like a tree.

Below is an example of how you could configure Log4j with a ConfigSlurper script:

```
log4j.appender.stdout =
"org.apache.log4j.ConsoleAppender"
log4j.appender."stdout.layout"="org.apache.l
og4j.PatternLayout"
log4j.rootLogger="error, stdout "
log4j.logger.org.springframework="info, stdou
t"
log4j.additivity.org.springframework=false
```

To load this into a readable config you can do:

```

def config = new ConfigSlurper().parse(new
File('myconfig.groovy').toURL())

assert "info,stdout" ==
config.log4j.logger.org.springframework
assert false ==
config.log4j.additivity.org.springframework

```

As you can see from the example above you can navigate the config using dot notation and the return values are Java types like strings and booleans.

You can also use scoping in config scripts to avoid repeating yourself. So the above config could also be written as:

```

log4j {
    appender.stdout =
"org.apache.log4j.ConsoleAppender"

appender."stdout.layout"="org.apache.log4j.P
atternLayout"
    rootLogger="error,stdout"
    logger {
        org.springframework="info,stdout"
    }
    additivity {
        org.springframework=false
    }
}

```

#### *Converting to and from Java properties files*

You can convert ConfigSlurper configs to and from Java properties files. For example:

```
java.util.Properties props = // load from  
somewhere
```

```
def config = new  
ConfigSlurper().parse(props)
```

```
props = config.toProperties()
```

### *Merging configurations*

You can merge config objects so if you have multiple config files and want to create one central config object you can do:

```
def config1 = new ConfigSlurper().parse(..)  
def config2 = new ConfigSlurper().parse(..)
```

```
config1 = config1.merge(config2)
```

### *Serializing a configuration to disk*

You can serialize a config object to disk. Each config object implements the `groovy.lang.Writable` interface that allows you to write out the config to any `java.io.Writer`:

```
def config = new ConfigSlurper().parse(..)
```

```
new File("..").withWriter { writer ->  
    config.writeTo(writer)
```

```
}
```

### *Special "environments" Configuration*

The `ConfigSlurper` class has a special constructor other than the default constructor that takes an "environment" parameter. This special constructor works in concert with a property setting called `environments`. This allows a default setting to exist in the property file that can be superceded by a setting in the appropriate environments

closure. This allows multiple related configurations to be stored in the same file.

Given this groovy property file:

## Sample.groovy

```
sample {
    foo = "default_foo"
    bar = "default_bar"
}

environments {
    development {
        sample {
            foo = "dev_foo"
        }
    }
    test {
        sample {
            bar = "test_bar"
        }
    }
}
```

Here is the demo code that exercises this configuration:

```
def config = new
ConfigSlurper("development").parse(new
File('Sample.groovy').toURL())

assert config.sample.foo == "dev_foo"
assert config.sample.bar == "default_bar"

config = new ConfigSlurper("test").parse(new
File('Sample.groovy').toURL())

assert config.sample.foo == "default_foo"
assert config.sample.bar == "test_bar"
```

Note: the environments closure is not directly parsable. Without using the special environment constructor the closure is ignored.

The value of the environment constructor is also available in the configuration file, allowing you to build the configuration like this:

```
switch (environment) {
  case 'development':
    baseUrl = "devServer/"
    break
  case 'test':
    baseUrl = "testServer/"
    break
  default:
    baseUrl = "localhost/"
}
```

Further information

## ObjectGraphBuilder

ObjectGraphBuilder is a builder for an arbitrary graph of beans that follow the JavaBean convention, its useful for creating test data for example.

Let's say that the following classes belong into your domain model:

```
package com.acme

class Company {
    String name
    Address address
    List employees = []
}

class Address {
    String line1
    String line2
    int zip
    String state
}

class Employee {
    String name
    int employeeId
    Address address
    Company company
}
```

With ObjectGraphBuilder building a Company with three employees is as easy as

```

def builder = new ObjectGraphBuilder()
// uncomment the following line if running
this script with GroovyConsole
//builder.classloader =
getClass().classLoader
builder.classNameResolver = "com.acme"

def acme = builder.company( name: 'ACME' ){
    3.times {
        employee( id: it.toString(), name:
'Drone ${it}' )
    }
}

assert acme != null
assert acme.employees.size() == 3

```

Here is what's happening behind the scenes:

1. the builder will try to match a node name into a Class, using a default **ClassNameResolver** strategy that requires a package name.
2. then an instance of said class must be created, using a default **NewInstanceResolver** strategy that calls a no-args constructor.
3. the parent/child relationship must be resolved for nested nodes, here it gets a little tricky as two other strategies come into play. **RelationNameResolver** will yield the name of the child property in the parent, and the name of the parent property in the child (if any, in this case, Employee has a parent property aptly named 'company'). **ChildPropertySetter** will 'insert' the child into the parent taking into account if the child belongs to a Collection or not (in this case employees should be a list of Employee instances in Company).

All 4 strategies have a default implementation that work as expected if the code follows the usual conventions for writing JavaBeans. But if by any chance any of your beans does not follow the convention you may plug your own implementation of each strategy. Each strategy setter is Closure friendly, for example

```

builder.newInstanceResolver = { klass,
attributes ->
    if( attributes.foo ){
        return klass.newInstance(
[attributes.foo] as Object[] )
    }
    // default no-args constructor
    klass.newInstance()
}

```

ObjectGraphBuilder supports ids per node as SwingBuilder does, meaning that you can 'store' a reference to a node in the builder, this is useful to relate one instance with many others as well. Because a property named 'id' may be of business meaning in some domain models ObjectGraphBuilder has a strategy named **IdentifierResolver** that you may configure to change the default name value ('id'). The same may happen with the property used for referencing a previously saved instance, a strategy named **ReferenceResolver** will yield the appropriate value (default is 'refId'):

```

def company = builder.company( name: 'ACME'
) {
    address( id: 'a1', line1: '123 Groovy
Rd', zip: 12345, state: 'JV' )
    employee( name: 'Duke', employeeId: 1,
address: a1 )
}

```

```
def company = builder.company( name: 'ACME'
) {
    address( id: 'a1', line1: '123 Groovy
Rd', zip: 12345, state: 'JV' )
    employee( name: 'Duke', employeeId: 1 ){
        address( refId: 'a1' )
    }
}
```

Its worth mentioning that you cannot modify the properties of a referenced bean.

For those rare occasions where ObjectGraphBuilder can't locate your classes (it happens when you run a script using groovyConsole) you may define a classLoader for ObjectGraphBuilder to resolve classes. Try for example running the following script inside groovyConsole and then comment out the classLoader property.

```
class Conference {
    String name
    List speakers = []
}

class Speaker {
    String name
}

def ogb = new ObjectGraphBuilder(
    classLoader: getClass().classLoader )
def j1 = ogb.conference( name: 'JavaOne' ) {
    speaker( name: 'Duke' )
}
assert j1.speakers.size() == 1
assert j1.speakers[0].name == 'Duke'
```

## ObservableMap

An observable map will trigger a `PropertyChangeEvent` every time a value changes. We can convert a map into an observable one with the `'as'` keyword too:

```

// don't forget the imports
import java.beans.*
def map = [:] as ObservableMap
map.addPropertyChangeListener({ evt ->
    println "${evt.propertyName}:
    ${evt.oldValue} -> ${evt.newValue}"
} as PropertyChangeListener)

map.key = 'value' // prints key: null ->
value
map.key = 'Groovy' // prints key: value ->
Groovy

```

We can also wrap an existing map with an ObservableMap

```

import java.beans.*
def sorted = [a:1,b:2] as TreeMap
def map = new ObservableMap(sorted)
map.addPropertyChangeListener({ evt ->
    println "${evt.propertyName}:
    ${evt.oldValue} -> ${evt.newValue}"
} as PropertyChangeListener)
map.key = 'value'
assert ['a', 'b', 'key'] == (sorted.keySet()
as List)
assert ['a', 'b', 'key'] == (map.keySet() as
List)

```

Lastly we can specify a closure as an additional parameter, it will work like a filter for properties that should or should not trigger a PropertyChangeEvent when their values change, this is useful in conjunction with Expando. The

filtering closure may take 2 parameters (the property name and its value) or less (the value of the property).

```
import java.beans.*
def map = new ObservableMap({!(it instanceof
Closure)})
map.addPropertyChangeListener({ evt ->
    println "${evt.propertyName}:
${evt.oldValue} -> ${evt.newValue}"
} as PropertyChangeListener)
def bean = new Expando( map )
bean.lang = 'Groovy' // prints lang: null
-> Groovy
bean.sayHello = { name -> "Hello ${name}" }
// prints nothing, event is skipped
assert 'Groovy' == bean.lang
assert 'Hello Groovy' ==
bean.sayHello(bean.lang)
```

## GUI Programming with Groovy

### *Introduction*

Groovy has several options available for writing GUI code:

- you can directly use any of the AWT and Swing classes built in to Java
- you can use any Java libraries which sit on top of or alongside Swing, e.g.:
  - JGoodies ([home](#)) ([example](#))
  - JFreeChart ([home](#)) ([example](#)) ([builder](#))
- you can use Groovy's SwingBuilder (see Further Details)
- you can use [GraphicsBuilder](#) to access Java 2D features
- you can use SwingXBuilder (see Further Details)
- you can use the [JIDE builder](#) for the open source [JIDE Common Layer](#)
- you can use the [GroovySWT](#) module
- you can use the [PrefuseBuilder](#) for creating pretty representations of graphs of objects
- you can use the [wingS](#) framework: [WingsBuilder](#) - ([example](#)) ([comparison to swing](#))

You might also consider using [FEST](#) for testing your GUI application as shown [here](#).

### *Further Details*

- [Swing Builder](#)
  - [Alphabetical Widgets List](#)

- [SwingBuilder.borderLayout](#)
- [SwingBuilder.action](#)
- [SwingBuilder.actions](#)
- [SwingBuilder.bind](#)
- [SwingBuilder.borderLayout](#)
- [SwingBuilder.boundedRangeModel](#)
- [SwingBuilder.box](#)
- [SwingBuilder.boxLayout](#)
- [SwingBuilder.button](#)
- [SwingBuilder.buttonGroup](#)
- [SwingBuilder.cardLayout](#)
- [SwingBuilder.checkBox](#)
- [SwingBuilder.checkBoxMenuItem](#)
- [SwingBuilder.closureColumn](#)
- [SwingBuilder.colorChooser](#)
- [SwingBuilder.comboBox](#)
- [SwingBuilder.compoundBorder](#)
- [SwingBuilder.container](#)
- [SwingBuilder.desktopPane](#)
- [SwingBuilder.dialog](#)
- [SwingBuilder.editorPane](#)
- [SwingBuilder.emptyBorder](#)
- [SwingBuilder.etchedBorder](#)
- [SwingBuilder.fileChooser](#)
- [SwingBuilder.flowLayout](#)
- [SwingBuilder.formattedTextField](#)
- [SwingBuilder.frame](#)
- [SwingBuilder.gridBagConstraints](#)
- [SwingBuilder.gridBagLayout](#)
- [SwingBuilder.gridLayout](#)
- [SwingBuilder.imageIcon](#)
- [SwingBuilder.internalFrame](#)
- [SwingBuilder.JComponent](#)
- [SwingBuilder.label](#)
- [SwingBuilder.layeredPane](#)
- [SwingBuilder.lineBorder](#)
- [SwingBuilder.list](#)
- [SwingBuilder.loweredBevelBorder](#)
- [SwingBuilder.map](#)
- [SwingBuilder.matteBorder](#)
- [SwingBuilder.menu](#)
- [SwingBuilder.menuBar](#)
- [SwingBuilder.menuItem](#)
- [SwingBuilder.optionPane](#)
- [SwingBuilder.overlayLayout](#)
- [SwingBuilder.panel](#)
- [SwingBuilder.passwordField](#)
- [SwingBuilder.popupMenu](#)
- [SwingBuilder.progressBar](#)
- [SwingBuilder.propertyColumn](#)
- [SwingBuilder.radioButton](#)
- [SwingBuilder.radioButtonMenuItem](#)
- [SwingBuilder.raisedBevelBorder](#)

- [SwingBuilder.raisedEtchedBorder](#)
- [SwingBuilder.scrollBar](#)
- [SwingBuilder.scrollPane](#)
- [SwingBuilder.separator](#)
- [SwingBuilder.slider](#)
- [SwingBuilder.spinner](#)
- [SwingBuilder.spinnerDateModel](#)
- [SwingBuilder.spinnerListModel](#)
- [SwingBuilder.spinnerNumberModel](#)
- [SwingBuilder.splitPane](#)
- [SwingBuilder.springLayout](#)
- [SwingBuilder.tabbedPane](#)
- [SwingBuilder.table](#)
- [SwingBuilder.tableColumn](#)
- [SwingBuilder.tableLayout](#)
- [SwingBuilder.tableModel](#)
- [SwingBuilder.td](#)
- [SwingBuilder.textArea](#)
- [SwingBuilder.textField](#)
- [SwingBuilder.textPane](#)
- [SwingBuilder.titledBorder](#)
- [SwingBuilder.toggleButton](#)
- [SwingBuilder.toolBar](#)
- [SwingBuilder.tr](#)
- [SwingBuilder.tree](#)
- [SwingBuilder.viewport](#)
- [SwingBuilder.widget](#)
- [SwingBuilder.window](#)
- [Categorical Widget List](#)
- [Extending Swing Builder](#)
- [Multithreading with SwingBuilder](#)
- [SwingXBuilder](#)
  - [Effects](#)
  - [Extending SwingXBuilder](#)
  - [Graphs](#)
  - [MultiSplitPane](#)
  - [Painters](#)
  - [Widgets and Common Attributes](#)

## Swing Builder

SwingBuilder allows you to create full-fledged Swing GUIs in a declarative and concise fashion. It accomplishes this by employing a common idiom in Groovy, builders. Builders handle the busywork of creating complex objects for you, such as instantiating children, calling Swing methods, and attaching these children to their parents. As a consequence, your code is much more readable and maintainable, while still allowing you access to the full range of Swing components.

Here's a simple example of using SwingBuilder:

```

import groovy.swing.SwingBuilder
import java.awt.BorderLayout as BL

count = 0
new SwingBuilder().edt {
    frame(title:'Frame', size:[300,300], show:
true) {
        BorderLayout()
        textlabel = label(text:"Click the
button!", constraints: BL.NORTH)
        button(text:'Click Me',
            actionPerformed: {count++;
textlabel.text = "Clicked ${count}
time(s)."; println "clicked"},
            constraints:BL.SOUTH)
    }
}

```

Here is what it will look like:



This hierarchy of components would normally be created through a series of repetitive instantiations, setters, and finally attaching this child to its respective parent. Using SwingBuilder, however, allows you to define this hierarchy in its native form, which makes the interface design understandable simply by reading the code.

The flexibility shown here is made possible by leveraging the many programming features built-in to Groovy, such as closures, implicit constructor calling, import aliasing, and string interpolation. Of course, these do not have to be fully understood in order to use SwingBuilder; as you can see from the code above, their uses are intuitive.

Here is a slightly more involved example, with an example of SwingBuilder code re-use via a closure.

```

import groovy.swing.SwingBuilder

```

```
import javax.swing.*
import java.awt.*

def swing = new SwingBuilder()

def sharedPanel = {
    swing.panel() {
        label("Shared Panel")
    }
}

count = 0
swing.edt {
    frame(title:'Frame',
defaultCloseOperation:JFrame.EXIT_ON_CLOSE,
pack:true, show:true) {
        vbox {
            textlabel = label("Click the
button!")
            button(
                text:'Click Me',
                actionPerformed: {
                    count++
                    textlabel.text =
"Clicked ${count} time(s)."
                    println "Clicked!"
                }
            )
            widget(sharedPanel())
            widget(sharedPanel())
        }
    }
}
```

```
    }  
}
```

Here's another variation that relies on observable beans and binding

```
import groovy.swing.SwingBuilder  
import groovy.beans.Bindable  
  
class MyModel {  
    @Bindable int count = 0  
}  
  
def model = new MyModel()  
new SwingBuilder().edt {  
    frame(title: "Java Frame", size: [100,  
100], locationRelativeTo: null, show: true)  
    {  
        GridLayout(cols: 1, rows: 2)  
        label(text: bind(source: model,  
sourceProperty: "count", converter: { v ->  
v? "Clicked $v times": ''}))  
        button("Click me!", actionPerformed: {  
model.count++ })  
    }  
}
```

[@Bindable](#) is one of the core [AST Transformations](#). It generates all the required boilerplate code to turn a simple bean into an observable one. The `bind()` node creates appropriate `PropertyChangeListener`s that will update the interested parties whenever a `PropertyChangeEvent` is fired.

Mailer User Interface example

```

import javax.swing.*
import
javax.swing.tree.DefaultMutableTreeNode as
TreeNode
import groovy.swing.SwingBuilder

mboxes = [
    [name: "root@example.com", folders:
    [[name: "Inbox"], [name: "Trash"]]],
    [name: "test@foo.com", folders: [[name:
    "Inbox"], [name: "Trash"]]]
]
def swing = new SwingBuilder()
JTree mboxTree
swing.frame(title: 'Mailer',
defaultCloseOperation:
JFrame.DISPOSE_ON_CLOSE,
    size: [800, 600], show: true,
locationRelativeTo: null) {
    lookAndFeel("system")
    menuBar() {
        menu(text: "File", mnemonic: 'F') {
            menuItem(text: "Exit", mnemonic:
'X', actionPerformed: {dispose() })
        }
    }
    splitPane {
        scrollPane(constraints: "left",
preferredSize: [160, -1]) {
            mboxTree = tree(rootVisible:
false)

```

```

    }

    splitPane(orientation:JSplitPane.VERTICAL_SPLIT, dividerLocation:280) {
        scrollPane(constraints: "top") {
    mailTable = table() }
        scrollPane(constraints:
"bottom") { textArea() }
    }
    ["From", "Date", "Subject"].each {
    mailTable.model.addColumn(it) }
}

mboxTree.model.root.removeAllChildren()
mboxes.each {mbox ->
    def node = new TreeNode(mbox.name)
    mbox.folders.each { folder ->
node.add(new TreeNode(folder.name)) }
}

```

```

mboxTree.model.root.add(node)
}
mboxTree.model.reload(mboxTree.model.root)

```

- [Alphabetical Widgets List](#)
- [Categorical Widget List](#)
- [Extending Swing Builder](#)
- [Multithreading with SwingBuilder](#)

#### **@Bindable and @Vetoable AST transformations**

Although not specific to SwingBuilder, Groovy 1.6 introduced two [Bindable and Vetoable transformation](#) of interest to Swing developers.

#### **Alphabetical Widgets List**

Element	Principal Java Class	Notes
<a href="#">action</a>	javax.swing.Action	
<a href="#">actions</a>	java.util.Collection	
<a href="#">bind</a>	org.codehaus.groovy.binding.FullBinding	used to bind attribute to other bean properties
<a href="#">borderLayout</a>	java.awt.BorderLayout	
<a href="#">boundedRangeModel</a>	javax.swing.DefaultBoundedRangeModel	
<a href="#">box</a>	javax.swing.Box	
<a href="#">boxLayout</a>	javax.swing.BoxLayout	
<a href="#">button</a>	javax.swing.JButton	
<a href="#">buttonGroup</a>	javax.swing.ButtonGroup	
<a href="#">cardLayout</a>	java.awt.CardLayout	
<a href="#">checkBox</a>	javax.swing.JCheckBox	
<a href="#">checkBoxMenuItem</a>	javax.swing.JCheckBoxMenuItem	
<a href="#">closureColumn</a>	groovy.model.DefaultTableColumn	

<a href="#"><u>compoundBorder</u></a>	javax.swing.border.CompoundBorder	
<a href="#"><u>colorChooser</u></a>	javax.swing.JColorChooser	
<a href="#"><u>comboBox</u></a>	javax.swing.JComboBox	
<a href="#"><u>container</u></a>		placeholder for external container widget
<a href="#"><u>desktopPane</u></a>	javax.swing.JDesktopPane	
<a href="#"><u>dialog</u></a>	javax.swing.JDialog	
<a href="#"><u>editorPane</u></a>	javax.swing.JEditorPane	
<a href="#"><u>emptyBorder</u></a>	javax.swing.border.EmptyBorder	
<a href="#"><u>etchedBorder</u></a>	javax.swing.border.EtchedBorder	
<a href="#"><u>fileChooser</u></a>	javax.swing.JFileChooser	
<a href="#"><u>flowLayout</u></a>	java.awt.FlowLayout	
<a href="#"><u>formattedTextField</u></a>	javax.swing.JFormattedTextField	
<a href="#"><u>frame</u></a>	javax.swing.JFrame	
<a href="#"><u>gbc</u></a>	java.awt.GridBagConstraints	alias for gridBagConstraintss
<a href="#"><u>glue</u></a>		calls Box.createGlue()
<a href="#"><u>gridBagConstraints</u></a>	java.awt.GridBagConstraints	
<a href="#"><u>gridBagLayout</u></a>	java.awt.GridBagLayout	
<a href="#"><u>gridLayout</u></a>	java.awt.GridLayout	
<a href="#"><u>hbox</u></a>		calls Box.createHorizontalBox()
<a href="#"><u>hglue</u></a>		calls Box.createHorizontalGlue()
<a href="#"><u>hstrut</u></a>		calls Box.createHorizontalStrut()
<a href="#"><u>internalFrame</u></a>	javax.swing.JInternalFrame	
<a href="#"><u>imageIcon</u></a>	javax.swing.ImageIcon	

<a href="#">label</a>	javax.swing.JLabel	
<a href="#">layeredPane</a>	javax.swing.JLayeredPane	
<a href="#">lineBorder</a>	javax.swing.border.LineBorder	
<a href="#">list</a>	javax.swing.JList	
<a href="#">loweredBevelBorder</a>	javax.swing.border.BevelBorder	
<a href="#">loweredEtchedBorder</a>	javax.swing.border.EtchedBorder	
<a href="#">map</a>	java.util.Map	returns the attributes
<a href="#">matteBorder</a>	javax.swing.border.MatteBorder	
<a href="#">menu</a>	javax.swing.JMenu	
<a href="#">menuBar</a>	javax.swing.JMenuBar	
<a href="#">menuItem</a>	javax.swing.JMenuItem	
<a href="#">optionPane</a>	javax.swing.JOptionPane	
<a href="#">overlayLayout</a>	javax.swing.OverlayLayout	
<a href="#">panel</a>	javax.swing.JPanel	
<a href="#">passwordField</a>	javax.swing.JPasswordField	
<a href="#">popupMenu</a>	javax.swing.JPopupMenu	
<a href="#">progressBar</a>	javax.swing.JProgressBar	
<a href="#">propertyColumn</a>	groovy.model.DefaultTableColumn	
<a href="#">radioButton</a>	javax.swing.JRadioButton	
<a href="#">raisedBevelBorder</a>	javax.swing.border.BevelBorder	
<a href="#">raisedEtchedBorder</a>	javax.swing.border.EtchedBorder	
<a href="#">radioButtonMenuItem</a>	javax.swing.JRadioButtonMenuItem	
rigidArea		calls Box.createRigidArea()
<a href="#">scrollBar</a>	javax.swing.JScrollBar	

<a href="#">scrollPane</a>	javax.swing.JScrollPane	
<a href="#">separator</a>	javax.swing.JSeparator	
<a href="#">slider</a>	javax.swing.JSlider	
<a href="#">spinner</a>	javax.swing.JSpinner	
<a href="#">spinnerDateModel</a>	javax.swing.SpinnerDateModel	
<a href="#">spinnerListModel</a>	javax.swing.SpinnerListModel	
<a href="#">spinnerNumberModel</a>	javax.swing.SpinnerNumberModel	
<a href="#">splitPane</a>	javax.swing.JSplitPane	
<a href="#">springLayout</a>	javax.swing.SpringLayout	
<a href="#">tabbedPane</a>	javax.swing.JTabbedPane	
<a href="#">table</a>	javax.swing.JTable	
<a href="#">tableColumn</a>	javax.swing.table.TableColumn	
<a href="#">tableLayout</a>	groovy.swing.impl.TableLayout	
<a href="#">tableModel</a>	javax.swing.table.TableModel	can pass value through
<a href="#">td</a>	groovy.swing.impl.TableLayoutCell	
<a href="#">textArea</a>	javax.swing.JTextArea	
<a href="#">textField</a>	javax.swing.JTextField	
<a href="#">textPane</a>	javax.swing.JTextPane	
<a href="#">titledBorder</a>	javax.swing.border.TitledBorder	
<a href="#">toggleButton</a>	javax.swing.JToggleButton	
<a href="#">toolBar</a>	javax.swing.JToolBar	
<a href="#">tr</a>	groovy.swing.impl.TableLayoutRow	
<a href="#">tree</a>	javax.swing.JTree	
<a href="#">vbox</a>		calls Box.createVerticalBox()

vglue		calls Box.createVerticalGlue()
<a href="#">viewport</a>	javax.swing.JViewport	
vstrut		calls Box.createVerticalStrut()
<a href="#">widget</a>		pass through, should have no child content
<a href="#">window</a>	javax.swing.JWindow	

### *SwingBuidler.borderLayout*

(Automatically generated from BeanInfo)

## Generated Object

A java.awt.BorderLayout is returned, unless the user passes in a subclass of BorderLayout as the value argument, in which case the value argument is returned.

## Attributes Declared in java.awt.BorderLayout

\* \*hgap\* <int>  
 \* \*vgap\* <int>

## Events Content Usage Examples Observable Attributes

### *SwingBuilder.action*

## Generated Object

A javax.swing.Action object, of some sort.

## Value Argument

A pre-existing javax.swing.Action instance can be provided. In this cases the attributes will be applied to that action instead of creating a new groovy.swing.impl.DefaultAction.

## Attributes

- **closure** <Closure> The action to be performed when the actionPerformed method is called. The actionPerformed will be the only parameter passed in.
- **name** <String> The name to be shown on a button or menu item
- **shortDescription** <String> The text to be displayed as a button's tool tip
- **smallIcon** <String> The icon shown on the button or menu item
- **actionCommandKey** <String> Sets the actionCommand on the button or menu item
- **accelerator** <String> or <KeyStroke> The keyboard accelerator for a menu item to associate with this action
- **mnemonic** <char> or <String> The mnemonic for the action in the button or menu item
- **enabled** <boolean> Whether this action is enabled. Will be reflected in enabled state of button or menu item

unused Action magic values

- **longDescription** <String> Intended for context sensitive help (like a status bar).

## Content

No content is accepted by an action() node.

## Usage

Actions are the lifeblood of a full fledged swing application. [Other sources](#) can expound on their usefulness.

## Examples

//TBD

### *SwingBuilder.actions*

(Automatically generated from BeanInfo)

## Generated Object

A java.util.Collection is returned, unless the user passes in a subclass of Collection as the value argument, in which case the value argument is returned.

## Attributes Declared in java.util.Collection

- **empty** <boolean>

## Events Content Usage Examples Observable Attributes

### *SwingBuilder.bind*

## Generated Object

An internal class that currently should not be relied on.

## Attributes

- **source** <Object> The object that triggers the binding update
- **sourceProperty** <String> A property that is read for the update and whose change triggers the update
- **sourceEvent** <String> The event name trigger an update
- **sourceValue** <Closure> A closure providing the value when sourceEvent triggers
- **target** <Object> The object to be updated when a bound value changes
- **targetProperty** <String> The property on the target update to which the queried value will be written
- **validator** <Closure> If the closure returns null or Boolean.FALSE then a triggered binding will be canceled
- **converter** <Closure> Takes the queried value and converts it to another object to be written to the targetProperty. The queried value is passed in "it" to the closure.

## Content

*bind* accepts no content

# Usage

*bind* can be used as a stand alone element or as a value for any attribute of any object.

When used as an attribute value the target and targetProperty values implicitly become the current object being constructed and the property being written to.

Until the external API is finalized, it is not recommended to use the object returned from *bind*

# Examples

Example binding one property to another:

```
swing.frame() {
  panel {
    textField('Change Me!', id:'tf')
    button(text: bind(source:tf,
sourceProperty:'text'))
  }
}
```

Example when triggering on a particular event with a value closure

```
swing.frame() {
  panel {
    textField('Change Me!', id:'tf')
    button(text: bind(source:tf.document,
sourceEvent:'undoableEditHappened',
sourceValue:{System.currentTimeMillis()}))
  }
}
```

Note that the source value can get any arbitrary value.

Example with validator and converter:

```

swing.frame() {
  panel
  {
    textField('Change Me!', id:'tf')
    button(text: bind(source:tf,
sourceProperty:'text',
    validator: { tf.text.length() >= 1},
converter: { it.toLowerCase() } ))
  }
}

```

In this example the validator prevents the button text from having less than 1 char and the converter makes the button text all lowercase.

**Note:** Tested on Groovy 1.5.1

#### *SwingBuilder.borderLayout*

(Automatically generated from BeanInfo)

## Generated Object

A java.awt.BorderLayout is returned, unless the user passes in a subclass of BorderLayout as the value argument, in which case the value argument is returned.

## Attributes Declared in java.awt.BorderLayout

- hgap <int>
- vgap <int>

## Events Content Usage Examples Observable Attributes

#### *SwingBuilder.boundedRangeModel*

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.DefaultBoundedRangeModel is returned, unless the user passes in a subclass of DefaultBoundedRangeModel as the value argument, in which case the value argument is returned.

## Attributes Declared in

## javax.swing.DefaultBoundedRangeModel

- **extent** <int>
- **maximum** <int>
- **minimum** <int>
- **value** <int>
- **valueIsAdjusting** <boolean>

## Events Declared in javax.swing.DefaultBoundedRangeModel

- **stateChanged** (ChangeEvent) - part of ChangeListener

## Content Usage Examples Observable Attributes

### *SwingBuilder.box*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.Box is returned, unless the user passes in a subclass of Box as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>

- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener

- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

### *SwingBuilder.boxLayout*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)

- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.BoxLayout` is returned, unless the user passes in a subclass of `BoxLayout` as the value argument, in which case the value argument is returned.

## Attributes Events Content Usage Examples Creating a frame with a panel that has a `BoxLayout` with `Y_AXIS`

```
def swing = new SwingBuilder()

swing.edt{
  frame( title:'Frame Title',
        id: 'frameID',
        location: [700,200],
        pack:true,
        defaultCloseOperation:
WindowConstants.DISPOSE_ON_CLOSE,
        show:true){

    panel(border:new BorderLayout(2,2,2,2)){

      boxLayout(axis:BoxLayout.Y_AXIS)
      label("Top Label")
      label("Bottom Label")
    }
  }
}
```

The line

```
BoxLayout (axis:BoxLayout.Y_AXIS)
```

can be replaced with

```
BoxLayout (axis:BoxLayout.X_AXIS)
```

```
BoxLayout (axis:BoxLayout.PAGE_AXIS)
```

etc..

*SwingBuilder.button*

## Generated Object

A javax.swing.JButton is returned, unless the user passes in a subclass of JButton as the value argument, in which case the value argument is returned.

## Attributes

See also: [JComponent](#)

## Declared in JButton

- defaultButton <boolean> Whether or not this button is the default button
- defaultCapable <boolean> Whether or not this button can be the default button

## Declared in AbstractButton

- action <Action> the Action instance connected with this ActionEvent source
- actionCommand <String> actionCommand
- model <ButtonModel> Model that the Button uses.
- multiClickThreshold <long> multiClickThreshold
- selected <boolean> selected

## Text related

- text <String> The button's text.
- label <String> Replace by setText(text)
- mnemonic <int> the keyboard character mnemonic
- displayedMnemonicIndex <int> the index into the String to draw the keyboard character mnemonic at
- horizontalTextPosition <int> The horizontal position of the text relative to the icon.
- verticalTextPosition <int> The vertical position of the text relative to the icon.

## Rendering related

- borderPainted <boolean> Whether the border should be painted.

- `contentAreaFilled` <boolean> Whether the button should paint the content area or leave it transparent.
- `focusPainted` <boolean> Whether focus should be painted
- `rolloverEnabled` <boolean> Whether rollover effects should be enabled.
- `margin` <Insets> The space between the button's border and the label
- `iconTextGap` <int> If both the icon and text properties are set, this property defines the space between them.

## Icon related

- `icon` <Icon> The button's default icon
- `disabledIcon` <Icon> The disabled icon for the button.
- `pressedIcon` <Icon> The pressed icon for the button.
- `rolloverIcon` <Icon> The rollover icon for the button.
- `selectedIcon` <Icon> The selected icon for the button.
- `disabledSelectedIcon` <Icon> The disabled selection icon for the button.
- `rolloverSelectedIcon` <Icon> The rollover selected icon for the button.

## Alignment Related

- `horizontalAlignment` <int> The horizontal alignment of the icon and text.
- `verticalAlignment` <int> The vertical alignment of the icon and text.

## Content

No child content is accepted in a button.

## Usage

button creates a slider-bar type widget.

//TODO add an image

//TODO describe how to fire code on a button press

The `selected` property is not very interesting for the `Button` class, as it is not user changeable. This is mostly a leaky abstraction for `checkBox` and `radioButton`

## Examples

This will run as a Groovy script.

```
import groovy.swing.SwingBuilder
import java.awt.Insets
import javax.swing.JFrame

println 'hello, world'

new SwingBuilder().frame(title: 'demo using
buttons', defaultCloseOperation:
JFrame.EXIT_ON_CLOSE, show: true, pack:
```

```
true) {
    vbox() {
        button(action: action(name: 'basic',
closure: {println "basic"}))
        button(action: action(name:
'mnemonic', mnemonic: 'P', closure: {println
'with mnemonic'}))
        button(action: action(name:
'default', defaultButton: true, closure:
{println 'this is the default button'}))
        button(borderPainted:false, action:
action(name: 'unpainted border', closure:
{println 'unpainted border'}))
        button(contentAreaFilled:false,
action: action(name: 'unfilled content
area', closure: {println 'unfilled content
area'}))
        button(focusPainted:false, action:
action(name: 'unpainted focus (select it and
you\'ll see)', closure: {println 'unpainted
focus'}))
        button(margin: new Insets(5, 10, 15,
20), action: action(name: 'margins',
```

```
closure: {println 'margins'})
    }
}
```

//TBD further examples for icon related behaviour

## Observable Attributes

- defaultCapable
- selected - via synthetic observation
- action
- borderPainted
- contentAreaFilled
- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- focusPainted
- hideActionText
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- label
- margin
- mnemonic
- model
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- verticalAlignment
- verticalTextPosition

### *SwingBuilder.buttonGroup*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.ButtonGroup` is returned, unless the user passes in a subclass of `ButtonGroup` as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.ButtonGroup

- `buttonCount` <int>
- `elements` <Enumeration>
- `selection` <ButtonModel>

## Events Content Usage Examples

*SwingBuilder.cardLayout*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A `java.awt.CardLayout` is returned, unless the user passes in a subclass of `CardLayout` as the value argument, in which case the value argument is returned.

## Attributes Declared in java.awt.CardLayout

- `hgap` <int>
- `vgap` <int>

## Events Content Usage Examples

```
actions
{
    action(id: 'nextAction',
           name: 'Next >>',
           closure: {
cards.layout.show(cards, 'Options') })

    action(id: 'prevAction',
           name: '<< Prev',
           closure: {
cards.layout.show(cards, 'Main') })
}
```

```
application(title: 'Test',
    pack: true,
    locationByPlatform: true,
    iconImage:
imageIcon('/griffon-icon-48x48.png').image,
    iconImages:
[imageIcon('/griffon-icon-48x48.png').image,
imageIcon('/griffon-icon-32x32.png').image,
imageIcon('/griffon-icon-16x16.png').image]
)
{
    panel(id: 'cards')
    {
        cardLayout()

        panel(constraints: 'Main')
        {
            label('Main Page')

            button(action: nextAction)
        }

        panel(constraints: 'Options')
        {
            label('Options Page')

            button(action: prevAction)
        }
    }
}
```

}

```
cards.layout.show(cards, 'Main')
}
```

*SwingBuilder.checkBox*

## Generated Object

A `javax.swing.JCheckBox` is returned, unless the user passes in a subclass of `JCheckBox` as the value argument, in which case the value argument is returned.

## Attributes

See also: [JComponent](#)

## Declared in JCheckBox

- `borderPaintedFlat` <boolean> Whether the border is painted flat.

## Declared in AbstractButton

- `selected` <boolean> If the checkbox is currently selected
- `action` <Action> the Action instance connected with this ActionEvent source
- `actionCommand` <String> actionCommand
- `model` <ButtonModel> Model that the Button uses.
- `multiClickThreshold` <long> multiClickThreshold.
- `enabled` <boolean> Whether the Button is active.

## Text related

- `text` <String> The button's text.
- `label` <String> Replace by `setText(text)`
- `mnemonic` <int> the keyboard character mnemonic
- `displayedMnemonicIndex` <int> the index into the String to draw the keyboard character mnemonic at
- `horizontalTextPosition` <int> The horizontal position of the text relative to the icon.
- `verticalTextPosition` <int> The vertical position of the text relative to the icon.

## Rendering related

- `borderPainted` <boolean> Whether the border should be painted.
- `contentAreaFilled` <boolean> Whether the button should paint the content area or leave it transparent.
- `focusPainted` <boolean> Whether focus should be painted
- `rolloverEnabled` <boolean> Whether rollover effects should be enabled.
- `margin` <Insets> The space between the button's border and the label
- `iconTextGap` <int> If both the icon and text properties are set, this property defines the space between them.

## Icon related

- `icon` <Icon> The button's default icon. The Icon is the checkbox in the unselected state. Set it to null to get the platform default.
- `disabledIcon` <Icon> The disabled icon for the button.
- `pressedIcon` <Icon> The pressed icon for the button.

- rolloverIcon <Icon> The rollover icon for the button.
- selectedIcon <Icon> The selected icon for the button. The Icon is the checkbox in the selected state. Set it to null to get the platform default.
- disabledSelectedIcon <Icon> The disabled selection icon for the button.
- rolloverSelectedIcon <Icon> The rollover selected icon for the button.

## Alignment Related

- horizontalAlignment <int> The horizontal alignment of the icon and text.
- verticalAlignment <int> The vertical alignment of the icon and text.

## Content

No child content is accepted in a checkBox.

## Usage

checkBox creates a visual toggling checkbox type widget.

//TODO add an image

## Examples

//TODO

## Observable Attributes

- borderPaintedFlat
- selected - via synthetic observation
- action
- borderPainted
- contentAreaFilled
- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- focusPainted
- hideActionText
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- label
- margin
- mnemonic
- model
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- verticalAlignment
- verticalTextPosition

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JCheckBoxMenuItem` is returned, unless the user passes in a subclass of `JCheckBoxMenuItem` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JCheckBoxMenuItem`

- **state** <boolean> The selection state of the check box menu item

## Declared in `javax.swing.JMenuItem`

- **accelerator** <KeyStroke> The keystroke combination which will invoke the `JMenuItem`'s actionlisteners without navigating the menu hierarchy
- **armed** <boolean> Mouse release will fire an action event
- **subElements** <MenuElement[]>

## Declared in `javax.swing.AbstractButton`

- **UI** <ButtonUI> The UI object that implements the LookAndFeel.
- **action** <Action> the Action instance connected with this ActionEvent source
- **borderPainted** <boolean> Whether the border should be painted.
- **contentAreaFilled** <boolean> Whether the button should paint the content area or leave it transparent.
- **disabledIcon** <Icon> The disabled icon for the button.
- **disabledSelectedIcon** <Icon> The disabled selection icon for the button.
- **displayedMnemonicIndex** <int> the index into the String to draw the keyboard character mnemonic at
- **focusPainted** <boolean> Whether focus should be painted
- **horizontalAlignment** <int> The horizontal alignment of the icon and text.
- **horizontalTextPosition** <int> The horizontal position of the text relative to the icon.
- **icon** <Icon> The button's default icon
- **iconTextGap** <int> If both the icon and text properties are set, this property defines the space between them.
- **label** <String> Replace by `setText(text)`
- **margin** <Insets> The space between the button's border and the label.
- **mnemonic** <int> the keyboard character mnemonic
- **model** <ButtonModel> Model that the Button uses.
- **pressedIcon** <Icon> The pressed icon for the button.
- **rolloverEnabled** <boolean> Whether rollover effects should be enabled.
- **rolloverIcon** <Icon> The rollover icon for the button.
- **rolloverSelectedIcon** <Icon> The rollover selected icon for the button.
- **selectedIcon** <Icon> The selected icon for the button.
- **text** <String> The button's text.
- **verticalAlignment** <int> The vertical alignment of the icon and text.
- **verticalTextPosition** <int> The vertical position of the text relative to the icon.
- **actionCommand** <String>
- **multiClickThreshold** <long>

- **selected** <boolean>
- **selectedObjects** <Object[]>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>

- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JMenuItem

- **menuKeyPressed** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyReleased** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyTyped** (MenuKeyEvent) - part of MenuKeyListener
- **menuDragMouseDragged** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseEntered** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseExited** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseReleased** (MenuDragMouseEvent) - part of MenuDragMouseListener

## Declared in javax.swing.AbstractButton

- **itemStateChanged** (ItemEvent) - part of ItemListener
- **actionPerformed** (ActionEvent) - part of ActionListener
- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener

- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples

An example of binding a checkBox state with a model. This is a one way binding (checkBox -> model).

```
def cbiMerge =
  checkBoxMenuItem(setMergeData)
  //setMergeData is an action
  bind(source: cbiMerge,
    sourceProperty:'selected', target:model,
    targetProperty:'isMergeData')
  //model class
  class Model {
    //...
    @Bindable boolean isMergeData
  }
```

## Observable Attributes

- UI
- accelerator
- action
- alignmentX
- alignmentY
- background
- border
- borderPainted
- componentCount
- componentPopupMenu
- components
- containerListeners
- contentAreaFilled
- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusPainted
- focusTraversalKeys

- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- margin
- maximumSize
- minimumSize
- mnemonic
- model
- opaque
- preferredSize
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

#### ***SwingBuilder.closureColumn***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A groovy.model.DefaultTableColumn is returned, unless the user passes in a subclass of DefaultTableColumn as the value argument, in which case the value argument is returned.

## **Attributes Declared in groovy.model.DefaultTableColumn**

- **type** <Class>
- **valueModel** <ValueModel>

## Declared in javax.swing.table.TableColumn

- `cellEditor` <TableCellEditor>
- `cellRenderer` <TableCellRenderer>
- `headerRenderer` <TableCellRenderer>
- `headerValue` <Object>
- `identifier` <Object>
- `maxWidth` <int>
- `minWidth` <int>
- `modelIndex` <int>
- `preferredWidth` <int>
- `resizable` <boolean>
- `width` <int>

## Events Declared in javax.swing.table.TableColumn

- `propertyChange` (PropertyChangeEvent) - part of PropertyChangeListener

## Content Usage Examples

```
table {
  tableModel(list: listOfItems) {
    closureColumn(
      header: 'Location',
      read: {row -> return row.location},
      write: {row, newValue -> println
"write: $row new value: $newValue"}
    )
  }
}
```

## Observable Attributes

- `cellEditor`
- `cellRenderer`
- `headerRenderer`
- `headerValue`
- `identifier`
- `maxWidth`
- `minWidth`
- `modelIndex`
- `preferredWidth`
- `propertyChangeListeners`
- `resizable`
- `type`

- valueModel
- width

### *SwingBuilder.colorChooser*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JColorChooser is returned, unless the user passes in a subclass of JColorChooser as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JColorChooser

- **UI** <ColorChooserUI> The UI object that implements the color chooser's LookAndFeel.
- **chooserPanels** <AbstractColorChooserPanel[]> An array of different chooser types.
- **color** <Color> The current color the chooser is to display.
- **dragEnabled** <boolean> Determines whether automatic drag handling is enabled.
- **previewPanel** <JComponent> The UI component which displays the current color.
- **selectionModel** <ColorSelectionModel> The model which contains the currently selected color.

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>

- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener

- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- chooserPanels
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- previewPanel
- selectionModel
- transferHandler
- verifyInputWhenFocusTarget

### **SwingBuilder.comboBox**

- [Generated Object](#)
- [Attributes](#)

- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JComboBox` is returned, unless the user passes in a subclass of `JComboBox` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JComboBox`

- **UI** `<ComboBoxUI>` The UI object that implements the Component's LookAndFeel.
- **action** `<Action>` the Action instance connected with this ActionEvent source
- **editable** `<boolean>` If true, the user can type a new value in the combo box.
- **editor** `<ComboBoxEditor>` The editor that combo box uses to edit the current value
- **keySelectionManager** `<KeySelectionManager>` The objects that changes the selection when a key is pressed.
- **lightWeightPopupEnabled** `<boolean>` Set to `<code>>false</code>` to require heavyweight popups.
- **maximumRowCount** `<int>` The maximum number of rows the popup should have
- **model** `<ComboBoxModel>` Model that the combo box uses to get data to display.
- **prototypeDisplayValue** `<Object>` The display prototype value, used to compute display width and height.
- **renderer** `<ListCellRenderer>` The renderer that paints the item selected in the list.
- **selectedIndex** `<int>` The item at index is selected.
- **selectedItem** `<Object>` Sets the selected item in the JComboBox.
- **actionCommand** `<String>`
- **itemCount** `<int>`
- **popupVisible** `<boolean>`
- **selectedObjects** `<Object[]>`

## Declared in `javax.swing.JComponent`

- **autoscrolls** `<boolean>` Determines if this component automatically scrolls its contents when dragged.
- **border** `<Border>` The component's border.
- **componentPopupMenu** `<JPopupMenu>` Popup to show
- **debugGraphicsOptions** `<int>` Diagnostic options for graphics operations.
- **inheritsPopupMenu** `<boolean>` Whether or not the JPopupMenu is inherited
- **inputVerifier** `<InputVerifier>` The component's input verifier.
- **opaque** `<boolean>` The component's opacity
- **toolTipText** `<String>` The text to display in a tool tip.
- **transferHandler** `<TransferHandler>` Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** `<boolean>` Whether the Component verifies input before accepting focus.
- **UIClassID** `<String>`
- **accessibleContext** `<AccessibleContext>`
- **actionMap** `<ActionMap>`
- **doubleBuffered** `<boolean>`
- **graphics** `<Graphics>`
- **height** `<int>`
- **managingFocus** `<boolean>`
- **nextFocusableComponent** `<Component>`

- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComboBox

- **itemStateChanged** (ItemEvent) - part of ItemListener
- **actionPerformed** (ActionEvent) - part of ActionListener
- **popupMenuCanceled** (PopupMenuEvent) - part of PopupMenuListener
- **popupMenuWillBecomeInvisible** (PopupMenuEvent) - part of PopupMenuListener
- **popupMenuWillBecomeVisible** (PopupMenuEvent) - part of PopupMenuListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener

- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- action
- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- editable
- editor
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier

- insets
- layout
- lightWeightPopupEnabled
- maximumRowCount
- maximumSize
- minimumSize
- model
- opaque
- preferredSize
- prototypeDisplayValue
- renderer
- transferHandler
- verifyInputWhenFocusTarget

*SwingBuilder.compoundBorder*

## Generated Object

The generated object is one or more nested CompoundBorders creating a border from the supplied border.

## Value Argument

The optional value argument is an array of one or more border objects. If only one border object is passed in then the result of the node is that border. Otherwise the result is one or more nested compound borders containing the borders in an outside to inside order. The compound border representing the innermost border and its parent compound will be returned.

If the value argument is used the `outerBorder:` and `innerBorder:` attribute arguments are prohibited.

## Attributes

Only the simple two border compound can be created via attributes. Both `inner:` and `outer:` attributes are required.

- **outerBorder** <Border> the outside border
- **innerBorder** <Border> the inside border
- **parent** <boolean> Whether or not this border is to be added to the parent JComponent

## Content

`compoundBorder()` is a leaf node, no child content is allowed.

## Examples

```
panel (border:
compoundBorder ([ emptyBorder (3) ,
titledBorder ('title' ) ]))
```

```
panel {
    compoundBorder (outerBorder:emptyBorder (3) ,
insideBorder:titledBorder ('title' ) ,
parent:true)
}
```

```
compoundBorder ([ raisedBevelBorder () ,
loweredBevelBorder , etchedBorder ])
```

#### *SwingBuilder.container*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A java.awt.Container is returned, unless the user passes in a subclass of Container as the value argument, in which case the value argument is returned.

## Attributes Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>

- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- component
- componentCount
- components
- containerListeners
- focusCycleRoot

- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- insets
- layout
- maximumSize
- minimumSize
- preferredSize

### **SwingBuilder.desktopPane**

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A javax.swing.JDesktopPane is returned, unless the user passes in a subclass of JDesktopPane as the value argument, in which case the value argument is returned.

## **Attributes Declared in javax.swing.JDesktopPane**

- **UI** <DesktopPaneUI> The UI object that implements the Component's LookAndFeel.
- **desktopManager** <DesktopManager> Desktop manager to handle the internal frames in the desktop pane.
- **dragMode** <int> Dragging style for internal frame children.
- **allFrames** <JInternalFrame[]>
- **selectedFrame** <JInternalFrame>

## **Declared in javax.swing.JComponent**

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>

- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- desktopManager
- dragMode
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize

- opaque
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

### *SwingBuilder.dialog*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JDialog is returned, unless the user passes in a subclass of JDialog as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JDialog

- **JMenuBar** <JMenuBar> The menubar for accessing pulldown menus from this dialog.
- **contentPane** <Container> The client area of the dialog where child components are normally inserted.
- **defaultCloseOperation** <int> The dialog's default close operation.
- **glassPane** <Component> A transparent pane used for menu rendering.
- **layeredPane** <JLayeredPane> The pane which holds the various dialog layers.
- **rootPane** <JRootPane> the RootPane object for this dialog.

## Declared in java.awt.Dialog

- **modal** <boolean>
- **resizable** <boolean>
- **title** <String>
- **undecorated** <boolean>

## Declared in java.awt.Window

- **accessibleContext** <AccessibleContext>
- **active** <boolean>
- **alwaysOnTop** <boolean>
- **bufferStrategy** <BufferStrategy>
- **cursor** <Cursor>
- **focusCycleRootAncestor** <Container>
- **focusOwner** <Component>
- **focusableWindow** <boolean>
- **focusableWindowState** <boolean>
- **focused** <boolean>
- **graphicsConfiguration** <GraphicsConfiguration>
- **inputContext** <InputContext>
- **locale** <Locale>
- **locationByPlatform** <boolean>
- **locationRelativeTo** <Component>

- **mostRecentFocusOwner** <Component>
- **ownedWindows** <Window[]>
- **owner** <Window>
- **showing** <boolean>
- **toolkit** <Toolkit>
- **warningString** <String>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in java.awt.Window

- **windowStateChanged** (WindowEvent) - part of WindowStateListener
- **windowGainedFocus** (WindowEvent) - part of WindowFocusListener
- **windowLostFocus** (WindowEvent) - part of WindowFocusListener
- **windowActivated** (WindowEvent) - part of WindowListener
- **windowClosed** (WindowEvent) - part of WindowListener
- **windowClosing** (WindowEvent) - part of WindowListener
- **windowDeactivated** (WindowEvent) - part of WindowListener
- **windowDeiconified** (WindowEvent) - part of WindowListener
- **windowIconified** (WindowEvent) - part of WindowListener
- **windowOpened** (WindowEvent) - part of WindowListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener

- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- accessibleContext
- active
- alignmentX
- alignmentY
- alwaysOnTop
- background
- bufferStrategy
- component
- componentCount
- components
- containerListeners
- cursor
- focusCycleRoot
- focusCycleRootAncestor
- focusOwner
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- focusableWindow
- focusableWindowState
- focused
- font
- foreground
- graphicsConfiguration
- inputContext
- insets
- layout
- locale
- locationByPlatform

- locationRelativeTo
- maximumSize
- minimumSize
- modal
- mostRecentFocusOwner
- ownedWindows
- owner
- preferredSize
- resizable
- showing
- title
- toolkit
- undecorated
- warningString
- windowFocusListeners
- windowListeners
- windowStateListeners

### ***SwingBuilder.editorPane***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A javax.swing.JEditorPane is returned, unless the user passes in a subclass of JEditorPane as the value argument, in which case the value argument is returned.

## **Attributes Declared in javax.swing.JEditorPane**

- **contentType** <String> the type of content
- **editorKit** <EditorKit> the currently installed kit for handling content
- **page** <URL> the URL used to set content

## **Declared in javax.swing.text.JTextComponent**

- **caret** <Caret> the caret used to select/navigate
- **caretColor** <Color> the color used to render the caret
- **caretPosition** <int> the caret position
- **disabledTextColor** <Color> color used to render disabled text
- **document** <Document> the text document model
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **editable** <boolean> specifies if the text can be edited
- **focusAccelerator** <char> accelerator character used to grab focus
- **highlighter** <Highlighter> object responsible for background highlights
- **keymap** <Keymap> set of key event to action bindings to use
- **margin** <Insets> desired space between the border and text area
- **selectedTextColor** <Color> color used to render selected text

- **selectionColor** <Color> color used to render selection background
- **selectionEnd** <int> ending location of the selection.
- **selectionStart** <int> starting location of the selection.
- **text** <String> the text of this component
- **UI** <TextUI>
- **inputMethodRequests** <InputMethodRequests>
- **navigationFilter** <NavigationFilter>
- **preferredScrollableViewportSize** <Dimension>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectedText** <String>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>

- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in **java.awt.Component**

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in **javax.swing.JEditorPane**

- **hyperlinkUpdate** (HyperlinkEvent) - part of HyperlinkListener

## Declared in **javax.swing.text.JTextComponent**

- **caretUpdate** (CaretEvent) - part of CaretListener

## Declared in **javax.swing.JComponent**

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in **java.awt.Container**

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in **java.awt.Component**

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener

- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- caret
- caretColor
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- disabledTextColor
- document
- editable
- editorKit
- enabled
- focusAccelerator
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- highlighter
- inheritsPopupMenu
- inputVerifier
- insets
- keymap
- layout
- margin
- maximumSize
- minimumSize
- opaque
- page
- preferredSize
- selectedTextColor
- selectionColor
- transferHandler
- verifyInputWhenFocusTarget

*SwingBuilder.emptyBorder*

## Generated Object

A `javax.swing.border.EmptyBorder`, a border whose margins are not painted with anything.

## Value Argument

Either an Integer or an array of four Integers.

A single integer is expanded to an array of four integers of the same value.

An array of four integers is used to generate the top, left, bottom, and right margins, respectively.

If a value argument is used then the `top:`, `left:`, `bottom:`, and `right:` attributes cannot be used.

## Attributes

If any of `top:`, `left:`, `bottom:`, or `right:` are specified the all of them must be specified.

- **top** <int> the top margin of the border
- **left** <int> the left margin of the border
- **bottom** <int> the bottom margin of the border
- **right** <int> the right margin of the border
- **parent** <boolean> Whether or not this border is to be added to the parent JComponent

## Content

`emptyBorder()` is a leaf node, no child content is allowed.

## Examples

```
panel (border:emptyBorder(3))
```

```
panel {  
    emptyBorder([6,6,3,3], parent:true)  
}
```

```
emptyBorder(top:5, left:15, bottom:15,  
right:5)
```

**Note:**

Do work:

```
border:
```

```
BorderFactory.createEmptyBorder(0,3,0,3)
```

```
border: emptyBorder([0,3,0,3])
```

```
border:
```

```
emptyBorder(top:0,left:3,bottom:0,right:3)
```

Does not work:

```
border: emptyBorder(0,3,0,3)
```

```
border: emptyBorder(top:3)
```

*SwingBuilder.etchedBorder*

## Generated Object

A javax.swing.border.EtchedBorder, a border that looks like a groove has been etched into the panel.

## Value Argument

etchedBorder() takes no value arguments.

## Attributes

If any of shadow: or highlight: are specified, all must be specified.

- **highlight** <Color> The color that is to be painted on the highlight side of the etch.
- **shadow** <Color> The color that is to be painted on the shadow side of the etch.
- **parent** <boolean> Whether or not this border is to be added to the parent JComponent.

## Content

etchedBorder() is a leaf node, no child content is allowed.

## Examples

```
panel (border:etchedBorder ( ) )
```

```
panel {  
    etchedBorder (shadow:Color.RED,  
highlight:Color.PINK, parent:true)  
}
```

### *SwingBuilder.fileChooser*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JFileChooser is returned, unless the user passes in a subclass of JFileChooser as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JFileChooser

- **acceptAllFileFilterUsed** <boolean> Sets whether the AcceptAll FileFilter is used as an available choice in the choosable filter list.
- **accessory** <JComponent> Sets the accessory component on the JFileChooser.
- **approveButtonMnemonic** <int> The mnemonic key accelerator for the ApproveButton.
- **approveButtonText** <String> The text that goes in the ApproveButton.
- **approveButtonToolTipText** <String> The tooltip text for the ApproveButton.
- **controlButtonsAreShown** <boolean> Sets whether the approve & cancel buttons are shown.
- **currentDirectory** <File> The directory that the JFileChooser is showing files of.
- **dialogTitle** <String> The title of the JFileChooser dialog window.
- **dialogType** <int> The type (open, save, custom) of the JFileChooser.
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **fileFilter** <FileFilter> Sets the File Filter used to filter out files of type.
- **fileHidingEnabled** <boolean> Sets file hiding on or off.
- **fileSelectionMode** <int> Sets the types of files that the JFileChooser can choose.
- **fileSystemView** <FileSystemView> Sets the FileSytemView used to get filesystem information.
- **fileView** <FileView> Sets the File View used to get file type information.
- **multiSelectionEnabled** <boolean> Sets multiple file selection mode.
- **selectedFiles** <File[]> The list of selected files if the chooser is in multiple selection mode.
- **UI** <FileChooserUI>

- **acceptAllFileFilter** <FileFilter>
- **choosableFileFilters** <FileFilter[]>
- **directorySelectionEnabled** <boolean>
- **fileSelectionEnabled** <boolean>
- **selectedFile** <File>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JFileChooser

- **actionPerformed** (ActionEvent) - part of ActionListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples

Example of a dialog to choose .xls files:

```

import javax.swing.filechooser.FileFilter

def openExcelDialog =
fileChooser(dialogTitle:"Choose an excel
file",

id:"openExcelDialog", fileSelectionMode :
JFileChooser.FILES_ONLY,

//the
file filter must show also directories, in
order to be able to look into them

fileFilter: [getDescription: {-> "*.xls"},
accept:{file-> file ==~ /.*?\.xls/ ||
file.isDirectory() }] as FileFilter) {
}
//later, in the controller
def fc = openExcelDialog
if(fc.showOpenDialog() !=
JFileChooser.APPROVE_OPTION) return //user
cancelled
model.configFile = fc.selectedFile

```

## Observable Attributes

- acceptAllFileFilterUsed
- accessory
- alignmentX
- alignmentY
- approveButtonMnemonic
- approveButtonText
- approveButtonToolTipText
- background
- border
- component

- componentCount
- componentPopupMenu
- components
- containerListeners
- controlButtonsAreShown
- currentDirectory
- dialogTitle
- dialogType
- enabled
- fileFilter
- fileHidingEnabled
- fileSelectionMode
- fileSystemView
- fileView
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- multiSelectionEnabled
- opaque
- preferredSize
- selectedFile
- selectedFiles
- transferHandler
- verifyInputWhenFocusTarget

### ***SwingBuilder.flowLayout***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A java.awt.FlowLayout is returned, unless the user passes in a subclass of FlowLayout as the value argument, in which case the value argument is returned.

## **Attributes Declared in java.awt.FlowLayout**

- **alignment** <int>
- **hgap** <int>
- **vgap** <int>

## Events Content Usage Examples

### *SwingBuilder.formattedTextField*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JFormattedTextField` is returned, unless the user passes in a subclass of `JFormattedTextField` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JFormattedTextField`

- **focusLostBehavior** <int> Behavior when component loses focus
- **formatter** <AbstractFormatter> `TextFormatter`, responsible for formatting the current value
- **formatterFactory** <AbstractFormatterFactory> `AbstractFormatterFactory`, responsible for returning an `AbstractFormatter` that can format the current value.
- **value** <Object> The value to be formatted.
- **editValid** <boolean>

## Declared in `javax.swing.JTextField`

- **action** <Action> the Action instance connected with this ActionEvent source
- **columns** <int> the number of columns preferred for display
- **horizontalAlignment** <int> Set the field alignment to LEFT, CENTER, RIGHT, LEADING (the default) or TRAILING
- **horizontalVisibility** <BoundedRangeModel>
- **scrollOffset** <int>

## Declared in `javax.swing.text.JTextComponent`

- **caret** <Caret> the caret used to select/navigate
- **caretColor** <Color> the color used to render the caret
- **caretPosition** <int> the caret position
- **disabledTextColor** <Color> color used to render disabled text
- **document** <Document> the text document model
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **editable** <boolean> specifies if the text can be edited
- **focusAccelerator** <char> accelerator character used to grab focus
- **highlighter** <Highlighter> object responsible for background highlights
- **keymap** <Keymap> set of key event to action bindings to use
- **margin** <Insets> desired space between the border and text area
- **selectedTextColor** <Color> color used to render selected text

- **selectionColor** <Color> color used to render selection background
- **selectionEnd** <int> ending location of the selection.
- **selectionStart** <int> starting location of the selection.
- **text** <String> the text of this component
- **UI** <TextUI>
- **inputMethodRequests** <InputMethodRequests>
- **navigationFilter** <NavigationFilter>
- **preferredScrollableViewportSize** <Dimension>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectedText** <String>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>

- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in **java.awt.Component**

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in **javax.swing.JTextField**

- **actionPerformed** (ActionEvent) - part of ActionListener

## Declared in **javax.swing.text.JTextComponent**

- **caretUpdate** (CaretEvent) - part of CaretListener

## Declared in **javax.swing.JComponent**

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in **java.awt.Container**

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in **java.awt.Component**

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener

- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- action
- alignmentX
- alignmentY
- background
- border
- caret
- caretColor
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- disabledTextColor
- document
- editable
- enabled
- focusAccelerator
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- formatter
- formatterFactory
- highlighter
- horizontalAlignment
- inheritsPopupMenu
- inputVerifier
- insets
- keymap
- layout
- margin
- maximumSize
- minimumSize
- opaque
- preferredSize
- selectedTextColor
- selectionColor
- transferHandler
- value
- verifyInputWhenFocusTarget

## *SwingBuilder.frame*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JFrame` is returned, unless the user passes in a subclass of `JFrame` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JFrame`

- **JMenuBar** <JMenuBar> The menubar for accessing pulldown menus from this frame.
- **contentPane** <Container> The client area of the frame where child components are normally inserted.
- **defaultCloseOperation** <int> The frame's default close operation.
- **glassPane** <Component> A transparent pane used for menu rendering.
- **layeredPane** <JLayeredPane> The pane that holds the various frame layers.
- **rootPane** <JRootPane> the RootPane object for this frame.

## Declared in `java.awt.Frame`

- **cursorType** <int>
- **extendedState** <int>
- **iconImage** <Image>
- **maximizedBounds** <Rectangle>
- **menuBar** <MenuBar>
- **resizable** <boolean>
- **state** <int>
- **title** <String>
- **undecorated** <boolean>

## Declared in `java.awt.Window`

- **accessibleContext** <AccessibleContext>
- **active** <boolean>
- **alwaysOnTop** <boolean>
- **bufferStrategy** <BufferStrategy>
- **cursor** <Cursor>
- **focusCycleRootAncestor** <Container>
- **focusOwner** <Component>
- **focusableWindow** <boolean>
- **focusableWindowState** <boolean>
- **focused** <boolean>
- **graphicsConfiguration** <GraphicsConfiguration>
- **inputContext** <InputContext>
- **locale** <Locale>
- **locationByPlatform** <boolean>
- **locationRelativeTo** <Component>
- **mostRecentFocusOwner** <Component>

- **ownedWindows** <Window[]>
- **owner** <Window>
- **showing** <boolean>
- **toolkit** <Toolkit>
- **warningString** <String>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in java.awt.Window

- **windowStateChanged** (WindowEvent) - part of WindowStateListener
- **windowGainedFocus** (WindowEvent) - part of WindowFocusListener
- **windowLostFocus** (WindowEvent) - part of WindowFocusListener
- **windowActivated** (WindowEvent) - part of WindowListener
- **windowClosed** (WindowEvent) - part of WindowListener
- **windowClosing** (WindowEvent) - part of WindowListener
- **windowDeactivated** (WindowEvent) - part of WindowListener
- **windowDeiconified** (WindowEvent) - part of WindowListener
- **windowIconified** (WindowEvent) - part of WindowListener
- **windowOpened** (WindowEvent) - part of WindowListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener

- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- accessibleContext
- active
- alignmentX
- alignmentY
- alwaysOnTop
- background
- bufferStrategy
- component
- componentCount
- components
- containerListeners
- cursor
- cursorType
- defaultCloseOperation
- extendedState
- focusCycleRoot
- focusCycleRootAncestor
- focusOwner
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- focusableWindow
- focusableWindowState
- focused
- font
- foreground
- graphicsConfiguration
- iconImage
- inputContext
- insets

- layout
- locale
- locationByPlatform
- locationRelativeTo
- maximizedBounds
- maximumSize
- menuBar
- minimumSize
- mostRecentFocusOwner
- ownedWindows
- owner
- preferredSize
- resizable
- showing
- state
- title
- toolkit
- undecorated
- warningString
- windowFocusListeners
- windowListeners
- windowStateListeners

#### ***SwingBuilder.gridBagConstraints***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A java.awt.GridBagConstraints is returned, unless the user passes in a subclass of GridBagConstraints as the value argument, in which case the value argument is returned.

## **Attributes Events Content Usage Examples**

#### ***SwingBuilder.gridBagLayout***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A `java.awt.GridBagLayout` is returned, unless the user passes in a subclass of `GridBagLayout` as the value argument, in which case the value argument is returned.

## Attributes Declared in `java.awt.GridBagLayout`

- `layoutDimensions` <`int[][]`>
- `layoutOrigin` <`Point`>
- `layoutWeights` <`double[][]`>

## Events Content Usage Examples

```
swing.panel() {
    gridBagLayout()

    label(text:"Hello",
constraints:gbc(gridx:0,gridy:0,gridwidth:GridBagConstraints.REMAINDER,
fill:GridBagConstraints.HORIZONTAL,
insets:[10,10,10,10]))

    button(text:"Click me"
constraints:gbc(gridx:0,gridy:1))
}
```

### *SwingBuilder.gridLayout*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A `java.awt.GridLayout` is returned, unless the user passes in a subclass of `GridLayout` as the value argument, in which case the value argument is returned.

## Attributes Declared in `java.awt.GridLayout`

- `columns` <`int`>
- `hgap` <`int`>

- rows <int>
- vgap <int>

## Events Content Usage Examples

```
import groovy.swing.SwingBuilder
import java.awt.GridLayout
import javax.swing.JFrame

SwingBuilder swing = new SwingBuilder()
swing.frame(defaultCloseOperation:
JFrame.EXIT_ON_CLOSE, pack: true, show:
true) {
    panel(layout: GridLayout(cols: 2, rows:
5)) {

        (0..<10).each {
            println it
            swing.label it.toString()
        }

    }
}
```

*SwingBuilder.imageIcon*

## Generated Object

A javax.swing.ImageIcon, where the supplied image details are loaded into the icon.

## Value Argument

The value argument can be a String, URL, java.awt.Image, or byte[]. There is no attribute equivalent to the byte[] value argument.

If the value argument is a String it can be treated in multiple ways. If the value resolved to a File resolved without any context, then the value argument is treated as though it was passed in as a file: attribute. If the file does not

exist, it is treated as though it were passed in as a resource: attribute.

## Attributes

the value argument, file :, url:, image:, and resource: are mutually exclusive.

- **file** <String> The name of a file where the ImageIcon can be loaded from
- **image** <java.awt.Image> The image to be used for the icon
- **resource** <String> The name of a resource to be resolved against the class loader
  - **class** <Object or Class> The class (or instance of a Class) that the resource is to be resolved against. Relative resources will be resolved relative to the package of the Class in the classes ClassLoader. If this attribute is not passed in then the ClassLoader of the SwingBuilder ImageIconFactory will be used to resolve the resource
- **url** <URL> A URL to an image to load

//TODO we should accept strings to file: and url:, in the future.

## Content

imageIcon() is a leaf node, no child content is allowed.

## Examples

```
label(icon:imageIcon('http://docs.codehaus.org/download/userResources/GROOVY/logo'))
```

```
button(icon:imageIcon(url:'http://www.google.com/favicon.ico'))
```

```
imageIcon('ConsoleIcon.png',  
class:groovy.ui.Console)
```

```
imageIcon(resource:'/groovy/ui/ConsoleIcon.png')
```

*SwingBuilder.internalFrame*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

- [Observable Attributes](#)
- [Vetoable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JInternalFrame is returned, unless the user passes in a subclass of JInternalFrame as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JInternalFrame

- **JMenuBar** <JMenuBar> The menu bar for accessing pulldown menus from this internal frame.
- **UI** <InternalFrameUI> The UI object that implements the Component's LookAndFeel.
- **closable** <boolean> Indicates whether this internal frame can be closed.
- **closed** <boolean> Indicates whether this internal frame has been closed.
- **contentPane** <Container> The client area of the internal frame where child components are normally inserted.
- **desktopIcon** <JDesktopIcon> The icon shown when this internal frame is minimized.
- **framelcon** <Icon> The icon shown in the top-left corner of this internal frame.
- **glassPane** <Component> A transparent pane used for menu rendering.
- **icon** <boolean> The image displayed when this internal frame is minimized.
- **iconifiable** <boolean> Determines whether this internal frame can be iconified.
- **layer** <int> Specifies what desktop layer is used.
- **layeredPane** <JLayeredPane> The pane which holds the various desktop layers.
- **maximizable** <boolean> Determines whether this internal frame can be maximized.
- **maximum** <boolean> Indicates whether this internal frame is maximized.
- **resizable** <boolean> Determines whether this internal frame can be resized by the user.
- **selected** <boolean> Indicates whether this internal frame is currently the active frame.
- **title** <String> The text displayed in the title bar.
- **defaultCloseOperation** <int>
- **desktopPane** <JDesktopPane>
- **focusCycleRootAncestor** <Container>
- **focusOwner** <boolean>
- **menuBar** <JMenuBar>
- **mostRecentFocusOwner** <Component>
- **normalBounds** <Rectangle>
- **warningString** <String>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>

- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JInternalFrame

- **internalFrameActivated** (InternalFrameEvent) - part of InternalFrameListener
- **internalFrameClosed** (InternalFrameEvent) - part of InternalFrameListener
- **internalFrameClosing** (InternalFrameEvent) - part of InternalFrameListener
- **internalFrameDeactivated** (InternalFrameEvent) - part of InternalFrameListener
- **internalFrameDeiconified** (InternalFrameEvent) - part of InternalFrameListener
- **internalFrameIconified** (InternalFrameEvent) - part of InternalFrameListener
- **internalFrameOpened** (InternalFrameEvent) - part of InternalFrameListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in **java.awt.Container**

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in **java.awt.Component**

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- JMenuBar
- UI
- alignmentX
- alignmentY
- background
- border
- closable
- closed
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- contentPane
- desktopIcon
- enabled
- focusCycleRoot

- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- frameIcon
- glassPane
- icon
- iconifiable
- inheritsPopupMenu
- inputVerifier
- insets
- layeredPane
- layout
- maximizable
- maximum
- maximumSize
- minimumSize
- opaque
- preferredSize
- resizable
- rootPane
- selected
- title
- transferHandler
- verifyInputWhenFocusTarget

## Vetoable Attributes

- closed
- icon
- maximum
- selected

### *SwingBuilder.JComponent*

Here are attributes that all JComponent type nodes accept. For a better description of these properties consult the [JDK Documentation](#).

## Attributes

- **visible** <boolean>
- **enabled** <boolean>
- **opaque** <boolean>
- **foreground** <Color>
- **background** <Color>
- **font** <Font>
- **toolTipText** <String>
- **preferredSize** <Dimension>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **border** <Border>

- **alignmentY** <float>
- **alignmentX** <float>
- **componentPopupMenu** <JPopupMenu>
- **inheritsPopupMenu** <boolean>
- **autoScrolls** <boolean>
- **UI** <javax.swing.plaf.ComponentUI>
- **nextFocusableComponent** <Component>
- **requestFocusEnabled** <boolean>
- **verifyInputWhenFocusTarget** <boolean>
- **inputVerifier** <InputVerifier>
- **debugGraphicsOption** <int> DebugGraphics.NONE\_OPTION, DebugGraphics.LOG\_OPTION, DebugGraphics.FLASH\_OPTION, DebugGraphics.BUFFERED\_OPTION
- **transferHandler** <TransferHandler>
- **doubleBuffered** <boolean>
- **actionMap** <ActionMap>

### *SwingBuilder.label*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JLabel is returned, unless the user passes in a subclass of JLabel as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JLabel

- **UI** <LabelUI> The UI object that implements the Component's LookAndFeel.
- **disabledIcon** <Icon> The icon to display if the label is disabled.
- **displayedMnemonic** <int> The mnemonic keycode.
- **displayedMnemonicIndex** <int> the index into the String to draw the keyboard character mnemonic at
- **horizontalAlignment** <int> The alignment of the label's content along the X axis.
- **horizontalTextPosition** <int> The horizontal position of the label's text, relative to its image.
- **icon** <Icon> The icon this component will display.
- **iconTextGap** <int> If both the icon and text properties are set, this property defines the space between them.
- **labelFor** <Component> The component this is labelling.
- **text** <String> Defines the single line of text this component will display.
- **verticalAlignment** <int> The alignment of the label's contents along the Y axis.
- **verticalTextPosition** <int> The vertical position of the text relative to it's image.

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited

- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener

- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- disabledIcon
- displayedMnemonic
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider

- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- labelFor
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

#### ***SwingBuilder.layeredPane***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JLayeredPane is returned, unless the user passes in a subclass of JLayeredPane as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>

- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque

- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

*SwingBuilder.lineBorder*

## Generated Object

A javax.swing.border.LineBorder, a border that consists of a colored line.

## Value Argument

No value argument is accepted.

## Attributes

color: is required.

- **color** <Color> The color of the line
- **roundedCorners** <boolean> Whether or not the border should appear rounded. Only effective with a thickness of 1, otherwise you will get a wierd quasi-bevel effect. Default is false.
- **thickness** <int> The thickness of the line, in pixels. Default is 1.
- **parent** <boolean> Whether or not this border is to be added to the parent JComponent

## Content

lineBorder() is a leaf node, no child content is allowed.

## Examples

```
panel (border : lineBorder (Color . RED ) )
```

```
panel {  
    lineBorder (color : Color . WHITE ,  
thickness : 3 , parent : true )  
}
```

```
lineBorder (color : BLUE , roundedCorners : true )
```

*SwingBuilder.list*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)

- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JList` is returned, unless the user passes in a subclass of `JList` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JList`

- **UI** <ListUI> The UI object that implements the Component's LookAndFeel.
- **cellRenderer** <ListCellRenderer> The component used to draw the cells.
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **fixedCellHeight** <int> Defines a fixed cell height when greater than zero.
- **fixedCellWidth** <int> Defines a fixed cell width when greater than zero.
- **layoutOrientation** <int> Defines the way list cells are layed out.
- **leadSelectionIndex** <int> The lead selection index.
- **model** <ListModel> The object that contains the data to be drawn by this JList.
- **prototypeCellValue** <Object> The cell prototype value, used to compute cell width and height.
- **selectedIndex** <int> The index of the selected cell.
- **selectionBackground** <Color> The background color of selected cells.
- **selectionForeground** <Color> The foreground color of selected cells.
- **selectionMode** <int> The selection mode.
- **selectionModel** <ListSelectionModel> The selection model, recording which cells are selected.
- **visibleRowCount** <int> The preferred number of cells that can be displayed without a scroll bar.
- **anchorSelectionIndex** <int>
- **firstVisibleIndex** <int>
- **lastVisibleIndex** <int>
- **maxSelectionIndex** <int>
- **minSelectionIndex** <int>
- **preferredScrollableViewportSize** <Dimension>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectedIndices** <int[]>
- **selectedValue** <Object>
- **selectedValues** <Object[]>
- **selectionEmpty** <boolean>
- **valuesAdjusting** <boolean>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.

- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JList

- **valueChanged** (ListSelectionEvent) - part of ListSelectionListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener

- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- cellRenderer
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- fixedCellHeight
- fixedCellWidth
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable

- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- layoutOrientation
- maximumSize
- minimumSize
- model
- opaque
- preferredSize
- prototypeCellValue
- selectionBackground
- selectionForeground
- selectionModel
- transferHandler
- verifyInputWhenFocusTarget
- visibleRowCount

*SwingBuilder.loweredBevelBorder*

## Generated Object

A `javax.swing.border.BevelBorder` with type `BevelBorder.LOWERED`, a border that looks recessed into the panel.

## Value Argument

`loweredBevelBorder()` takes no value arguments.

## Attributes

If either `shadow:` or `highlight:` are specified, both must be specified.

If any of `shadowInner:`, `shadowOuter:`, `highlightInner:`, or `highlightOuter:` are specified, all must be specified.

The attributes `shadow:` and `highlight:` are mutually exclusive to `shadowInner:`, `shadowOuter:`, `highlightInner:`, and `highlightOuter:`.

- **highlight** <Color> The color that is to be painted on the highlight side of the bevel.
- **shadow** <Color> The color that is to be painted on the shadow side of the bevel.
- **highlightInner** <Color> The inner color that is to be painted on the highlight side of the bevel.
- **highlightOuter** <Color> The outer color that is to be painted on the highlight side of the bevel.
- **shadowInner** <Color> The inner color that is to be painted on the shadow side of the bevel.
- **shadowOuter** <Color> The outer color that is to be painted on the shadow side of the bevel.
- **parent** <boolean> Whether or not this border is to be added to the parent `JComponent`.

## Content

`loweredBevelBorder()` is a leaf node, no child content is allowed.

## Examples

```
panel (border: loweredBevelBorder ())
```

```
panel {  
    loweredBevelBorder (shadow:Color.RED,  
highlight:Color.PINK, parent:true)  
}
```

```
loweredBevelBorder (  
    shadowOuter:Color.BLACK,  
shadowInner:Color.GREY,  
    highlightOuter:Color.RED,  
highlightInner:Color.PINK)
```

#### *SwingBuilder.map*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A java.util.Map is returned, unless the user passes in a subclass of Map as the value argument, in which case the value argument is returned.

## Attributes Declared in java.util.Map

- empty <boolean>

## Events Content Usage Examples

#### *SwingBuilder.matteBorder*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)

- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.border.MatteBorder` is returned, unless the user passes in a subclass of `MatteBorder` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.border.MatteBorder`

- `matteColor` <Color>
- `tileIcon` <Icon>

## Declared in `javax.swing.border.EmptyBorder`

- `borderInsets` <Insets>

## Declared in `javax.swing.border.AbstractBorder`

- `borderOpaque` <boolean>

## Events Content Usage Examples

### *SwingBuilder.menu*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JMenu` is returned, unless the user passes in a subclass of `JMenu` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JMenu`

- `delay` <int> The delay between menu selection and making the popup menu visible
- `popupMenuVisible` <boolean> The popup menu's visibility
- `itemCount` <int>
- `menuComponentCount` <int>
- `menuComponents` <Component[]>
- `popupMenu` <JPopupMenu>
- `tearOff` <boolean>
- `topLevelMenu` <boolean>

## Declared in `javax.swing.JMenuItemem`

- `accelerator` <KeyStroke> The keystroke combination which will invoke the `JMenuItemem`'s `actionlisteners` without navigating the menu hierarchy
- `armed` <boolean> Mouse release will fire an action event

- **subElements** <MenuElement[]>

## Declared in javax.swing.AbstractButton

- **UI** <ButtonUI> The UI object that implements the LookAndFeel.
- **action** <Action> the Action instance connected with this ActionEvent source
- **borderPainted** <boolean> Whether the border should be painted.
- **contentAreaFilled** <boolean> Whether the button should paint the content area or leave it transparent.
- **disabledIcon** <Icon> The disabled icon for the button.
- **disabledSelectedIcon** <Icon> The disabled selection icon for the button.
- **displayedMnemonicIndex** <int> the index into the String to draw the keyboard character mnemonic at
- **focusPainted** <boolean> Whether focus should be painted
- **horizontalAlignment** <int> The horizontal alignment of the icon and text.
- **horizontalTextPosition** <int> The horizontal position of the text relative to the icon.
- **icon** <Icon> The button's default icon
- **iconTextGap** <int> If both the icon and text properties are set, this property defines the space between them.
- **label** <String> Replace by setText(text)
- **margin** <Insets> The space between the button's border and the label.
- **mnemonic** <int> the keyboard character mnemonic
- **model** <ButtonModel> Model that the Button uses.
- **pressedIcon** <Icon> The pressed icon for the button.
- **rolloverEnabled** <boolean> Whether rollover effects should be enabled.
- **rolloverIcon** <Icon> The rollover icon for the button.
- **rolloverSelectedIcon** <Icon> The rollover selected icon for the button.
- **selectedIcon** <Icon> The selected icon for the button.
- **text** <String> The button's text.
- **verticalAlignment** <int> The vertical alignment of the icon and text.
- **verticalTextPosition** <int> The vertical position of the text relative to the icon.
- **actionCommand** <String>
- **multiClickThreshold** <long>
- **selected** <boolean>
- **selectedObjects** <Object[]>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>

- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JMenu

- **menuCanceled** (MenuEvent) - part of MenuListener
- **menuDeselected** (MenuEvent) - part of MenuListener
- **menuSelected** (MenuEvent) - part of MenuListener

## Declared in javax.swing.JMenuItem

- **menuKeyPressed** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyReleased** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyTyped** (MenuKeyEvent) - part of MenuKeyListener
- **menuDragMouseDragged** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseEntered** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseExited** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseReleased** (MenuDragMouseEvent) - part of MenuDragMouseListener

## Declared in javax.swing.AbstractButton

- **itemStateChanged** (ItemEvent) - part of ItemListener
- **actionPerformed** (ActionEvent) - part of ActionListener
- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- accelerator
- action
- alignmentX
- alignmentY
- background
- border
- borderPainted
- componentCount
- componentPopupMenu
- components
- containerListeners

- contentAreaFilled
- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusPainted
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- margin
- maximumSize
- minimumSize
- mnemonic
- model
- opaque
- preferredSize
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

### ***SwingBuilder.menuBar***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

# Generated Object

A `javax.swing.JMenuBar` is returned, unless the user passes in a subclass of `JMenuBar` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JMenuBar`

- **UI** <MenuBarUI> The UI object that implements the Component's LookAndFeel.
- **borderPainted** <boolean> Whether the border should be painted.
- **margin** <Insets> The space between the menubar's border and its contents
- **selectionModel** <SingleSelectionModel> The selection model, recording which child is selected.
  
- **helpMenu** <JMenu>
- **menuCount** <int>
- **selected** <boolean>
- **subElements** <MenuElement[]>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in `java.awt.Container`

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>

- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in **java.awt.Component**

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in **javax.swing.JComponent**

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in **java.awt.Container**

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in **java.awt.Component**

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener

- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- borderPainted
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- margin
- maximumSize
- minimumSize
- opaque
- preferredSize
- selectionModel
- transferHandler
- verifyInputWhenFocusTarget

### *SwingBuilder.menuitem*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JMenuItemem` is returned, unless the user passes in a subclass of `JMenuItemem` as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JMenuItem

- **accelerator** <KeyStroke> The keystroke combination which will invoke the JMenuItem's actionlisteners without navigating the menu hierarchy
- **armed** <boolean> Mouse release will fire an action event
- **subElements** <MenuElement[]>

## Declared in javax.swing.AbstractButton

- **UI** <ButtonUI> The UI object that implements the LookAndFeel.
- **action** <Action> the Action instance connected with this ActionEvent source
- **borderPainted** <boolean> Whether the border should be painted.
- **contentAreaFilled** <boolean> Whether the button should paint the content area or leave it transparent.
- **disabledIcon** <Icon> The disabled icon for the button.
- **disabledSelectedIcon** <Icon> The disabled selection icon for the button.
- **displayedMnemonicIndex** <int> the index into the String to draw the keyboard character mnemonic at
- **focusPainted** <boolean> Whether focus should be painted
- **horizontalAlignment** <int> The horizontal alignment of the icon and text.
- **horizontalTextPosition** <int> The horizontal position of the text relative to the icon.
- **icon** <Icon> The button's default icon
- **iconTextGap** <int> If both the icon and text properties are set, this property defines the space between them.
- **label** <String> Replace by setText(text)
- **margin** <Insets> The space between the button's border and the label.
- **mnemonic** <int> the keyboard character mnemonic
- **model** <ButtonModel> Model that the Button uses.
- **pressedIcon** <Icon> The pressed icon for the button.
- **rolloverEnabled** <boolean> Whether rollover effects should be enabled.
- **rolloverIcon** <Icon> The rollover icon for the button.
- **rolloverSelectedIcon** <Icon> The rollover selected icon for the button.
- **selectedIcon** <Icon> The selected icon for the button.
- **text** <String> The button's text.
- **verticalAlignment** <int> The vertical alignment of the icon and text.
- **verticalTextPosition** <int> The vertical position of the text relative to the icon.
- **actionCommand** <String>
- **multiClickThreshold** <long>
- **selected** <boolean>
- **selectedObjects** <Object[]>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>

- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JMenuItemem

- **menuKeyPressed** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyReleased** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyTyped** (MenuKeyEvent) - part of MenuKeyListener
- **menuDragMouseDragged** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseEntered** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseExited** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseReleased** (MenuDragMouseEvent) - part of MenuDragMouseListener

## Declared in javax.swing.AbstractButton

- **itemStateChanged** (ItemEvent) - part of ItemListener

- **actionPerformed** (ActionEvent) - part of ActionListener
- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- accelerator
- action
- alignmentX
- alignmentY
- background
- border
- borderPainted
- componentCount
- componentPopupMenu
- components
- containerListeners
- contentAreaFilled

- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusPainted
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- margin
- maximumSize
- minimumSize
- mnemonic
- model
- opaque
- preferredSize
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

### ***SwingBuilder.optionPane***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A `javax.swing.JOptionPane` is returned, unless the user passes in a subclass of `JOptionPane` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JOptionPane`

- **UI** `<OptionPaneUI>` The UI object that implements the optionpane's LookAndFeel
- **icon** `<Icon>` The option pane's type icon.
- **initialSelectionValue** `<Object>` The option pane's initial selection value object.
- **initialValue** `<Object>` The option pane's initial value object.
- **inputValue** `<Object>` The option pane's input value object.
- **message** `<Object>` The optionpane's message object.
- **messageType** `<int>` The option pane's message type.
- **optionType** `<int>` The option pane's option type.
- **options** `<Object[]>` The option pane's options objects.
- **selectionValues** `<Object[]>` The option pane's selection values.
- **value** `<Object>` The option pane's value object.
- **wantsInput** `<boolean>` Flag which allows the user to input a value.
  
- **maxCharactersPerLineCount** `<int>`

## Declared in `javax.swing.JComponent`

- **autoscrolls** `<boolean>` Determines if this component automatically scrolls its contents when dragged.
- **border** `<Border>` The component's border.
- **componentPopupMenu** `<JPopupMenu>` Popup to show
- **debugGraphicsOptions** `<int>` Diagnostic options for graphics operations.
- **inheritsPopupMenu** `<boolean>` Whether or not the `JPopupMenu` is inherited
- **inputVerifier** `<InputVerifier>` The component's input verifier.
- **opaque** `<boolean>` The component's opacity
- **toolTipText** `<String>` The text to display in a tool tip.
- **transferHandler** `<TransferHandler>` Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** `<boolean>` Whether the Component verifies input before accepting focus.
  
- **UIClassID** `<String>`
- **accessibleContext** `<AccessibleContext>`
- **actionMap** `<ActionMap>`
- **doubleBuffered** `<boolean>`
- **graphics** `<Graphics>`
- **height** `<int>`
- **managingFocus** `<boolean>`
- **nextFocusableComponent** `<Component>`
- **optimizedDrawingEnabled** `<boolean>`
- **paintingTile** `<boolean>`
- **registeredKeyStrokes** `<KeyStroke[]>`
- **requestFocusEnabled** `<boolean>`
- **rootPane** `<JRootPane>`
- **topLevelAncestor** `<Container>`
- **validateRoot** `<boolean>`
- **visibleRect** `<Rectangle>`
- **width** `<int>`
- **x** `<int>`
- **y** `<int>`

## Declared in `java.awt.Container`

- **alignmentX** `<float>`

- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener

- `focusLost` (FocusEvent) - part of FocusListener
- `mouseWheelMoved` (MouseEvent) - part of MouseWheelListener
- `caretPositionChanged` (InputMethodEvent) - part of InputMethodListener
- `inputMethodTextChanged` (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples

```
import groovy.swing.SwingBuilder

import javax.swing.*

def swingBuilder = new SwingBuilder()
def options = ['1995', '2003', '2007']
def pane =
swingBuilder.optionPane(message: 'Groovy
starts in ', selectionValues:options,
optionType:JOptionPane.CLOSED_OPTION)
def dialog = pane.createDialog(null,
'worthless message')
dialog.show()

pane = swingBuilder.optionPane(
message:'Are you sure that Groovy starts in
',

selectionValues:options,

initialSelectionValue:pane.getInputValue(),

messageType:JOptionPane.QUESTION_MESSAGE,
```

```
optionType:JOptionPane.OK_CANCEL_OPTION)
dialog = pane.createDialog(null, 'worthless
message')
dialog.show()
```

```
def choice
def lastPane = swingBuilder.optionPane()
choice = lastPane.showOptionDialog(
null,

'worthless message',

'Last chance : Groovy starts in ',

JOptionPane.YES_NO_CANCEL_OPTION,

JOptionPane.QUESTION_MESSAGE,

null,

options as Object[],
```

```
pane.getInputValue()
```

```
println options[choice]
```

## Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- icon
- inheritsPopupMenu
- initialSelectionValue
- initialValue
- inputValue
- inputVerifier
- insets
- layout
- maximumSize
- message
- messageType
- minimumSize
- opaque
- optionType
- options
- preferredSize
- selectionValues
- transferHandler
- value
- verifyInputWhenFocusTarget
- wantsInput

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.OverlayLayout is returned, unless the user passes in a subclass of OverlayLayout as the value argument, in which case the value argument is returned.

## Attributes Events Content Usage Examples

### *SwingBuilder.panel*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JPanel is returned, unless the user passes in a subclass of JPanel as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JPanel

- **UI** <PanelUI> The UI object that implements the Component's LookAndFeel.

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>

- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener

- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

## *SwingBuilder.passwordField*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JPasswordField` is returned, unless the user passes in a subclass of `JPasswordField` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JPasswordField`

- **echoChar** <char> character to display in place of the real characters
- **password** <char[]>

## Declared in `javax.swing.JTextField`

- **action** <Action> the Action instance connected with this ActionEvent source
- **columns** <int> the number of columns preferred for display
- **horizontalAlignment** <int> Set the field alignment to LEFT, CENTER, RIGHT, LEADING (the default) or TRAILING
- **horizontalVisibility** <BoundedRangeModel>
- **scrollOffset** <int>

## Declared in `javax.swing.text.JTextComponent`

- **caret** <Caret> the caret used to select/navigate
- **caretColor** <Color> the color used to render the caret
- **caretPosition** <int> the caret position
- **disabledTextColor** <Color> color used to render disabled text
- **document** <Document> the text document model
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **editable** <boolean> specifies if the text can be edited
- **focusAccelerator** <char> accelerator character used to grab focus
- **highlighter** <Highlighter> object responsible for background highlights
- **keymap** <Keymap> set of key event to action bindings to use
- **margin** <Insets> desired space between the border and text area
- **selectedTextColor** <Color> color used to render selected text
- **selectionColor** <Color> color used to render selection background
- **selectionEnd** <int> ending location of the selection.
- **selectionStart** <int> starting location of the selection.
- **text** <String> the text of this component
  
- **UI** <TextUI>
- **inputMethodRequests** <InputMethodRequests>
- **navigationFilter** <NavigationFilter>
- **preferredScrollableViewportSize** <Dimension>
- **scrollableTracksViewportHeight** <boolean>

- **scrollableTracksViewportWidth** <boolean>
- **selectedText** <String>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>

- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JTextField

- **actionPerformed** (ActionEvent) - part of ActionListener

## Declared in javax.swing.text.JTextComponent

- **caretUpdate** (CaretEvent) - part of CaretListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- action
- alignmentX

- alignmentY
- background
- border
- caret
- caretColor
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- disabledTextColor
- document
- editable
- enabled
- focusAccelerator
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- highlighter
- horizontalAlignment
- inheritsPopupMenu
- inputVerifier
- insets
- keymap
- layout
- margin
- maximumSize
- minimumSize
- opaque
- preferredSize
- selectedTextColor
- selectionColor
- transferHandler
- verifyInputWhenFocusTarget

### ***SwingBuilder.popupMenu***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A `javax.swing.JPopupMenu` is returned, unless the user passes in a subclass of `JPopupMenu` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JPopupMenu`

- **UI** <PopupMenuUI> The UI object that implements the Component's LookAndFeel.
- **borderPainted** <boolean> Is the border of the popup menu painted
- **invoker** <Component> The invoking component for the popup menu
- **label** <String> The label for the popup menu.
- **lightWeightPopupEnabled** <boolean> Determines whether lightweight popups are used when possible
- **selectionModel** <SingleSelectionModel> The selection model for the popup menu
  
- **margin** <Insets>
- **subElements** <MenuElement[]>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in `java.awt.Container`

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>

- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in **java.awt.Component**

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in **javax.swing.JPopupMenu**

- **menuKeyPressed** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyReleased** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyTyped** (MenuKeyEvent) - part of MenuKeyListener
- **popupMenuCanceled** (PopupMenuEvent) - part of PopupMenuListener
- **popupMenuWillBecomeInvisible** (PopupMenuEvent) - part of PopupMenuListener
- **popupMenuWillBecomeVisible** (PopupMenuEvent) - part of PopupMenuListener

## Declared in **javax.swing.JComponent**

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in **java.awt.Container**

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in **java.awt.Component**

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener

- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget
- visible

### *SwingBuilder.progressBar*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

# Generated Object

A `javax.swing.JProgressBar` is returned, unless the user passes in a subclass of `JProgressBar` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JProgressBar`

- **UI** <ProgressBarUI> The UI object that implements the Component's LookAndFeel.
- **borderPainted** <boolean> Whether the progress bar should paint its border.
- **indeterminate** <boolean> Is the progress bar indeterminate (true) or normal (false)?
- **maximum** <int> The progress bar's maximum value.
- **minimum** <int> The progress bar's minimum value.
- **model** <BoundedRangeModel> The data model used by the JProgressBar.
- **orientation** <int> Set the progress bar's orientation.
- **string** <String> Specifies the progress string to paint
- **stringPainted** <boolean> Whether the progress bar should render a string.
- **value** <int> The progress bar's current value.
  
- **percentComplete** <double>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in `java.awt.Container`

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JProgressBar

- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener

- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- borderPainted
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- orientation
- preferredSize
- string
- stringPainted
- transferHandler
- verifyInputWhenFocusTarget

### *SwingBuilder.propertyColumn*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `groovy.model.DefaultTableColumn` is returned, unless the user passes in a subclass of `DefaultTableColumn` as the value argument, in which case the value argument is returned.

## Attributes Declared in `groovy.model.DefaultTableColumn`

- **type** <Class>
- **valueModel** <ValueModel>

## Declared in `javax.swing.table.TableColumn`

- **cellEditor** <TableCellEditor>
- **cellRenderer** <TableCellRenderer>
- **headerRenderer** <TableCellRenderer>
- **headerValue** <Object>
- **identifier** <Object>
- **maxWidth** <int>
- **minWidth** <int>
- **modelIndex** <int>
- **preferredWidth** <int>
- **resizable** <boolean>
- **width** <int>

## Events Declared in `javax.swing.table.TableColumn`

- **propertyChange** (`PropertyChangeEvent`) - part of `PropertyChangeListener`

## Content Usage Examples Access to properties by `propertyName` name

```

data = [[first:'qwer', last:'asdf'],
        [first:'zxcv', last:'tyui'],
        [first:'ghjk', last:'bnm']]

swing = new SwingBuilder()
frame = swing.frame(title:'table test'){
    scrollPane {
        table {
            tableModel( list : data ) {
                propertyColumn(header:'First
Name', propertyName:'first')
                propertyColumn(header:'Last
Name', propertyName:'last')
            }
        }
    }
}
frame.pack()
frame.show()

```

## Access to properties by closure Observable Attributes

- cellEditor
- cellRenderer
- headerRenderer
- headerValue
- identifier
- maxWidth
- minWidth
- modelIndex
- preferredWidth
- propertyChangeListeners
- resizable
- type
- valueModel
- width

## SwingBuilder.radioButton

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JRadioButton is returned, unless the user passes in a subclass of JRadioButton as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.AbstractButton

- **UI** <ButtonUI> The UI object that implements the LookAndFeel.
- **action** <Action> the Action instance connected with this ActionEvent source
- **borderPainted** <boolean> Whether the border should be painted.
- **contentAreaFilled** <boolean> Whether the button should paint the content area or leave it transparent.
- **disabledIcon** <Icon> The disabled icon for the button.
- **disabledSelectedIcon** <Icon> The disabled selection icon for the button.
- **displayedMnemonicIndex** <int> the index into the String to draw the keyboard character mnemonic at
- **focusPainted** <boolean> Whether focus should be painted
- **horizontalAlignment** <int> The horizontal alignment of the icon and text.
- **horizontalTextPosition** <int> The horizontal position of the text relative to the icon.
- **icon** <Icon> The button's default icon
- **iconTextGap** <int> If both the icon and text properties are set, this property defines the space between them.
- **label** <String> Replace by setText(text)
- **margin** <Insets> The space between the button's border and the label.
- **mnemonic** <int> the keyboard character mnemonic
- **model** <ButtonModel> Model that the Button uses.
- **pressedIcon** <Icon> The pressed icon for the button.
- **rolloverEnabled** <boolean> Whether rollover effects should be enabled.
- **rolloverIcon** <Icon> The rollover icon for the button.
- **rolloverSelectedIcon** <Icon> The rollover selected icon for the button.
- **selectedIcon** <Icon> The selected icon for the button.
- **text** <String> The button's text.
- **verticalAlignment** <int> The vertical alignment of the icon and text.
- **verticalTextPosition** <int> The vertical position of the text relative to the icon.
  
- **actionCommand** <String>
- **multiClickThreshold** <long>
- **selected** <boolean>
- **selectedObjects** <Object[]>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.

- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.AbstractButton

- **itemStateChanged** (ItemEvent) - part of ItemListener
- **actionPerformed** (ActionEvent) - part of ActionListener
- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- action
- alignmentX
- alignmentY
- background
- border
- borderPainted
- component
- componentCount
- componentPopupMenu
- components
- containerListeners

- contentAreaFilled
- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusPainted
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- margin
- maximumSize
- minimumSize
- mnemonic
- model
- opaque
- preferredSize
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

### ***SwingBuilder.radioButtonMenuItem***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

# Generated Object

A `javax.swing.JRadioButtonMenuItem` is returned, unless the user passes in a subclass of `JRadioButtonMenuItem` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JMenuItem`

- **accelerator** <KeyStroke> The keystroke combination which will invoke the `JMenuItem`'s actionlisteners without navigating the menu hierarchy
- **armed** <boolean> Mouse release will fire an action event
- **subElements** <MenuElement[]>

## Declared in `javax.swing.AbstractButton`

- **UI** <ButtonUI> The UI object that implements the LookAndFeel.
- **action** <Action> the Action instance connected with this ActionEvent source
- **borderPainted** <boolean> Whether the border should be painted.
- **contentAreaFilled** <boolean> Whether the button should paint the content area or leave it transparent.
- **disabledIcon** <Icon> The disabled icon for the button.
- **disabledSelectedIcon** <Icon> The disabled selection icon for the button.
- **displayedMnemonicIndex** <int> the index into the String to draw the keyboard character mnemonic at
- **focusPainted** <boolean> Whether focus should be painted
- **horizontalAlignment** <int> The horizontal alignment of the icon and text.
- **horizontalTextPosition** <int> The horizontal position of the text relative to the icon.
- **icon** <Icon> The button's default icon
- **iconTextGap** <int> If both the icon and text properties are set, this property defines the space between them.
- **label** <String> Replace by `setText(text)`
- **margin** <Insets> The space between the button's border and the label.
- **mnemonic** <int> the keyboard character mnemonic
- **model** <ButtonModel> Model that the Button uses.
- **pressedIcon** <Icon> The pressed icon for the button.
- **rolloverEnabled** <boolean> Whether rollover effects should be enabled.
- **rolloverIcon** <Icon> The rollover icon for the button.
- **rolloverSelectedIcon** <Icon> The rollover selected icon for the button.
- **selectedIcon** <Icon> The selected icon for the button.
- **text** <String> The button's text.
- **verticalAlignment** <int> The vertical alignment of the icon and text.
- **verticalTextPosition** <int> The vertical position of the text relative to the icon.
  
- **actionCommand** <String>
- **multiClickThreshold** <long>
- **selected** <boolean>
- **selectedObjects** <Object[]>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the `JPopupMenu` is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.

- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JMenuItem

- **menuKeyPressed** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyReleased** (MenuKeyEvent) - part of MenuKeyListener
- **menuKeyTyped** (MenuKeyEvent) - part of MenuKeyListener
- **menuDragMouseDragged** (MenuDragMouseEvent) - part of MenuDragMouseListener

- **menuDragMouseEntered** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseExited** (MenuDragMouseEvent) - part of MenuDragMouseListener
- **menuDragMouseReleased** (MenuDragMouseEvent) - part of MenuDragMouseListener

## Declared in javax.swing.AbstractButton

- **itemStateChanged** (ItemEvent) - part of ItemListener
- **actionPerformed** (ActionEvent) - part of ActionListener
- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- accelerator
- action
- alignmentX
- alignmentY
- background

- border
- borderPainted
- componentCount
- componentPopupMenu
- components
- containerListeners
- contentAreaFilled
- disabledIcon
- disabledSelectedIcon
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusPainted
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- margin
- maximumSize
- minimumSize
- mnemonic
- model
- opaque
- preferredSize
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

*SwingBuilder.raisedBevelBorder*

## Generated Object

A javax.swing.border.BevelBorder with type BevelBorder.RAISED, a border that looks recessed into the panel.

# Value Argument

`raisedBevelBorder()` takes no value arguments.

## Attributes

If either `shadow:` or `highlight:` are specified, both must be specified.

If any of `shadowInner:`, `shadowOuter:`, `highlightInner:`, or `highlightOuter` are specified, all must be specified.

The attributes `shadow:` and `highlight:` are mutually exclusive to `shadowInner:`, `shadowOuter:`, `highlightInner:`, and `highlightOuter:`.

- **highlight** <Color> The color that is to be painted on the highlight side of the bevel.
- **shadow** <Color> The color that is to be painted on the shadow side of the bevel.
- **highlightInner** <Color> The inner color that is to be painted on the highlight side of the bevel.
- **highlightOuter** <Color> The outer color that is to be painted on the highlight side of the bevel.
- **shadowInner** <Color> The inner color that is to be painted on the shadow side of the bevel.
- **shadowOuter** <Color> The outer color that is to be painted on the shadow side of the bevel.
- **parent** <boolean> Whether or not this border is to be added to the parent `JComponent`.

## Content

`raisedBevelBorder()` is a leaf node, no child content is allowed.

## Examples

```
panel (border:raisedBevelBorder())
```

```
panel {  
    raisedBevelBorder (shadow:Color.RED,  
highlight:Color.PINK, parent:true)  
}
```

```
raisedBevelBorder(  
    shadowOuter:Color.BLACK,  
shadowInner:Color.GRAY,  
    highlightOuter:Color.RED,  
highlightInner:Color.PINK)
```

*SwingBuilder.raisedEtchedBorder*

## Generated Object

A `javax.swing.border.EtchedBorder` with type of `EtchedBorder.RAISED`, a border that looks like a groove has been raised out of the panel.

## Value Argument

`raisedEtchedBorder()` takes no value arguments.

## Attributes

If any of `shadow:` or `highlight:` are specified, all must be specified.

- **highlight** <Color> The color that is to be painted on the highlight side of the raised etch.
- **shadow** <Color> The color that is to be painted on the shadow side of the raised etch.
- **parent** <boolean> Whether or not this border is to be added to the parent `JComponent`.

## Content

`raisedEtchedBorder()` is a leaf node, no child content is allowed.

## Examples

```
panel (border: raisedEtchedBorder ( ) )

panel {
    raisedEtchedBorder ( shadow: Color . RED ,
highlight: Color . PINK , parent: true )
}
```

### *SwingBuilder.scrollBar*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from `BeanInfo`)

## Generated Object

A `javax.swing.JScrollBar` is returned, unless the user passes in a subclass of `JScrollBar` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JScrollBar`

- **UI** <ScrollBarUI> The UI object that implements the Component's LookAndFeel
- **blockIncrement** <int> The scrollbar's block increment.
- **maximum** <int> The scrollbar's maximum value.
- **minimum** <int> The scrollbar's minimum value.
- **model** <BoundedRangeModel> The scrollbar's BoundedRangeModel.
- **orientation** <int> The scrollbar's orientation.
- **unitIncrement** <int> The scrollbar's unit increment.
- **value** <int> The scrollbar's current value.
- **valueIsAdjusting** <boolean> True if the scrollbar thumb is being dragged.
- **visibleAmount** <int> The amount of the view that is currently visible.

## Declared in javax.swing.JComponent

- **autoScrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>

- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JScrollBar

- **adjustmentValueChanged** (AdjustmentEvent) - part of AdjustmentListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener

- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- blockIncrement
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- model
- opaque
- orientation
- preferredSize
- transferHandler
- unitIncrement
- verifyInputWhenFocusTarget

### ***SwingBuilder.scrollPane***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JScrollPane` is returned, unless the user passes in a subclass of `JScrollPane` as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JScrollPane

- **UI** <ScrollPaneUI> The UI object that implements the Component's LookAndFeel.
- **columnHeader** <JViewport> The column header child for this scrollpane
- **horizontalScrollBar** <JScrollBar> The horizontal scrollbar.
- **horizontalScrollBarPolicy** <int> The scrollpane scrollbar policy
- **rowHeader** <JViewport> The row header child for this scrollpane
- **verticalScrollBar** <JScrollBar> The vertical scrollbar.
- **verticalScrollBarPolicy** <int> The scrollpane vertical scrollbar policy
- **viewport** <JViewport> The viewport child for this scrollpane
- **viewportBorder** <Border> The border around the viewport.
- **wheelScrollingEnabled** <boolean> Flag for enabling/disabling mouse wheel scrolling
  
- **viewportBorderBounds** <Rectangle>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>

- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

# Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- columnHeader
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalScrollBar
- horizontalScrollBarPolicy
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- rowHeader
- transferHandler
- verifyInputWhenFocusTarget
- verticalScrollBar
- verticalScrollBarPolicy
- viewport
- viewportBorder
- wheelScrollingEnabled

## ***SwingBuilder.separator***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

# Generated Object

A `javax.swing.JSeparator` is returned, unless the user passes in a subclass of `JSeparator` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JSeparator`

- **UI** `<SeparatorUI>` The UI object that implements the Component's LookAndFeel.
- **orientation** `<int>` The orientation of the separator.

## Declared in `javax.swing.JComponent`

- **autoscrolls** `<boolean>` Determines if this component automatically scrolls its contents when dragged.
- **border** `<Border>` The component's border.
- **componentPopupMenu** `<JPopupMenu>` Popup to show
- **debugGraphicsOptions** `<int>` Diagnostic options for graphics operations.
- **inheritsPopupMenu** `<boolean>` Whether or not the `JPopupMenu` is inherited
- **inputVerifier** `<InputVerifier>` The component's input verifier.
- **opaque** `<boolean>` The component's opacity
- **toolTipText** `<String>` The text to display in a tool tip.
- **transferHandler** `<TransferHandler>` Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** `<boolean>` Whether the Component verifies input before accepting focus.
  
- **UIClassID** `<String>`
- **accessibleContext** `<AccessibleContext>`
- **actionMap** `<ActionMap>`
- **doubleBuffered** `<boolean>`
- **graphics** `<Graphics>`
- **height** `<int>`
- **managingFocus** `<boolean>`
- **nextFocusableComponent** `<Component>`
- **optimizedDrawingEnabled** `<boolean>`
- **paintingTile** `<boolean>`
- **registeredKeyStrokes** `<KeyStroke[]>`
- **requestFocusEnabled** `<boolean>`
- **rootPane** `<JRootPane>`
- **topLevelAncestor** `<Container>`
- **validateRoot** `<boolean>`
- **visibleRect** `<Rectangle>`
- **width** `<int>`
- **x** `<int>`
- **y** `<int>`

## Declared in `java.awt.Container`

- **alignmentX** `<float>`
- **alignmentY** `<float>`
- **componentCount** `<int>`
- **components** `<Component[]>`
- **focusCycleRoot** `<boolean>`
- **focusTraversalPolicy** `<FocusTraversalPolicy>`
- **focusTraversalPolicyProvider** `<boolean>`
- **focusTraversalPolicySet** `<boolean>`
- **insets** `<Insets>`
- **layout** `<LayoutManager>`

- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX

- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- orientation
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

### *SwingBuilder.slider*

## Generated Object

A javax.swing.JSlider is returned, unless the user passes in a subclass of JSlider as the value argument, in which case the value argument is returned.

## Attributes

See also: [JComponent](#)

- extent <int> Size of the range covered by the knob.
- inverted <boolean> If true reverses the slider values from their normal order
- labelTable <Dictionary> Specifies what labels will be drawn for any given value.
- majorTickSpacing <int> Sets the number of values between major tick marks.
- maximum <int> The sliders maximum value.
- minimum <int> The sliders minimum value.
- minorTickSpacing <int> Sets the number of values between minor tick marks.
- model <BoundedRangeModel> The sliders BoundedRangeModel.
- orientation <int> Set the scrollbars orientation to either VERTICAL or HORIZONTAL.
- paintLabels <boolean> If true labels are painted on the slider.
- paintTicks <boolean> If true tick marks are painted on the slider.
- paintTrack <boolean> If true, the track is painted on the slider.
- snapToTicks <boolean> If true snap the knob to the nearest tick mark.

- **value** <int> The sliders current value.

## Content

No child content is accepted in a slider.

## Examples

//TODO

## Observable Attributes

The following properties are observable and can be used as sourceProperty in bind():

- **value** - via synthetic observation
- **inverted**
- **labelTable**
- **majorTickSpacing**
- **maximum**
- **minimum**
- **minorTickSpacing**
- **model**
- **orientation**
- **paintLabels**
- **paintTicks**
- **paintTrack**
- **snapToTicks**

### *SwingBuilder.spinner*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JSpinner is returned, unless the user passes in a subclass of JSpinner as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JSpinner

- **editor** <JComponent> JComponent that displays the current value of the model
- **model** <SpinnerModel> Model that represents the value of this spinner.
- **UI** <SpinnerUI>
- **nextValue** <Object>
- **previousValue** <Object>
- **value** <Object>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>

- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JSpinner

- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components

- containerListeners
- editor
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- model
- opaque
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

#### ***SwingBuilder.spinnerDateModel***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A javax.swing.SpinnerDateModel is returned, unless the user passes in a subclass of SpinnerDateModel as the value argument, in which case the value argument is returned.

## **Attributes Declared in javax.swing.SpinnerDateModel**

- **calendarField** <int>
- **date** <Date>
- **end** <Comparable>
- **nextValue** <Object>
- **previousValue** <Object>
- **start** <Comparable>
- **value** <Object>

## **Events Declared in javax.swing.AbstractSpinnerModel**

- **stateChanged** (ChangeEvent) - part of ChangeListener

## **Content Usage Examples**

### *SwingBuilder.spinnerListModel*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.SpinnerListModel is returned, unless the user passes in a subclass of SpinnerListModel as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.SpinnerListModel

- **list** <List>
- **nextValue** <Object>
- **previousValue** <Object>
- **value** <Object>

## Events Declared in javax.swing.AbstractSpinnerModel

- **stateChanged** (ChangeEvent) - part of ChangeListener

## Content Usage Examples

### *SwingBuilder.spinnerNumberModel*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.SpinnerNumberModel is returned, unless the user passes in a subclass of SpinnerNumberModel as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.SpinnerNumberModel

- **maximum** <Comparable>
- **minimum** <Comparable>
- **nextValue** <Object>
- **number** <Number>
- **previousValue** <Object>
- **stepSize** <Number>
- **value** <Object>

## Events Declared in javax.swing.AbstractSpinnerModel

- **stateChanged** (ChangeEvent) - part of ChangeListener

# Content Usage Examples

## *SwingBuilder.splitPane*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JSplitPane` is returned, unless the user passes in a subclass of `JSplitPane` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JSplitPane`

- **UI** <SplitPaneUI> The L&F object that renders this component.
- **bottomComponent** <Component> The component below, or to the right of the divider.
- **continuousLayout** <boolean> Whether the child components are continuously redisplayed and laid out during user intervention.
- **dividerLocation** <int> The location of the divider.
- **dividerSize** <int> The size of the divider.
- **lastDividerLocation** <int> The last location the divider was at.
- **leftComponent** <Component> The component to the left (or above) the divider.
- **minimumDividerLocation** <int> The minimum location of the divider from the L&F.
- **oneTouchExpandable** <boolean> UI widget on the divider to quickly expand/collapse the divider.
- **orientation** <int> The orientation, or how the splitter is divided.
- **resizeWeight** <double> Specifies how to distribute extra space when the split pane resizes.
- **rightComponent** <Component> The component to the right (or below) the divider.
- **topComponent** <Component> The component above, or to the left of the divider.
- **maximumDividerLocation** <int>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>

- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- continuousLayout
- dividerLocation
- dividerSize
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- lastDividerLocation
- layout
- maximumSize

- `minimumSize`
- `oneTouchExpandable`
- `opaque`
- `orientation`
- `preferredSize`
- `resizeWeight`
- `transferHandler`
- `verifyInputWhenFocusTarget`

#### *SwingBuilder.springLayout*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.SpringLayout` is returned, unless the user passes in a subclass of `SpringLayout` as the value argument, in which case the value argument is returned.

## Attributes Events Content Usage Examples

#### *SwingBuilder.tabbedPane*

## Generated Object

A `JTabbedPane`

## Value Argument

The `tabbedPane()` node accepts no value argument.

## Attributes

In addition to [JComponent](#) attributes, `tabbedPane` supports these additional attributes:

- **model** `<SingleSelectionModel>` The model that tracks the tab selection. not normally set by the end user.
- **tabPlacement** `<int>` One of `JTabbedPane.TOP`, `JTabbedPane.BOTTOM`, `JTabbedPane.LEFT`, or `JTabbedPane.RIGHT`, indicates where the tabs should be placed.
- **tabLayoutPolicy** `<int>` One of `JTabbedPane.WRAP_TAB_LAYOUT` or `JTabbedPane.SCROLL_TAB_LAYOUT`. Determines the behavior of the tabs when there are too many for one line. Either multiple lines, or a single line with scroll controls (respectively)
- **selectedIndex** `<int>` The tab index to be selected initially. Do not combine with `selectedComponent`
- **selectedComponent** `<Component>` The tab representing the component to be selected initially. Do not combine with `selectedIndex`.

## Content

All immediate children of the `JTabbedPane` that are `Components` are added to the tabbed pane as tabs. In addition,

these children also can have the following attributes, which apply to the tab representing those components

- **title** <String> The text title on the tab. If this is missing the name property of the component is used (which defaults to an empty string)
- **tabIcon** <Icon> The icon to be used on the tab, usually to the left of the text.
- **tabDisabledIcon** <Icon> The icon to be displayed when the tab is disabled. If tabIcon: is specified but tabDisabledIcon isn't then a disable icon will be automatically generated base on the Look and Feel.
- **tabToolTip** <String> The toolTip to be displayed when the mouse is hovering over the tab. (note that this is different than the component in the tab, use tooltip: for the component itself)
- **tabBackground** <Color> The background color of the tab. If this isn't specified a Look and Feel dependent value will be provided. The Look and Feel may ignore this (Windows Vista does)
- **tabForeground** <Color> The foreground text color of the tab. If this isn't specified a Look and Feel dependent value will be provided. The Look and Feel may ignore this (Windows Vista does)
- **tabEnabled** <boolean> A flag to indicate whether or not this tab should be considered activated and accessible.
- **tabMnemonic** <int or String or char> The mnemonic of the tab to be used for keyboard navigation. Do not combine with tabDisplayedMnemonicIndex unless the characters match up. Character must be in title/upper case
- **tabDisplayedMnemonicIndex** <int> The character index in the title to use for the mnemonic. Useful when a more prominent character should be used, like "Stop Operation", a mnemonic of O would ordinarily underline the 'o' in Stop.

## Examples

```
tabbedPane(tabPlacement: JTabbedPane.LEFT) {  
    label('One', title:'One',  
    tabToolTip:'Uno!')  
    label('Green', title:'Green',  
    tabBackground:java.awt.Color.GREEN)  
    label('Stop Operation', title:'Stop  
Operation', tabMnemonic:'O')  
    label('Stop Operation', title:'Stop  
Operation', tabDisplayedMnemonicIndex:5)  
}
```

```

tabbedPane() {
    panel(name: 'One') {
        label('One')
    }
    panel(name: 'Two') {
        label('Two')
    }
}

```

#### *SwingBuilder.table*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JTable` is returned, unless the user passes in a subclass of `JTable` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JTable`

- **UI** <TableUI> The UI object that implements the Component's LookAndFeel.
- **autoCreateColumnsFromModel** <boolean> Automatically populates the columnModel when a new TableModel is submitted.
- **autoResizeMode** <int> Whether the columns should adjust themselves automatically.
- **cellEditor** <TableCellEditor> The table's active cell editor.
- **cellSelectionEnabled** <boolean> Select a rectangular region of cells rather than rows or columns.
- **columnModel** <TableColumnModel> The object governing the way columns appear in the view.
- **columnSelectionAllowed** <boolean> If true, an entire column is selected for each selected cell.
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **gridColor** <Color> The grid color.
- **intercellSpacing** <Dimension> The spacing between the cells, drawn in the background color of the JTable.
- **model** <TableModel> The model that is the source of the data for this view.
- **preferredScrollableViewportSize** <Dimension> The preferred size of the viewport.
- **rowMargin** <int> The amount of space between cells.
- **rowSelectionAllowed** <boolean> If true, an entire row is selected for each selected cell.
- **selectionBackground** <Color> A default background color for selected cells.

- **selectionForeground** <Color> A default foreground color for selected cells.
- **selectionModel** <ListSelectionModel> The selection model for rows.
- **showHorizontalLines** <boolean> Whether horizontal lines should be drawn in between the cells.
- **showVerticalLines** <boolean> Whether vertical lines should be drawn in between the cells.
- **tableHeader** <JTableHeader> The JTableHeader instance which renders the column headers.
- **columnCount** <int>
- **editing** <boolean>
- **editingColumn** <int>
- **editingRow** <int>
- **editorComponent** <Component>
- **rowCount** <int>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectedColumn** <int>
- **selectedColumnCount** <int>
- **selectedColumns** <int[]>
- **selectedRow** <int>
- **selectedRowCount** <int>
- **selectedRows** <int[]>
- **surrendersFocusOnKeystroke** <boolean>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener

- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- autoCreateColumnsFromModel
- autoResizeMode
- background
- border
- cellEditor
- cellSelectionEnabled
- columnModel
- columnSelectionAllowed
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- gridColor
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- model
- opaque
- preferredSize
- rowMargin
- rowSelectionAllowed
- selectionBackground
- selectionForeground
- selectionModel
- showHorizontalLines
- showVerticalLines
- tableHeader
- transferHandler

- `verifyInputWhenFocusTarget`

### ***SwingBuilder.tableColumn***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A `javax.swing.table.TableColumn` is returned, unless the user passes in a subclass of `TableColumn` as the value argument, in which case the value argument is returned.

## **Attributes Declared in `javax.swing.table.TableColumn`**

- **`cellEditor`** <`TableCellEditor`>
- **`cellRenderer`** <`TableCellRenderer`>
- **`headerRenderer`** <`TableCellRenderer`>
- **`headerValue`** <`Object`>
- **`identifier`** <`Object`>
- **`maxWidth`** <`int`>
- **`minWidth`** <`int`>
- **`modelIndex`** <`int`>
- **`preferredWidth`** <`int`>
- **`resizable`** <`boolean`>
- **`width`** <`int`>

## **Events Declared in `javax.swing.table.TableColumn`**

- **`propertyChange`** (`PropertyChangeEvent`) - part of `PropertyChangeListener`

## **Content Usage Examples Observable Attributes**

- `cellEditor`
- `cellRenderer`
- `headerRenderer`
- `headerValue`
- `identifier`
- `maxWidth`
- `minWidth`
- `modelIndex`
- `preferredWidth`
- `propertyChangeListeners`
- `resizable`
- `width`

### ***SwingBuilder.tableLayout***

- [Generated Object](#)
- [Attributes](#)

- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `groovy.swing.impl.TableLayout` is returned, unless the user passes in a subclass of `TableLayout` as the value argument, in which case the value argument is returned.

## Attributes Declared in `groovy.swing.impl.TableLayout`

- **cellpadding** <int>

## Declared in `javax.swing.JPanel`

- **UI** <PanelUI> The UI object that implements the Component's LookAndFeel.

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in `java.awt.Container`

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener

- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- cellpadding
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- transferHandler
- verifyInputWhenFocusTarget

### *SwingBuilder.tableModel*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A groovy.model.DefaultTableModel is returned, unless the user passes in a subclass of DefaultTableModel as the value argument, in which case the value argument is returned.

## Attributes Declared in groovy.model.DefaultTableModel

- **columnCount** <int>
- **columnList** <List>
- **columnModel** <TableColumnModel>
- **rowCount** <int>
- **rowModel** <ValueModel>
- **rowsModel** <ValueModel>

## Events Declared in javax.swing.table.AbstractTableModel

- **tableChanged** (TableModelEvent) - part of TableModelListener

## Content Usage Examples

### *SwingBuilder.td*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A groovy.swing.impl.TableLayoutCell is returned, unless the user passes in a subclass of TableLayoutCell as the value argument, in which case the value argument is returned.

## Attributes Declared in groovy.swing.impl.TableLayoutCell

- **align** <String>
- **colfill** <boolean>
- **colspan** <int>
- **component** <Component>
- **constraints** <GridBagConstraints>
- **rowfill** <boolean>
- **rowspan** <int>
- **valign** <String>

## Events Content Usage Examples

### *SwingBuilder.textArea*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

# Generated Object

A `javax.swing.JTextArea` is returned, unless the user passes in a subclass of `JTextArea` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JTextArea`

- **columns** <int> the number of columns preferred for display
- **lineWrap** <boolean> should lines be wrapped
- **rows** <int> the number of rows preferred for display
- **tabSize** <int> the number of characters to expand tabs to
- **wrapStyleWord** <boolean> should wrapping occur at word boundaries
  
- **lineCount** <int>

## Declared in `javax.swing.text.JTextComponent`

- **caret** <Caret> the caret used to select/navigate
- **caretColor** <Color> the color used to render the caret
- **caretPosition** <int> the caret position
- **disabledTextColor** <Color> color used to render disabled text
- **document** <Document> the text document model
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **editable** <boolean> specifies if the text can be edited
- **focusAccelerator** <char> accelerator character used to grab focus
- **highlighter** <Highlighter> object responsible for background highlights
- **keymap** <Keymap> set of key event to action bindings to use
- **margin** <Insets> desired space between the border and text area
- **selectedTextColor** <Color> color used to render selected text
- **selectionColor** <Color> color used to render selection background
- **selectionEnd** <int> ending location of the selection.
- **selectionStart** <int> starting location of the selection.
- **text** <String> the text of this component
  
- **UI** <TextUI>
- **inputMethodRequests** <InputMethodRequests>
- **navigationFilter** <NavigationFilter>
- **preferredScrollableViewportSize** <Dimension>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectedText** <String>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the `JPopupMenu` is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>

- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.text.JTextComponent

- **caretUpdate** (CaretEvent) - part of CaretListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- caret
- caretColor
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- disabledTextColor
- document
- editable
- enabled
- focusAccelerator
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable

- font
- foreground
- highlighter
- inheritsPopupMenu
- inputVerifier
- insets
- keymap
- layout
- lineWrap
- margin
- maximumSize
- minimumSize
- opaque
- preferredSize
- selectedTextColor
- selectionColor
- tabSize
- transferHandler
- verifyInputWhenFocusTarget
- wrapStyleWord

*SwingBuilder.textField*

## Generated Object

A `javax.swing.JTextField` is returned, unless the user passes in a subclass of `JTextField` as the value argument, in which case the value argument is returned.

## Value Argument

The value argument can be either a `JTextField`, or a `String`. If the value argument is a string it is treated as though it were passed in as the text attribute.

## Attributes

See also: [JComponent](#)

## Declared in JTextField

- `action` `<Action>` the Action instance connected with this ActionEvent source
- `columns` `<int>` the number of columns preferred for display
- `horizontalAlignment` `<int>` Set the field alignment to LEFT, CENTER, RIGHT, LEADING (the default) or TRAILING
- `scrollOffset` `<int>` scrollOffset

## Declared in javax.swing.text.JTextComponent

- `caret` `<javax.swing.text.Caret>` the caret used to select/navigate
- `caretColor` `<Color>` the color used to render the caret
- `caretPosition` `<int>` the caret position
- `disabledTextColor` `<Color>` color used to render disabled text
- `document` `<javax.swing.text.Document>` the text document model
- `editable` `<boolean>` specifies if the text can be edited
- `focusAccelerator` `<char>` accelerator character used to grab focus

- highlighter <javax.swing.text.Highlighter> object responsible for background highlights
- margin <Insets> desired space between the border and text area
- selectedText <String> selectedText
- selectedTextColor <Color> color used to render selected text
- selectionColor <Color> color used to render selection background
- selectionEnd <int> ending location of the selection.
- selectionStart <int> starting location of the selection.
- text <String> the text of this component

## Content

No child content is accepted in a textField.

## Usage

The action connected to the textField is usually fired when the 'Enter' key is pressed by the user when the textField has keyboard focus.

## Examples

//TODO

## Observable Attributes

- text - via syntetic observation
- caret
- caretColor
- disabledTextColor
- document
- dragEnabled
- editable
- focusAccelerator
- highlighter
- keymap
- margin
- selectedTextColor
- selectionColor

### *SwingBuilder.textPane*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.JTextPane is returned, unless the user passes in a subclass of JTextPane as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.JTextPane

- **characterAttributes** <AttributeSet>
- **inputAttributes** <MutableAttributeSet>
- **logicalStyle** <Style>
- **paragraphAttributes** <AttributeSet>
- **styledDocument** <StyledDocument>

## Declared in javax.swing.JEditorPane

- **contentType** <String> the type of content
- **editorKit** <EditorKit> the currently installed kit for handling content
- **page** <URL> the URL used to set content

## Declared in javax.swing.text.JTextComponent

- **caret** <Caret> the caret used to select/navigate
- **caretColor** <Color> the color used to render the caret
- **caretPosition** <int> the caret position
- **disabledTextColor** <Color> color used to render disabled text
- **document** <Document> the text document model
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **editable** <boolean> specifies if the text can be edited
- **focusAccelerator** <char> accelerator character used to grab focus
- **highlighter** <Highlighter> object responsible for background highlights
- **keymap** <Keymap> set of key event to action bindings to use
- **margin** <Insets> desired space between the border and text area
- **selectedTextColor** <Color> color used to render selected text
- **selectionColor** <Color> color used to render selection background
- **selectionEnd** <int> ending location of the selection.
- **selectionStart** <int> starting location of the selection.
- **text** <String> the text of this component
  
- **UI** <TextUI>
- **inputMethodRequests** <InputMethodRequests>
- **navigationFilter** <NavigationFilter>
- **preferredScrollableViewportSize** <Dimension>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectedText** <String>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>

- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JEditorPane

- **hyperlinkUpdate** (HyperlinkEvent) - part of HyperlinkListener

## Declared in javax.swing.text.JTextComponent

- **caretUpdate** (CaretEvent) - part of CaretListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener

- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- alignmentX
- alignmentY
- background
- border
- caret
- caretColor
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- disabledTextColor
- document
- editable
- editorKit
- enabled
- focusAccelerator
- focusCycleRoot
- focusTraversalKeys

- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- highlighter
- inheritsPopupMenu
- inputVerifier
- insets
- keymap
- layout
- margin
- maximumSize
- minimumSize
- opaque
- page
- preferredSize
- selectedTextColor
- selectionColor
- transferHandler
- verifyInputWhenFocusTarget

#### *SwingBuilder.titledBorder*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## Generated Object

A javax.swing.border.TitledBorder is returned, unless the user passes in a subclass of TitledBorder as the value argument, in which case the value argument is returned.

## Attributes Declared in javax.swing.border.TitledBorder

- **border** <Border>
- **title** <String>
- **titleColor** <Color>
- **titleFont** <Font>
- **titleJustification** <int>
- **titlePosition** <int>

## Declared in javax.swing.border.AbstractBorder

- **borderOpaque** <boolean>

## Events Content Usage Examples

#### *SwingBuilder.toggleButton*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JToggleButton` is returned, unless the user passes in a subclass of `JToggleButton` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.AbstractButton`

- **UI** `<ButtonUI>` The UI object that implements the `LookAndFeel`.
- **action** `<Action>` the Action instance connected with this ActionEvent source
- **borderPainted** `<boolean>` Whether the border should be painted.
- **contentAreaFilled** `<boolean>` Whether the button should paint the content area or leave it transparent.
- **disabledIcon** `<Icon>` The disabled icon for the button.
- **disabledSelectedIcon** `<Icon>` The disabled selection icon for the button.
- **displayedMnemonicIndex** `<int>` the index into the String to draw the keyboard character mnemonic at
- **focusPainted** `<boolean>` Whether focus should be painted
- **horizontalAlignment** `<int>` The horizontal alignment of the icon and text.
- **horizontalTextPosition** `<int>` The horizontal position of the text relative to the icon.
- **icon** `<Icon>` The button's default icon
- **iconTextGap** `<int>` If both the icon and text properties are set, this property defines the space between them.
- **label** `<String>` Replace by `setText(text)`
- **margin** `<Insets>` The space between the button's border and the label.
- **mnemonic** `<int>` the keyboard character mnemonic
- **model** `<ButtonModel>` Model that the Button uses.
- **pressedIcon** `<Icon>` The pressed icon for the button.
- **rolloverEnabled** `<boolean>` Whether rollover effects should be enabled.
- **rolloverIcon** `<Icon>` The rollover icon for the button.
- **rolloverSelectedIcon** `<Icon>` The rollover selected icon for the button.
- **selectedIcon** `<Icon>` The selected icon for the button.
- **text** `<String>` The button's text.
- **verticalAlignment** `<int>` The vertical alignment of the icon and text.
- **verticalTextPosition** `<int>` The vertical position of the text relative to the icon.
- **actionCommand** `<String>`
- **multiClickThreshold** `<long>`
- **selected** `<boolean>`
- **selectedObjects** `<Object[]>`

## Declared in `javax.swing.JComponent`

- **autoscrolls** `<boolean>` Determines if this component automatically scrolls its contents when dragged.
- **border** `<Border>` The component's border.
- **componentPopupMenu** `<JPopupMenu>` Popup to show
- **debugGraphicsOptions** `<int>` Diagnostic options for graphics operations.
- **inheritsPopupMenu** `<boolean>` Whether or not the `JPopupMenu` is inherited
- **inputVerifier** `<InputVerifier>` The component's input verifier.

- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.AbstractButton

- **itemStateChanged** (ItemEvent) - part of ItemListener
- **actionPerformed** (ActionEvent) - part of ActionListener

- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- action
- alignmentX
- alignmentY
- background
- border
- borderPainted
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- contentAreaFilled
- disabledIcon

- disabledSelectedIcon
- displayedMnemonicIndex
- enabled
- focusCycleRoot
- focusPainted
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- horizontalAlignment
- horizontalTextPosition
- icon
- iconTextGap
- inheritsPopupMenu
- inputVerifier
- insets
- label
- layout
- margin
- maximumSize
- minimumSize
- mnemonic
- model
- opaque
- preferredSize
- pressedIcon
- rolloverEnabled
- rolloverIcon
- rolloverSelectedIcon
- selectedIcon
- text
- transferHandler
- verifyInputWhenFocusTarget
- verticalAlignment
- verticalTextPosition

#### ***SwingBuilder.toolBar***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A `javax.swing.JToolBar` is returned, unless the user passes in a subclass of `JToolBar` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JToolBar`

- **UI** <ToolBarUI> The UI object that implements the Component's LookAndFeel.
- **borderPainted** <boolean> Does the tool bar paint its borders?
- **floatable** <boolean> Can the tool bar be made to float by the user?
- **margin** <Insets> The margin between the tool bar's border and contents
- **orientation** <int> The current orientation of the tool bar
- **rollover** <boolean> Will draw rollover button borders in the toolbar.

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
  
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in `java.awt.Container`

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>

- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- background
- border
- borderPainted
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- floatable
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- margin
- maximumSize
- minimumSize
- opaque
- orientation
- preferredSize
- rollover
- transferHandler
- verifyInputWhenFocusTarget

### ***SwingBuilder.tr***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)

(Automatically generated from BeanInfo)

## **Generated Object**

A groovy.swing.impl.TableLayoutRow is returned, unless the user passes in a subclass of TableLayoutRow as the value argument, in which case the value argument is returned.

## **Attributes Declared in groovy.swing.impl.TableLayoutRow**

- `rowIndex` <int>

# Events Content Usage Examples

## *SwingBuilder.tree*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JTree` is returned, unless the user passes in a subclass of `JTree` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JTree`

- **UI** <TreeUI> The UI object that implements the Component's LookAndFeel.
- **anchorSelectionPath** <TreePath> Anchor selection path
- **cellEditor** <TreeCellEditor> The cell editor. A null value implies the tree cannot be edited.
- **cellRenderer** <TreeCellRenderer> The TreeCellRenderer that will be used to draw each cell.
- **dragEnabled** <boolean> determines whether automatic drag handling is enabled
- **editable** <boolean> Whether the tree is editable.
- **expandsSelectedPaths** <boolean> Indicates whether changes to the selection should make the parent of the path visible.
- **invokesStopCellEditing** <boolean> Determines what happens when editing is interrupted, selecting another node in the tree, a change in the tree's data, or some other means.
- **largeModel** <boolean> Whether the UI should use a large model.
- **leadSelectionPath** <TreePath> Lead selection path
- **model** <TreeModel> The TreeModel that will provide the data.
- **rootVisible** <boolean> Whether or not the root node from the TreeModel is visible.
- **rowHeight** <int> The height of each cell.
- **scrollsOnExpand** <boolean> Indicates if a node descendant should be scrolled when expanded.
- **selectionModel** <TreeSelectionModel> The tree's selection model.
- **showsRootHandles** <boolean> Whether the node handles are to be displayed.
- **toggleClickCount** <int> Number of clicks before a node will expand/collapse.
- **visibleRowCount** <int> The number of rows that are to be displayed.
  
- **editing** <boolean>
- **editingPath** <TreePath>
- **fixedRowHeight** <boolean>
- **lastSelectedPathComponent** <Object>
- **leadSelectionRow** <int>
- **maxSelectionRow** <int>
- **minSelectionRow** <int>
- **preferredScrollableViewportSize** <Dimension>
- **rowCount** <int>
- **scrollableTracksViewportHeight** <boolean>
- **scrollableTracksViewportWidth** <boolean>
- **selectionCount** <int>

- **selectionEmpty** <boolean>
- **selectionPath** <TreePath>
- **selectionPaths** <TreePath[]>
- **selectionRows** <int[]>

## Declared in javax.swing.JComponent

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JTree

- **treeCollapsed** (TreeExpansionEvent) - part of TreeExpansionListener
- **treeExpanded** (TreeExpansionEvent) - part of TreeExpansionListener
- **valueChanged** (TreeSelectionEvent) - part of TreeSelectionListener
- **treeWillCollapse** (TreeExpansionEvent) - part of TreeWillExpandListener
- **treeWillExpand** (TreeExpansionEvent) - part of TreeWillExpandListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

# Content Usage Examples Observable Attributes

- UI
- alignmentX
- alignmentY
- anchorSelectionPath
- background
- border
- cellEditor
- cellRenderer
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- editable
- enabled
- expandsSelectedPaths
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- invokesStopCellEditing
- largeModel
- layout
- leadSelectionPath
- maximumSize
- minimumSize
- model
- opaque
- preferredSize
- rootVisible
- rowHeight
- scrollsOnExpand
- selectionModel
- showsRootHandles
- toggleClickCount
- transferHandler
- verifyInputWhenFocusTarget
- visibleRowCount

## ***SwingBuilder.viewport***

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)

- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JViewport` is returned, unless the user passes in a subclass of `JViewport` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JViewport`

- **UI** <ViewportUI>
- **backingStoreEnabled** <boolean>
- **extentSize** <Dimension>
- **scrollMode** <int>
- **view** <Component>
- **viewPosition** <Point>
- **viewRect** <Rectangle>
- **viewSize** <Dimension>

## Declared in `javax.swing.JComponent`

- **autoscrolls** <boolean> Determines if this component automatically scrolls its contents when dragged.
- **border** <Border> The component's border.
- **componentPopupMenu** <JPopupMenu> Popup to show
- **debugGraphicsOptions** <int> Diagnostic options for graphics operations.
- **inheritsPopupMenu** <boolean> Whether or not the JPopupMenu is inherited
- **inputVerifier** <InputVerifier> The component's input verifier.
- **opaque** <boolean> The component's opacity
- **toolTipText** <String> The text to display in a tool tip.
- **transferHandler** <TransferHandler> Mechanism for transfer of data to and from the component
- **verifyInputWhenFocusTarget** <boolean> Whether the Component verifies input before accepting focus.
- **UIClassID** <String>
- **accessibleContext** <AccessibleContext>
- **actionMap** <ActionMap>
- **doubleBuffered** <boolean>
- **graphics** <Graphics>
- **height** <int>
- **managingFocus** <boolean>
- **nextFocusableComponent** <Component>
- **optimizedDrawingEnabled** <boolean>
- **paintingTile** <boolean>
- **registeredKeyStrokes** <KeyStroke[]>
- **requestFocusEnabled** <boolean>
- **rootPane** <JRootPane>
- **topLevelAncestor** <Container>
- **validateRoot** <boolean>
- **visibleRect** <Rectangle>
- **width** <int>
- **x** <int>
- **y** <int>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in javax.swing.JViewport

- **stateChanged** (ChangeEvent) - part of ChangeListener

## Declared in javax.swing.JComponent

- **vetoableChange** (PropertyChangeEvent) - part of VetoableChangeListener
- **ancestorMoved** (AncestorEvent) - part of AncestorListener
- **ancestorAdded** (AncestorEvent) - part of AncestorListener
- **ancestorRemoved** (AncestorEvent) - part of AncestorListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener

- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- UI
- UIClassID
- accessibleContext
- alignmentX
- alignmentY
- background
- backingStoreEnabled
- border
- changeListeners
- component
- componentCount
- componentPopupMenu
- components
- containerListeners
- enabled
- extentSize
- focusCycleRoot
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet
- focusable
- font
- foreground
- inheritsPopupMenu
- inputVerifier
- insets
- layout
- maximumSize
- minimumSize
- opaque
- preferredSize
- scrollMode
- transferHandler
- verifyInputWhenFocusTarget
- view
- viewPosition
- viewRect
- viewSize

## *SwingBuilder.widget*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A java.awt.Component is returned, unless the user passes in a subclass of Component as the value argument, in which case the value argument is returned.

## Attributes Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseWheelEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples

An example with I2fprod's property sheet:

```
dialog(title: "Configuration", id:
"aboutConfigDialog", modal: true ) {
    def pst = new PropertySheetTable()
    bind(source:model, sourceProperty:
'propertyTableModel', target: pst,
targetProperty:'model' )
    widget(new PropertySheetPanel(pst))
}
```

## Observable Attributes

- background
- focusable
- font
- foreground

### *SwingBuilder.window*

- [Generated Object](#)
- [Attributes](#)
- [Events](#)
- [Content](#)
- [Usage](#)
- [Examples](#)
- [Observable Attributes](#)

(Automatically generated from BeanInfo)

## Generated Object

A `javax.swing.JWindow` is returned, unless the user passes in a subclass of `JWindow` as the value argument, in which case the value argument is returned.

## Attributes Declared in `javax.swing.JWindow`

- **contentPane** <Container> The client area of the window where child components are normally inserted.
- **glassPane** <Component> A transparent pane used for menu rendering.
- **layeredPane** <JLayeredPane> The pane which holds the various window layers.
- **rootPane** <JRootPane> the `RootPane` object for this window.

## Declared in `java.awt.Window`

- **accessibleContext** <AccessibleContext>
- **active** <boolean>
- **alwaysOnTop** <boolean>

- **bufferStrategy** <BufferStrategy>
- **cursor** <Cursor>
- **focusCycleRootAncestor** <Container>
- **focusOwner** <Component>
- **focusableWindow** <boolean>
- **focusableWindowState** <boolean>
- **focused** <boolean>
- **graphicsConfiguration** <GraphicsConfiguration>
- **inputContext** <InputContext>
- **locale** <Locale>
- **locationByPlatform** <boolean>
- **locationRelativeTo** <Component>
- **mostRecentFocusOwner** <Component>
- **ownedWindows** <Window[]>
- **owner** <Window>
- **showing** <boolean>
- **toolkit** <Toolkit>
- **warningString** <String>

## Declared in java.awt.Container

- **alignmentX** <float>
- **alignmentY** <float>
- **componentCount** <int>
- **components** <Component[]>
- **focusCycleRoot** <boolean>
- **focusTraversalPolicy** <FocusTraversalPolicy>
- **focusTraversalPolicyProvider** <boolean>
- **focusTraversalPolicySet** <boolean>
- **insets** <Insets>
- **layout** <LayoutManager>
- **maximumSize** <Dimension>
- **minimumSize** <Dimension>
- **preferredSize** <Dimension>

## Declared in java.awt.Component

- **background** <Color>
- **enabled** <boolean>
- **focusable** <boolean>
- **font** <Font>
- **foreground** <Color>
- **name** <String>
- **visible** <boolean>

## Events Declared in java.awt.Window

- **windowStateChanged** (WindowEvent) - part of WindowStateListener
- **windowGainedFocus** (WindowEvent) - part of WindowFocusListener
- **windowLostFocus** (WindowEvent) - part of WindowFocusListener
- **windowActivated** (WindowEvent) - part of WindowListener
- **windowClosed** (WindowEvent) - part of WindowListener
- **windowClosing** (WindowEvent) - part of WindowListener
- **windowDeactivated** (WindowEvent) - part of WindowListener
- **windowDeiconified** (WindowEvent) - part of WindowListener

- **windowIconified** (WindowEvent) - part of WindowListener
- **windowOpened** (WindowEvent) - part of WindowListener

## Declared in java.awt.Container

- **componentAdded** (ContainerEvent) - part of ContainerListener
- **componentRemoved** (ContainerEvent) - part of ContainerListener

## Declared in java.awt.Component

- **keyPressed** (KeyEvent) - part of KeyListener
- **keyReleased** (KeyEvent) - part of KeyListener
- **keyTyped** (KeyEvent) - part of KeyListener
- **componentHidden** (ComponentEvent) - part of ComponentListener
- **componentMoved** (ComponentEvent) - part of ComponentListener
- **componentResized** (ComponentEvent) - part of ComponentListener
- **componentShown** (ComponentEvent) - part of ComponentListener
- **hierarchyChanged** (HierarchyEvent) - part of HierarchyListener
- **mouseClicked** (MouseEvent) - part of MouseListener
- **mouseEntered** (MouseEvent) - part of MouseListener
- **mouseExited** (MouseEvent) - part of MouseListener
- **mousePressed** (MouseEvent) - part of MouseListener
- **mouseReleased** (MouseEvent) - part of MouseListener
- **mouseDragged** (MouseEvent) - part of MouseMotionListener
- **mouseMoved** (MouseEvent) - part of MouseMotionListener
- **propertyChange** (PropertyChangeEvent) - part of PropertyChangeListener
- **ancestorMoved** (HierarchyEvent) - part of HierarchyBoundsListener
- **ancestorResized** (HierarchyEvent) - part of HierarchyBoundsListener
- **focusGained** (FocusEvent) - part of FocusListener
- **focusLost** (FocusEvent) - part of FocusListener
- **mouseWheelMoved** (MouseEvent) - part of MouseWheelListener
- **caretPositionChanged** (InputMethodEvent) - part of InputMethodListener
- **inputMethodTextChanged** (InputMethodEvent) - part of InputMethodListener

## Content Usage Examples Observable Attributes

- accessibleContext
- active
- alignmentX
- alignmentY
- alwaysOnTop
- background
- bufferStrategy
- component
- componentCount
- components
- containerListeners
- cursor
- focusCycleRoot
- focusCycleRootAncestor
- focusOwner
- focusTraversalKeys
- focusTraversalPolicy
- focusTraversalPolicyProvider
- focusTraversalPolicySet

- focusable
- focusableWindow
- focusableWindowState
- focused
- font
- foreground
- graphicsConfiguration
- inputContext
- insets
- layout
- locale
- locationByPlatform
- locationRelativeTo
- maximumSize
- minimumSize
- mostRecentFocusOwner
- ownedWindows
- owner
- preferredSize
- showing
- toolkit
- warningString
- windowFocusListeners
- windowListeners
- windowStateListeners

## **Categorical Widget List**

### ***Root Windows***

- dialog
- frame
- window

### ***Embeddable Windows***

- optionPane
- fileChooser
- colorChooser

### ***Containers***

- box
- desktopPane
  - internalFrame
- layeredPane
- panel
- scrollPane
  - viewport
- splitPane
- tabbedPane
- toolBar
- container - returns value argument or container attribute

### ***Menus***

- menuBar
- popupMenu

- menu
- menuItem
- checkBoxMenuItem
- radioButtonMenuItem

#### **Widgets**

- button
- checkBox
- comboBox
- editorPane
- formattedTextField
- label
- list
- passwordField
- progressBar
- radioButton
- scrollbar
- separator
- slider
- spinner
- table
- textArea
- textPane
- textField
- toggleButton
- tree
- widget - returns value argument or widget attribute

#### **Borders**

- compoundBorder
- emptyBorder
- etchedBorder
- lineBorder
- loweredBevelBorder
- loweredEtchedBorder - alias for etchedBorder
- matteBorder
- raisedBevelBorder
- raisedEtchedBorder
- titledBorder

#### **Layouts**

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
  - GridBagConstraints
  - gbc - alias for GridBagConstraints
- GridLayout
- OverlayLayout
- SpringLayout
- box
  - hbox

- hglue
- hstrut
- vbox
- vglue
- vstrut
- glue
- rigidArea

#### ***Models***

- boundedRangeModel
- buttonGroup
  
- spinnerDateModel
- spinnerListModel
- spinnerNumberModel
  
- tableModel
  - tableColumn
  - propertyColumn
  - closureColumn

#### ***Other***

- action
- actions
- imageIcon
- map

### **Extending Swing Builder**

While SwingBuilder comes with almost all of the standard Swing widgets, sometimes you need to add custom or extended components. There are several ways to accomplish this.

#### ***Use a Pass-Through Element***

SwingBuilder has two 'magic' elements that pass through the value argument or the named attribute to the parent container. The `widget` element has been in Groovy since 1.0. New in Groovy 1.1Beta2 is the `container` element. Currently they are identical in behavior and should be used mostly to increase code readability: elements that can contain children should use `container` while `widget` elements should contain no children. (in a future build this restriction may be enforced at the builder level).

To use the `widget` or `container` elements you pass in the desired custom widget or custom container object as the value argument or the `widget:` or `container:` attributes respectively.

```

swing.frame() {
    container(new MyCustomDocingPanel(),
constraints:BorderLayout.CENTER) {
        widget(new CustomMappingComponent())
    }
    container(container:new
CustomOutlookBar(),
constraints:BorderLayout.WEST) {
        widget(widget:new MyCustomButton())
    }
}

```

*Value Argument Replacement (new in 1.1Beta2)*

Most widgets accept a widget of the appropriate type in the value argument of the swing builder element. If your extended component extends a common swing widget type you can supply an instance of that component in the value argument.

```

swing.frame() {
    button(new MyCustomButton("custom
constructor"), enabled:false)
}

```

*Register a Bean Factory (new in 1.1Beta2)*

If your custom component follows the JavaBeans specification, specifically having a no-argument public constructor, then you can register your component directly with the SwingBuilder and use it as a first class element. To do this call the `registerBeanFactory(String, Class)` method with the desired element name and Bean class.

```
swing.registerBeanFactory('magicButton',
MagicButton.class)
swing.frame() {
    magicButton("It's Magical!")
}
```

#### *Register a Custom Factory (new in 1.1Beta2)*

Sometimes you will need greater interaction with the supplied values in the SwingBuilder, or you want to accomplish richer things with the supplied arguments. In this case you can register a custom factory. Custom factories implement `groovy.swing.factory.Factory` and are registered via the `registerFactory(String, Factory)` method.

All of the elements that do interesting things with their argument values and that need to access different constructors or lack no-argument constructors utilize this interface. Tricks such as accessing the Action constructors in the button type widgets, using the `javafx.swing.Box` factory methods are two such real world examples.

```
// a bit of a goofy example, we forcefully
make the button blue
swing.registerFactory('blueButton',
    [newInstance:{builder, name, value,
attrs ->
    attrs.remove('color')
    builder.button(value,
color:Color.BLUE)}
    ] as groovy.swing.factory.Factory)

swing.frame() {
    blueButton('I want to be red!',
color:Color.RED)
}
```

## Multithreading with SwingBuilder

An example of using threading with SwingBuilder to do lengthy operations in the background so GUI event handling remains responsive. This sample code demonstrates many of the issues that may occur including replacing the content of a "please wait" message with something else, and safely changing "realized" components only on the EDT (Event Dispatch Thread). This sample happens to use tableLayout, which can get pretty slow but that is a good stresser for the demonstration.

Eventually there will be more explanation of what's going on here, but hopefully this will be helpful.

```
// @author Jim White
<mailto:jim@pagesmiths.com>

import groovy.swing.SwingBuilder

import java.awt.BorderLayout
import javax.swing.JFrame
import java.awt.GridLayout
import java.awt.Color

SwingBuilder.build {
    myFrame = frame(title: 'SwingThreading',
pack: true, defaultCloseOperation:
JFrame.EXIT_ON_CLOSE) {
        BorderLayout()
        def lazyPanelsParent
        scrollPane(constraints:
BorderLayout.CENTER, preferredSize: [500,
300]) {
            lazyPanelsParent = panel(layout:
new GridLayout(0, 1, 5, 5)) {
                }
            }
        panel(constraints:
```

```

BorderLayout.NORTH) {
    button(text: 'Build One',
actionPerformed: {
    println 'Pressed'

buildInTheBackground(lazyPanelsParent)
    })
    }
}
myFrame.show()
}

/**
 * Create a Swing panel in the background
using SwingBuilder.
 * This can be called from any thread.
 */
def buildInTheBackground(parentPanel) {
    // If we are not already on the EDT,
static SwingBuilder.build(Closure) will do
that for us.
    // In the case of an event handler like
the actionPerformed for the button, then
naturally
    // we're on the EDT already and the
building will continue immediately.
    SwingBuilder.build {
        def statusMessage = label(border:
lineBorder(color:Color.RED, thickness:4))

        // Notice that the parameter to

```

```
setStatus() is declared as String.  
    // That way if code uses a GString  
it gets rendered at the time (and on the  
    // thread) of the call. If we don't  
do that then it might get deferred,  
    // by which time the results that  
the GString will produce may change and  
    // could be in a race (results  
varying with execution order and maybe  
invalid).  
    // Also the context here is that the  
caller is the work environment and that  
    // operations on the EDT are to be  
quick and determinate for display only.  
  
    def setStatus = { String msg ->  
println msg; statusMessage.setText(msg) }  
  
    // We can change the shown Swing  
tree here because we're on the EDT.  
  
    setStatus('Building...')  
  
    // Put the message at the top of the  
list if we use index = 0.  
    // Use plain  
parentPanel.add(statusMessage) if you want  
the end of the list.  
    parentPanel.add(statusMessage, 0)  
  
    // Component.validate() must be
```

called after any changes that affect layout  
// on realized (shown) AWT/Swing  
trees.

```
parentPanel.validate()
```

```
doOutside {
```

```
    // Now we're off on our own  
thread.
```

```
    // To change the display, be  
sure to run that code on the EDT.
```

```
    edt { setStatus('Working  
outside') }
```

```
    // That will wait, which isn't  
really necessary here.
```

```
    // For more parallelism, use  
doLater.
```

```
    // doLater { setStatus('Working  
outside') }
```

```
    // Here we do our lengthy work.  
sleep(3000)
```

```
    def powerData = [* 0..5].collect  
{power -> [* 0..10].collect { Math.pow(it,  
power) } }
```

```
    // SwingBuilding may take a  
while too.
```

```
    def newPanel =  
panel(minimumSize: [250, 250], border:
```

```

lineBorder(color:Color.BLUE, thickness:2)) {
    edt { setStatus("Building
new panel") }

        sleep(3000)
        tableLayout {
            powerData.each {row ->
tr { row.each {cell -> td {
textField(cell.toString()) } } } }
        }
    }

    edt { setStatus("New panel
ready") }

        // Back to the EDT for fiddling
with the Swing tree.

        doLater {
            setStatus("Adding new
panel")

                // We're gonna remove the
widget used for the new panel's progress
indication.

                    // An alternate (and
simpler) way to get this effect is to put
the status

                        // in a panel that doesn't
get removed. But this method is shown
because

```

```

        // it is fairly general in
that the status indicator can have a utility
        // layout that gets removed
when done and so doesn't impact the
customized
        // structure desired for the
application.

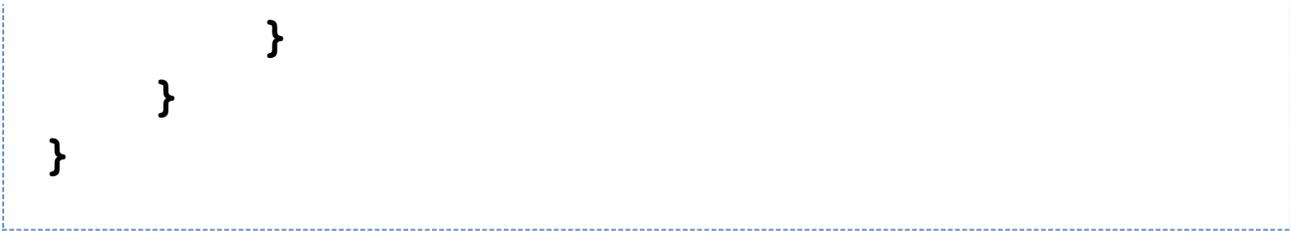
        // We may not be the zeroth
child (beginning of list) anymore.
        // (Or end of list either,
if that's where we started.)
        def idx =
(parentPanel.components as
List).indexOf(statusMessage)
        parentPanel.remove(idx)
        parentPanel.add(newPanel,
idx)

        parentPanel.validate()

// Cute groovyism, but we don't need the
extra overhead.
//
        parentPanel.with {
//
        // We may not be the
zeroth child anymore.
//
        def idx = (components
as List).indexOf(statusMessage)
//
        remove(idx)
//
        add(newPanel, idx)
//
        validate()
//
        }

```

```
//          myFrame.pack()  
          setStatus("Built")  
        }
```



[SwingThreading.groovy](#)

## SwingXBuilder

**What is SwingX(SwingLabs)?**

SwingX is the UI component library developed by Sun's SwingLabs. Many of the components used and developed in the "Filthy Rich Clients" and "Extreme GUI Makeover" presentations first make their mark in SwingLabs before making it to core Java. SwingX adds a lot of convenience features that originally one would have to make himself, such as TaskPanels, enhanced core components, glossy components, etc.

**Can I use it with my application built using v1.0?**

Yes... The SwingXBuilder subclasses the classic SwingBuilder so generally speaking you can just switch the declaration and everything SHOULD work. Please let me know if something from core is broken in SwingXBuilder. Any component that exists in both defaults to the SwingX version but can be accessed by adding the classicSwing property and setting its value to true.

**How do I build it?**

Checkout the package from SVN [here](#) .

**Download**

0.1.1 is compiled against the [SwingLabs 16 Sep 2007 build](#) a Make sure those jars are on your path, including the ones in the lib directory.

0.1.2 and 0.1.3 are using [SwingX 0.9 release build](#).

0.1.5 uses the [SwingX 0.9.1 release build](#), and ships with the Windows installer for Groovy 1.5.0.

[swingxbuilder-0.1.1.jar](#) (compatibile with Groovy 1.1-beta-3)

[swingxbuilder-0.1.2.jar](#) (compatibile with Groovy 1.1-rc-1)

[swingxbuilder-0.1.3.jar](#) (ships with Groovy 1.1-rc-2 windows installer)

[swingxbuilder-0.1.5.jar](#) (ships with Groovy 1.5 windows installer)

## Effects

Effects are objects that can be added to Painters to modify the way they are drawn.

SwingX includes several built-in Effects for your use:

- GlowPathEffect - Draws a glow around the Painter.
- InnerGlowPathEffect - Draws a glow inside the Painter's shape.
- InnerShadowPathEffect - Draws a shadow inside the Painter's shape.
- NeonBorderEffect - Draws a multicolored border around the Painter interpolating between two colors.
- ShadowPathEffect - Draws a shadow around the Painter's shape.

With the exception of NeonBorderEffect, the other effects have no argument constructors.

<i>Java class name:NeonBorderEffect</i>	<i>SwingXBuilder node: neonBorderEffect</i>
<b>Attribute</b>	Expected Type
<b>edgeColor</b>	Color
<b>centerColor</b>	Color
<b>effectWidth</b>	Integer

## Extending SwingXBuilder

Since the SwingXBuilder subclasses the SwingBuilder, the same operations apply to the SwingXBuilder.

### New Features in SwingXBuilder

#### *Shortcutting/Aliasing attribute names*

This has actually been around for a while but I only spoke about it to my mentors and not you guys. Sometimes attribute names are a bit long to type. In this example pulled from one of the unit tests, we can see how to add an alias for any property on a component. During builder parsing, the names are changed to their explicit forms.

```
builder.addShortcut(CheckerboardPainter.class, "squareSize", "size")
builder.addShortcut(CheckerboardPainter.class, "lightPaint", "light")
builder.addShortcut(CheckerboardPainter.class, "darkPaint", "dark")
def cp =
builder.checkerboardPainter(size:40.0d,
light:Color.WHITE, dark:Color.RED)
```

Before you ask, I did play around a bit with the idea of aliasing the component names but that was a bit cumbersome.

NOTE: Be careful when aliasing properties, making sure the alias doesn't already exist as a property name. For example, foreground and foregroundPainter exist for some components.

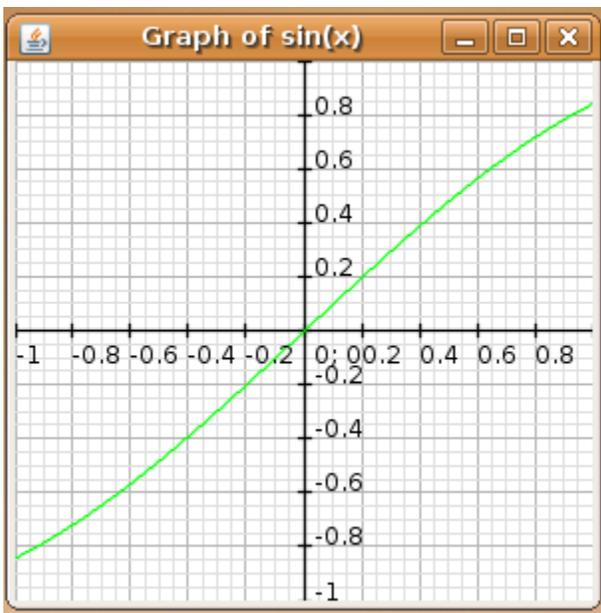
## Graphs

JXGraph allows you to plot functions on a graph. JXGraph in Java takes a Plot (usually an anonymous function) to plot a function. Because anonymous classes are not possible in Groovy, a new class, GroovyPlot, has been created that can take a closure and generate a Plot.

The graph node in SwingXBuilder adds a property *plots* that takes a multi-dimensional List as a parameter. The internal list has a Color component and a closure representing the plot. Here is an example of code plotting  $\sin x$  :

```
def swing = new SwingXBuilder()
def frame = swing.frame(size:[300,300]) {
    graph(plots:[ [Color.GREEN, {value ->
Math.sin(value)}]])
}.show()
```

And here is the corresponding graph:



To graph, for instance, the cosine as well as the sine, the graph node signature would be:

```
graph(plots:[ [Color.GREEN, {value ->
Math.sin(value)}], [Color.BLUE, { value ->
Math.cos(value)}] ])
```

### MultiSplitPane

JXMultiSplitPane is somewhat of a hybrid component using MultiSplitLayout which is somewhere between GridLayout and GridBagLayout. MultiSplitLayout divides its components into horizontal or vertical Splits which are composed of Leaf components(which accept components) and Dividers(which separate them).

```

import groovy.swing.SwingXBuilder
def swing = new SwingXBuilder()
def frame = swing.frame(size:[300,300],
title:'JXMultiSplitPane Demo') {
    multiSplitPane() {
        split() {
            leaf(name:"left")
            divider()
            leaf(name:"right")
        }
        button(text:"Left Button",
constraints:"left")
        button(text:"Right Button",
constraints:"right")
    }
}.show()

```

Vertical splits can be created by setting the Split property *rowLayout* to false.

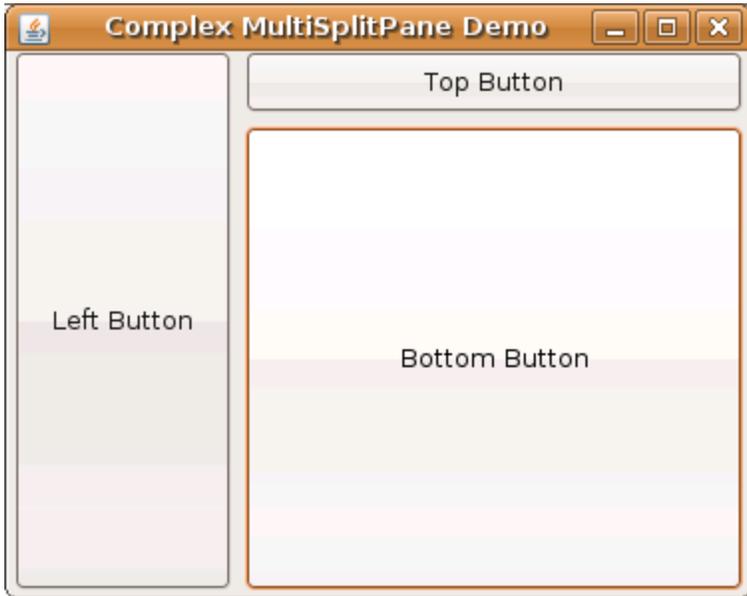
The code above produces the below layout:



Here is the code for a slightly more complex layout:

```
import groovy.swing.SwingXBuilder
def swing = new SwingXBuilder()
def frame = swing.frame(size:[300,300]) {
    multiSplitPane() {
        split() {
            leaf(name:"left")
            divider()
            split(rowLayout:false) {
                leaf(name:"top")
                divider()
                leaf(name:"bottom")
            }
        }
        button(text:"Left Button",
constraints:"left")
        button(text:"Right Button",
constraints:"right")
        button(text:"Top Button",
constraints:"top")
        button(text:"Bottom Button",
constraints:"bottom")
    }
}.show()
```

That code produces the following layout:



## Painters

Painters are a sort of user interface delegate for drawing foregrounds and backgrounds. SwingX includes some built-in painters:

<i>Java class name:AlphaPainter</i>	<i>SwingXBuilder node: alphaPainter</i>
Attribute	Expected Type
<b>alpha</b>	float
<b>painters</b>	ArrayList<Painter> or Painter[]
<i>Java class name:BusyPainter</i>	<i>SwingXBuilder node: busyPainter</i>
Attribute	Expected Type
<b>barLength</b>	float
<b>barWidth</b>	float
<b>baseColor</b>	Color
<i>Java class name:CapsulePainter</i>	<i>SwingXBuilder node: capsulePainter</i>
Attribute	Expected Type
<b>portion</b>	CapsulePainter.Portion
<i>Java class name:CheckerboardPainter</i>	<i>SwingXBuilder node: checkerboardPainter</i>

Attribute	Expected Type
<b>squareSize</b>	float
<b>lightPaint</b>	Color
<b>darkPaint</b>	Color
<i>Java class name:ImagePainter</i>	<i>SwingXBuilder node: imagePainter</i>
Attribute	Expected Type
<b>image</b>	BufferedImage
<b>url</b>	URL
<b>style</b>	AbstractAreaPainter.Style
<b>horizontal</b>	AbstractLayoutPainter.HorizontalAlignment
<b>vertical</b>	AbstractLayoutPainter.VerticalAlignment
<i>Java class name:ImagePainter</i>	<i>SwingXBuilder node: imagePainter</i>
Attribute	Expected Type
<b>width</b>	integer
<b>height</b>	integer
<b>rounded</b>	boolean
<b>roundHeight</b>	integer
<b>roundWidth</b>	integer
<b>fillPaint</b>	Color
<i>Java class name:PinstripePainter</i>	<i>SwingXBuilder node: pinstripePainter</i>
Attribute	Expected Type
<b>paint</b>	Paint
<b>angle</b>	double
<b>stripeWidth</b>	double

<b>spacing</b>	double
<i>Java class name:MattePainter</i>	<i>SwingXBuilder node: mattePainter</i>
Attribute	Expected Type
<b>paint</b>	Paint
<b>paintStretched</b>	boolean
<i>Java class name:TextPainter</i>	<i>SwingXBuilder node: textPainter</i>
Attribute	Expected Type
<b>font</b>	Font
<b>text</b>	String
<b>paint</b>	Paint
<i>Java class name:GlossPainter</i>	<i>SwingXBuilder node: glossPainter</i>
Attribute	Expected Type
<b>position</b>	GlossPainter.GlossPosition
<b>paint</b>	Paint

***CompoundPainter***

CompoundPainter, as its name indicates, is a collection of other Painters. Order is important. CompoundPainters build from back to front. Here is an example of a CompoundPainter:

```

comp =
swing.compoundPainter(clipPreserved:true) {

rectanglePainter(borderWidth:3f,
borderPaint:Color.BLACK,
                  areaEffects:new
GlowPathEffect(), roundHeight:40,
roundWidth:40,
                  insets: [3,3,3,3],
paintStretched:true,style:AbstractAreaPainte
r.Style.FILLED,
                  fillPaint:grad)

pinstripePainter(paint:[204,0,0,36] as
Color,angle:45, stripeWidth:8, spacing:8)

}

```

AlphaPainter is a CompoundPainter as well, adding an alpha value to a collection of Painters. CompoundPainter can take the painters attribute or you can enumerate the painters inside the closure.

#### Widgets and Common Attributes

Windows and Dialogs	
<i>Java class name:JXFrame</i>	<i>SwingXBuilder node: frame</i>
Attribute	Expected Type
<b>title</b>	String
<b>layout</b>	LayoutManager
<b>size</b>	Dimension or [x,y]
<b>defaultCloseOperation</b>	WindowConstants

<i>Java class name:JXPanel</i>	<i>SwingXBuilder node: panel</i>
Attribute	Expected Type
<b>layout</b>	LayoutManager
<b>size</b>	Dimension or [x,y]
<b>alpha</b>	float
<i>Java class name:JXTitledPanel</i>	<i>SwingXBuilder node: titledPanel</i>
Attribute	Expected Type
<b>layout</b>	LayoutManager
<b>titleForeground</b>	Color
<b>titlePainter</b>	Painter
<b>title</b>	String
<b>size</b>	Dimension or [x,y]
<i>Java class name:JXLoginDialog</i>	<i>SwingXBuilder node: loginDialog</i>
Attribute	Expected Type
<b>title</b>	String
<b>modal</b>	boolean
<b>service</b>	LoginService
<i>Java class name:JXCollapsiblePane</i>	<i>SwingXBuilder node: collapsiblePane</i>
Attribute	Expected Type
<b>orientation</b>	JXCollapsiblePane.Orientation
<b>layout</b>	LayoutManager
<b>animated</b>	boolean
<b>collapsed</b>	boolean

Basic Components

<i>Java class name: JXBusyLabel</i>	<i>SwingXBuilder node: busyLabel</i>
Attribute	Expected Type
<b>busy</b>	boolean
<i>Java class name: JXButton</i>	<i>SwingXBuilder node: button</i>
Attribute	Expected Type
<b>text</b>	String
<b>icon</b>	Icon
<b>backgroundPainter</b>	Painter
<b>foregroundPainter</b>	Painter
<i>Java class name: JXLabel</i>	<i>SwingXBuilder node: label</i>
Attribute	Expected Type
<b>text</b>	String
<b>icon</b>	Icon
<b>horizontalAlignment</b>	Painter
<b>foregroundPainter</b>	Painter
<b>backgroundPainter</b>	Painter
<b>textRotation</b>	double
<b>lineWrap</b>	boolean

## IDE Support

Groovy is supported by the following IDEs and related tools:

- [Eclipse Plugin](#)
  - [About GroovyEclipse Plugin v1](#)
  - [Groovy-Eclipse 2.0.0RC1 New and Noteworthy](#)
  - [Groovy Eclipse 1.5.7 Release Notes](#)
  - [Eclipse Plugin FAQ](#)
  - [Groovy-Eclipse 2.0.0M1 New and Noteworthy](#)
  - [Groovy-Eclipse 2.0.0M2 New and Noteworthy](#)
  - [Groovy-Eclipse 2.0.0 New and Noteworthy](#)
  - [Eclipse Plugin Refactoring](#)

- [Create Your First Groovy Project](#)
- [Install Groovy-Eclipse Plugin](#)
- [Groovy-Eclipse 2.0.1 New and Noteworthy](#)
- [Groovy-Eclipse coding conventions](#)
  - [\(Deprecated\) Getting Started With Groovy-Eclipse Source Code on Ubuntu](#)
  - [Getting Started With Groovy-Eclipse Source Code](#)
- [Extending Groovy Eclipse](#)
- [Groovy-Eclipse 2.0.2 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.0RC2 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.1RC1 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.2 New and Noteworthy](#)
- [DSL Descriptors for Groovy-Eclipse](#)
  - [DSL examples](#)
  - [Support for testing DSLs](#)
- [A unified approach to DSL support in IDEs](#)
- [Groovy-Eclipse 2.5.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.5.1 New and Noteworthy](#)
- [Compiler Switching within Groovy-Eclipse](#)
- [Groovy-Eclipse compiler plugin for Maven](#)
- [Groovy-Eclipse 2.1.0RC1 New and Noteworthy](#)
- [Groovy-Eclipse 2.5.2 New and Noteworthy](#)
- [Groovy-Eclipse 2.6.0 New and Noteworthy](#)
- [UCOSP](#)
- [Groovy-Eclipse 2.6.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.7.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.7.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.7.2 New and Noteworthy](#)
- [DevelopmentBuilds](#)
- [IntelliJ IDEA Plugin by JetBrains](#)
  - [Wish List \(JetBrains Edition\)](#)
- [Old and deprecated pages](#)
  - [JEdit Plugin](#)
  - [Oracle JDeveloper Plugin](#)
- [Other Plugins](#)
  - [Dash](#)
  - [Emacs Groovy Mode](#)
  - [Emacs Mode](#)
  - [Emacs Plugin](#)
  - [NetBeans Plugin](#)
  - [TextMate](#)
  - [UltraEdit Plugin](#)

## Eclipse Plugin

**Groovy-Eclipse** is the set of Eclipse plugins that provide Eclipse support for Groovy projects.

## Installation

The latest Groovy-Eclipse release is available from the following Eclipse update sites. To install, point your Eclipse update manager to the update site appropriate for your Eclipse version or drag and drop the marketplace icon into your Eclipse installation.

Eclipse level	Release update site	
4.3 (Kepler)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/">http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/</a>  Kepler-based builds are still snapshots. We are providing early access for those people who want to try it out.	
4.2 and 3.8 (Juno)	<a href="http://dist.springsource.org/release/GRECLIPSE/e4.2/">http://dist.springsource.org/release/GRECLIPSE/e4.2/</a>	
3.7 (Indigo)	<a href="http://dist.springsource.org/release/GRECLIPSE/e3.7/">http://dist.springsource.org/release/GRECLIPSE/e3.7/</a>	
3.6 (Helios)	<a href="http://dist.springsource.org/release/GRECLIPSE/e3.6/">http://dist.springsource.org/release/GRECLIPSE/e3.6/</a>	
3.5 (Galileo)	<a href="http://dist.springsource.org/release/GRECLIPSE/e3.5/">http://dist.springsource.org/release/GRECLIPSE/e3.5/</a>	

Or, you can choose to install a development build, see below for more update sites. In general, our development builds are quite stable. A development build is not published unless all unit tests pass.

-  The install directory must be writable by the current user. Groovy-Eclipse cannot be installed into a *shared install*. This includes protected directories on windows like C:\Program Files. For more information [Eclipse Bug 395516](#).

## News

### Eclipse Kepler support

Groovy-Eclipse now has snapshot builds targeting Eclipse 4.3 (Kepler). For now, we are using an old JDT and installing Groovy-Eclipse into Eclipse Kepler will downgrade your JDT to the Juno version. This will change as we put out more complete snapshots. Use the snapshot update site to install: <http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/> (February 7, 2013)

### Groovy 2.1 support

Groovy-Eclipse now supports Groovy 2.1. Install from the [development snapshot update site](#) in the *Extra compilers* section of the update site. Support for @DelegatesTo is included. (February 7, 2013)

## **i** Groovy-Eclipse 2.7.2 Released!

This service refresh of Groovy-Eclipse provides significant improvements with the `@CompileStatic` annotation. Also, we have upgraded to Groovy 2.0.6. Other fixes and features are included as well. See all the details: [Groovy-Eclipse 2.7.2 New and Noteworthy](#). (December 19, 2012)

- [Installation](#)
- [News](#)
- [Features and Status](#)
  - [Key Features](#)
  - [New and Noteworthy](#)
  - [Eclipse Version Compatibility](#)
  - [Groovy compiler Compatibility](#)
- [Getting Other Builds](#)
  - [Milestone Builds](#)
  - [Development Builds](#)
  - [Archived Development Builds](#)
- [Getting Started with Groovy-Eclipse v2](#)
  - [How to Install](#)
  - [How to Create a Groovy Project](#)
- [Information and Resources](#)
  - [FAQs](#)
  - [Tutorials](#)
  - [Mailing Lists](#)
- [Developer Resources](#)
- [Open Issues](#)

### **Features and Status**

Groovy-Eclipse allows you to edit, compile, run, and debug Groovy scripts and classes from the Eclipse SDK.



Groovy-Eclipse is the [top open source development tool](#) at [EclipseCon 2010](#).

### **Key Features**

- Syntax highlighting
- Type inferencing
- Compile and run Groovy classes and scripts in Eclipse
- Outline view for Groovy files

- Auto-completion
- [Refactoring](#)
- Source code formatting
- Basic debug support
- [Groovy 2.1, 2.0, 1.8 and 1.7 compilers](#)
- [DSL support using DSL descriptors](#)
- [Maven integraion](#)

Read about the goals, approach, and implementation of Groovy-Eclipse v2 [here](#).

### ***New and Noteworthy***

See what's new in the latest release at [Groovy-Eclipse 2.7.2 New and Noteworthy](#).

For information about prior releases, see here:

- [Groovy-Eclipse 2.7.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.7.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.6.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.6.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.5.2 New and Noteworthy](#)
- [Groovy-Eclipse 2.5.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.5.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.2 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.1RC1 New and Noteworthy](#)
- [Groovy-Eclipse 2.1.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.2 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.1 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.0 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.0RC1 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.0M2 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.0M1 New and Noteworthy](#)

### ***Eclipse Version Compatibility***

Groovy-Eclipse Plugin v2 supports Eclipse versions 4.2.x, 3.7.x, 3.6.x, 3.5.x, and 3.4.2. Groovy-Eclipse for Eclipse 3.6 and earlier are no longer supported, but the builds are still available.

### ***Groovy compiler Compatibility***

By default, Groovy-Eclipse comes with a 1.8 and 1.7 Groovy compiler. A Groovy 2.0 compiler can optionally be installed from the Groovy update site. See [Compiler Switching within Groovy-Eclipse](#) for more information.

### **Getting Other Builds**

Milestone and development builds are also available. A milestone build is a relatively stable build, released at the completion of a project milestone. Development builds are published more frequently, and contain more new features and fixes. Published development build have passed unit tests, but may be less stable than milestone builds.

### ***Milestone Builds***

We occasionally release milestone builds of upcoming releases. These are meant for users who want to have access to the most recent features and who are willing to give us feedback on them.

Groovy-Eclipse milestone and release candidate builds are available at:

Eclipse level	Milestone update site
4.2 / 3.8 (Juno)	<a href="http://dist.springsource.org/milestone/GRECLIPSE/e4.2/">http://dist.springsource.org/milestone/GRECLIPSE/e4.2/</a>
3.7 (Indigo)	<a href="http://dist.springsource.org/milestone/GRECLIPSE/e3.7/">http://dist.springsource.org/milestone/GRECLIPSE/e3.7/</a>
3.6 (Helios)	<a href="http://dist.springsource.org/milestone/GRECLIPSE/e3.6/">http://dist.springsource.org/milestone/GRECLIPSE/e3.6/</a>
3.5 (Galileo)	<a href="http://dist.springsource.org/milestone/GRECLIPSE/e3.5/">http://dist.springsource.org/milestone/GRECLIPSE/e3.5/</a>

Please note that these builds are temporary and are typically removed after the final release has been made public.

### ***Development Builds***

Use one of these update sites to install the latest and greatest Groovy-Eclipse. Development builds tend to be quite stable (they are not released unless all tests pass) and so if you are having any difficulty with your current install, then it is recommended to update to the latest dev build. To install, paste the appropriate update site below into your Eclipse update manager, select the artifacts to install and follow the instructions.

Eclipse level	Development update site
4.3 (Kepler)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/">http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/</a>
4.2 / 3.8 (Juno)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e4.2/">http://dist.springsource.org/snapshot/GRECLIPSE/e4.2/</a>
3.7 (Indigo)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e3.7/">http://dist.springsource.org/snapshot/GRECLIPSE/e3.7/</a>
3.6 (Helios)	<a href="http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.6/">http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.6/</a>
3.5 (Galileo)	<a href="http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.5/">http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.5/</a>

Although these builds may not be as stable as milestone or release builds, they pass our suite of unit and integration tests. These builds are typically usable, and if there is a problem it is often fixed quickly.

### ***Archived Development Builds***

If you want to use a local update site instead or you cannot access the update site because of a firewall then you can download an archive of the update site:

Eclipse level	Development update site zip
4.2 (Kepler)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/index.html">http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/index.html</a>
4.2 / 3.8 (Juno)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e4.2/index.html">http://dist.springsource.org/snapshot/GRECLIPSE/e4.2/index.html</a>

3.7 (Indigo)	<a href="http://dist.springsource.org/snapshot/GR_ECLIPSE/e3.7/index.html">http://dist.springsource.org/snapshot/GR_ECLIPSE/e3.7/index.html</a>
3.6 (Helios)	<a href="http://ci.repository.codehaus.org/greclipse/snapshot/e36/old">http://ci.repository.codehaus.org/greclipse/snapshot/e36/old</a>
3.5 (Galileo)	<a href="http://ci.repository.codehaus.org/greclipse/snapshot/e35/old">http://ci.repository.codehaus.org/greclipse/snapshot/e35/old</a>

There is also an ancient zip of Groovy-Eclipse V2.0.0 for Eclipse 3.5 available to [download](#).

These zip files are the compressed version of the update site. They are not plugins that you can just drop into your eclipse installation. Here's how to install an archive:

1. Download the zip file
2. Do *not* unzip or move into your dropins directory
3. Start eclipse
4. Help -> Install new software -> Add...
5. Select the location of the zip you just downloaded
6. Install as you would from the regular update site

## Getting Started with Groovy-Eclipse v2

This section has brief instructions for getting started with Groovy-Eclipse v2. The instructions assume familiarity with Eclipse. Users new to Eclipse, may wish to follow the links to tutorials, which illustrate and provide novice level instructions for installing and using the plugin.

### *How to Install*

Start with a compatible version of Eclipse (available from <http://eclipse.org/downloads>). Since Groovy-Eclipse relies on Eclipse's Java tooling (the JDT), almost any flavor of Eclipse that includes the JDT will be compatible. This includes the Java IDE, Classic, RCP, JavaEE, etc. Flavors such as Eclipse for PHP developers are not compatible because it does not include the JDT. If you are interested in Grails tooling, then you must install the [SpringSource Tool Suite](#).

Once you have a compatible Eclipse installed, then follow these directions to install Groovy-Eclipse:

- Go to: **Help -> Software Updates**.
- Change to the **Available Software** tab.
- Click on **Add Site**.
- Paste the update site URL appropriate for your version of Eclipse and click **OK**.
- You should see a "Groovy Update Site" entry in the list of update sites. Expand and select the **Groovy-Eclipse Plugin** feature. Optionally, you can choose to include the sources..
- Click **Install** and follow the prompts.
- Restart when asked
- Rejoice! You installed the Groovy-Eclipse Plugin v2.

For step-by-step installation instructions, see the [Install Groovy-Eclipse Plugin](#) tutorial.

### *How to Create a Groovy Project*

To create a basic Groovy project in Eclipse perform the following steps:

- Open the **Java perspective**

- Click on the new Groovy Project icon: 
- The new project wizard for Groovy uses most of the same options as the new project wizard for Java.
- Fill in the appropriate details and create the project.

You should have a **src** folder and several libraries. A **bin** folder is also created, but is hidden.

You can create a Groovy classes or test cases using wizards similar to the Eclipse's Java class and JUnit test case wizards. Use the options on the **File > New** menu or press CTRL-N.

For step-by-step instructions, see the [Create Your First Groovy Project](#) tutorial.

## Information and Resources

Use the resources listed below to learn about Groovy-Eclipse Plugin v2 and keep track of project status and milestones.

### FAQs

See the [Eclipse Plugin FAQ](#).

### Tutorials

The first two of a series of tutorials on using Groovy-Eclipse Plugins v2 are available.

- [Install Groovy-Eclipse Plugin](#)
- [Create Your First Groovy Project](#)
- [Installation and use of Grails tooling](#) (STS is required. Free download [here](#).)

### Mailing Lists

Join Groovy-Eclipse mailing lists to find answers, ask questions, and contribute to discussions about v2 of the plugin.

- For usage-related topics, join <http://xircles.codehaus.org/lists/eclipse-plugin-user@groovy.codehaus.org>.
- For dev-related topics, join <http://xircles.codehaus.org/lists/eclipse-plugin-dev@groovy.codehaus.org>.
- Report Bugs - Bugs can be viewed and filed at <http://jira.codehaus.org/browse/GRECLIPSE>.

## Developer Resources

For a detailed and reasonably up to date information on how to check out Groovy-Eclipse from Subversion, see [Getting Started With Groovy-Eclipse Source Code](#).

The summary:

- The CI Server for Groovy-Eclipse is located at the [Bamboo server](#). (Note that this is a new location as of August 3, 2012.)
- The Groovy-Eclipse Plugin v2 git repository is located at <https://github.com/groovy/groovy-eclipse>. If you are new to github and git, you can familiarize yourself by reading some good tutorials at [github](#).
- The Groovy-Eclipse Plugin v2 SVN repository is now deprecated. For historical reasons, you can still access it at <http://svn.codehaus.org/groovy/eclipse/trunk/>. If you are new to Subversion, see [Subversion home page](#).
- The plugin source itself is partially written in Groovy. And for that reason, you must have Groovy-Eclipse installed before you can compile Groovy-Eclipse.

- If you are interested in working Groovy source code or plan to do a local build, please contact the [eclipse plugin dev mailing list](#) and we can help you out.
- See a list of bugs suitable for community members to work on. All the issues labelled [help requested](#) are a good place to start.

## Open Issues

You can browse or enter new issues on the [GRECLIPSE](#) project on the jira server.

## About GroovyEclipse Plugin v1

**Groovy-Eclipse v1 is installable on Eclipse 3.3 and 3.4, but not 3.5. The Groovy-Eclipse team strongly recommends that users of v1 migrate to v2.**

To obtain the most recent release of Groovy-Eclipse Plugin v1, use the following update site:

- <http://dist.codehaus.org/groovy/distributions/update/>

To obtain recent development builds of Groovy-Eclipse Plugin v1, use the following update site:

- <http://dist.codehaus.org/groovy/distributions/updateDev/>

Releases on this site are not official. They pass all unit tests, but may contain experimental features.

## Groovy-Eclipse 2.0.0RC1 New and Noteworthy

Riding high on the back of the Groovy 1.7 release, the Groovy-Eclipse is pleased to announce its own 2.0.0RC1 release. Over [50 issues](#) have been addressed for this release and we consider this release high quality enough to be considered for release in early January, unless blocking bugs are found.

In this release, we have focused on polishing and refining existing features, but there are some new features introduced as well.

For information about previous Groovy-Eclipse releases, see [Groovy-Eclipse 2.0.0M1 New and Noteworthy](#) and [Groovy-Eclipse 2.0.0M2 New and Noteworthy](#).

### Groovy-Eclipse RC1 update site

To install RC1, add the following URL to your Eclipse update manager:

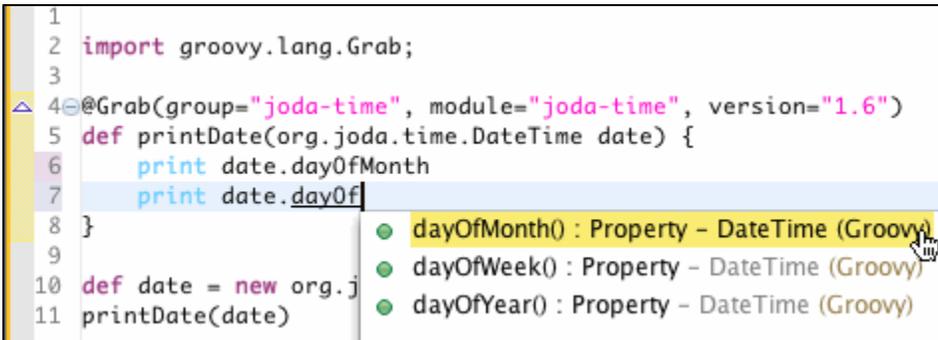
<http://dist.springsource.org/milestone/GRECLIPSE/e3.5/>

- [Better @Grab support](#)
- [Launching Groovy scripts](#)
- [Inferencing improvements](#)
- [Better AST transform support](#)
- [Jars from .groovy/lib directory added to classpath](#)
- [Groovy-aware file copy/rename](#)
- [Navigate to super class](#)
- [JavaDoc viewing](#)
- [Generated getter/setter navigation and JavaDocs](#)
- [Basic content assist options](#)
- [Issues fixed](#)
- [Thanks](#)

Better @Grab support

Until recently the use of @Grab to pull in source file dependencies was not recognized by groovy eclipse, resulting in a compilation error for the type that would have been located by following the Grab. Now they are being recognized. The @Grab referenced dependencies are added to the classpath used under the covers to build the project. Content assist is also possible for references to @Grab'd types. Grabbed dependencies are not currently reflected anywhere in the eclipse project metadata (ie. they are not seen in the classpath Eclipse manages for the project) - this may change in a future version of Groovy Eclipse.

When using @Grab, the Grab'd types are available through content assist and unknown references are underlined. In the following screenshot, you can see that the Joda-Time API can be verified and used within the editor even though its jars are only available through @Grab:



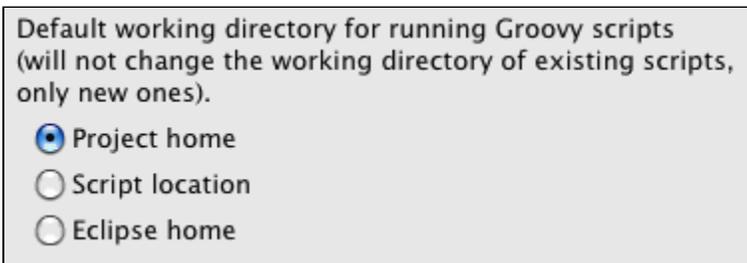
```
1
2 import groovy.lang.Grab;
3
4 @Grab(group="joda-time", module="joda-time", version="1.6")
5 def printDate(org.joda.time.DateTime date) {
6     print date.dayOfMonth
7     print date.dayOf
8 }
9
10 def date = new org.j
11 printDate(date)
```

The screenshot shows a code editor with Groovy code. Line 4 uses @Grab to fetch 'joda-time' dependencies. Line 5 defines a method printDate. Line 6 prints date.dayOfMonth. Line 7 prints date.dayOf, which is underlined. A content assist popup shows three options: dayOfMonth(), dayOfWeek(), and dayOfYear(), all with green circular icons and the text 'Property - DateTime (Groovy)'.

#### Launching Groovy scripts

Launching groovy scripts is now easier and more robust. We have addressed some key bugs in this area: [GRECLIP SE-555](#), [GRECLIPSE-560](#), [GRECLIPSE-478](#).

There is also an option to set the default working directory for the script launcher:



On the Groovy Preferences page, you can toggle between the project root, the workspace root, and the script location.

Additionally, the Run as Groovy Application launch configuration has been removed since it is a complete duplicate of the Java launch configuration. All compiled groovy scripts can be launched as a Java application through its generated main method.

#### Inferencing improvements

We are continuing to refine the [inferencing engine](#). Some of the notable improvements is support for navigation on the super, and this keywords. Also, content assist will show all the proper method and property completions when the class literal is accessed, e.g.:

```

class Bar {
    def field
}
class Foo extends Bar { }

```

Foo.class.c

- o canonicalName : String - Class (Groovy)
- o classes : Class[] - Class (Groovy)
- o classLoader : ClassLoader - Class (Groovy)
- △ classLoader0 : ClassLoader - Class (Groovy)
- o componentType : Class - Class (Groovy)
- △ constantPool : ConstantPool - Class (Groovy)
- o constructors : Constructor[] - Class (Groovy)
- o cast(Object obj) : Object - Class (Groovy)
- o checkInitted() : void - Class (Groovy)

And hovers now work correctly for references on super classes, methods, and fields:

```

class Bar {
    /**
     * Field from {@link Bar}
     */
    def field
}
class Foo extends Bar {
    /**
     * Field from {@link Foo}
     */
    def field

    def method() {
        super.field
    }
}

```

Object p.Bar.field  
Field from [Bar](#)

#### Better AST transform support

The AST transform support has been further refined. Eclipse does not allow individual source folders to be tagged as either 'source' or 'test' and then have them compiled separately. Due to this restriction it is hard to support the definition of an AST transform in the same project in which it is consumed. However, Groovy eclipse will allow this to occur **if** the transform is implemented in pure java (in a .java file) and that transform has no dependencies on uncompiled groovy code (ie. code in .groovy source files in the same project).

Also, groovy eclipse now allows AST transforms to be defined in projects upon which you depend. Previously it only supported consuming them from .jar files on the project classpath.

#### Jars from .groovy/lib directory added to classpath

All jars in your <user.home>/groovy/lib directory are added to your classpath as part of the Groovy support classpath container. After a change to the groovy lib directory, you can refresh the workspace's Groovy support classpath container by going to the Groovy Preferences page:

Reset the Groovy Classpath Containers.

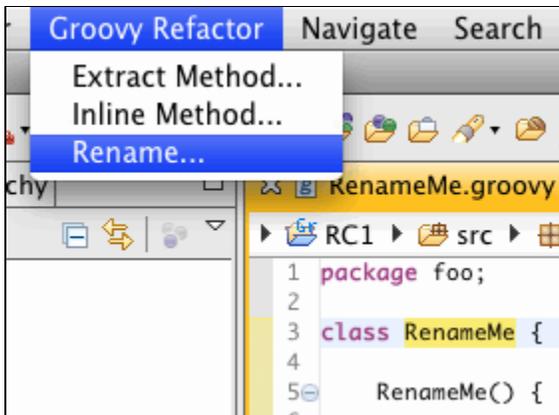
Update all Groovy Classpath Containers

Perform this action if there are changes to ~/.groovy/lib that should be reflected in your projects' classpaths.

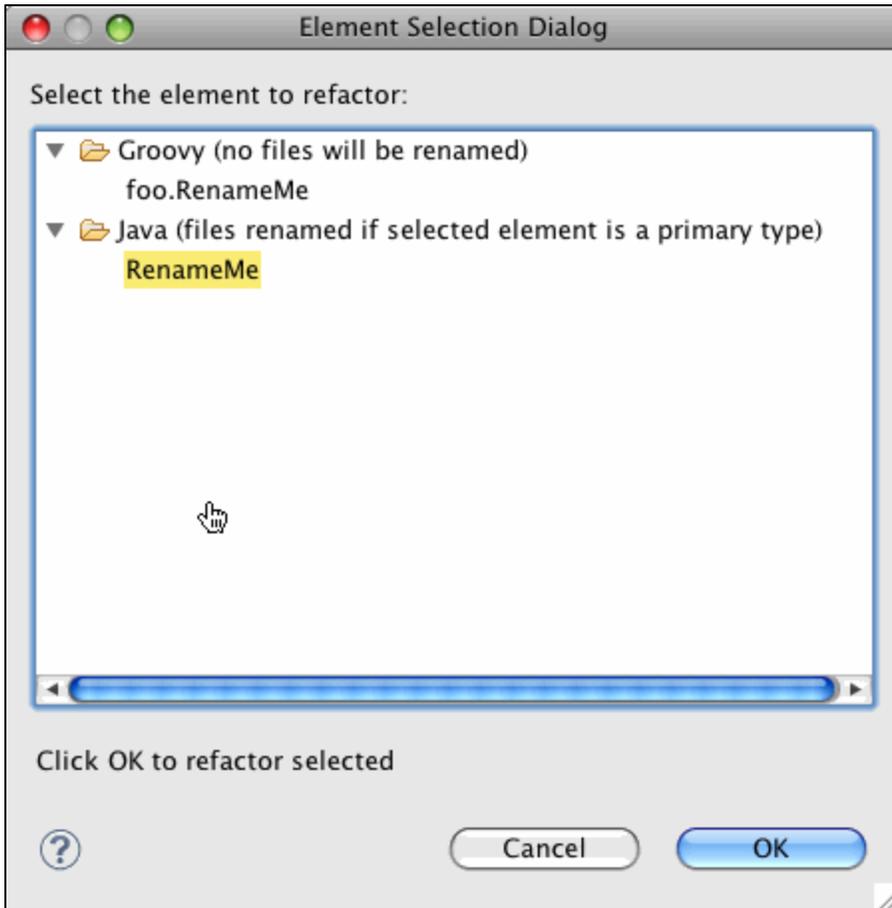
#### Groovy-aware file copy/rename

Groovy-aware file copy, rename, and paste are working. In the package explorer, you can select a groovy compilation unit and type CTRL-c and CTRL-v (or CMD-c and CMD-v on Mac). The compilation unit will be appropriately copied.

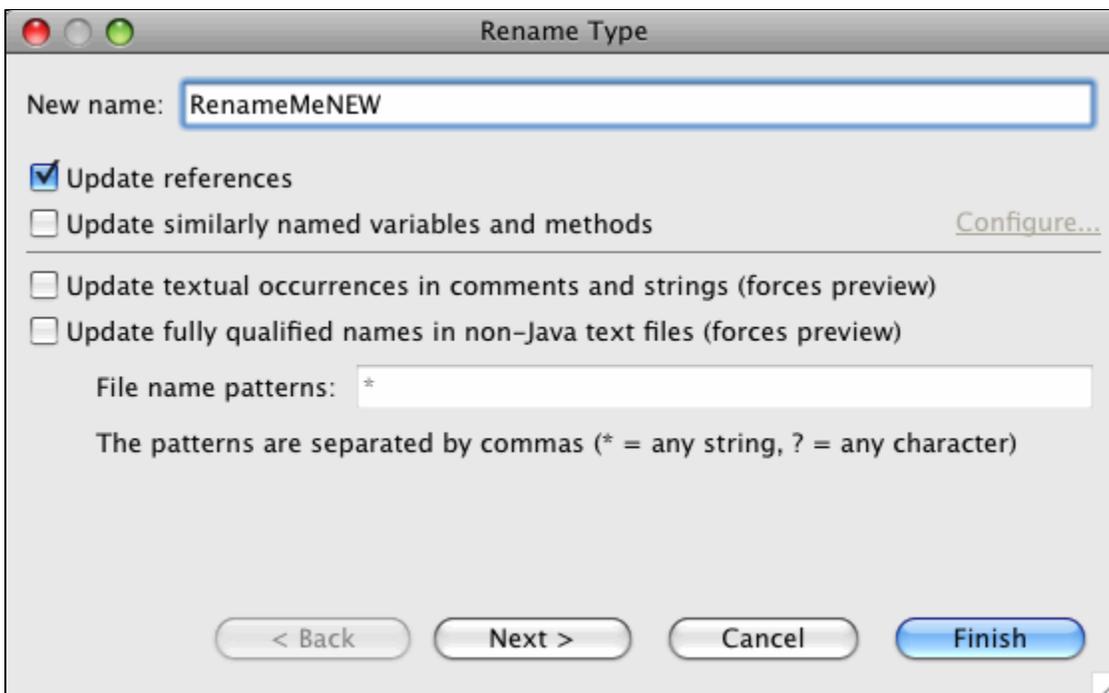
Furthermore, you can rename a class inside of a Groovy compilation unit. To start the refactoring, select a class reference or declaration and select *Rename...* from the Groovy refactoring menu:



Then the following choice appears:



Choose the *Groovy* refactoring to rename only the class and its references, but not the file itself. Choose the *Java* refactoring to rename the file as well as the class. You can then see the standard renaming options from the rename dialog:



After typing the new name and clicking finished, the Groovy class, its file (only if *Java* refactoring is selected), and all

references (both Groovy and Java) are properly renamed:

```
1 package foo;
2
3 class RenameMeNEW {
4
5     RenameMeNEW() {
6
7     }
8
9     RenameMeNEW foo() {
10        return new RenameMeNEW();
11    }
12 }
13
```

```
1 package foo;
2
3 public class SubRename extends RenameMeNEW {
4
5 }
6
```

#### Navigate to super class

Clicking on an override indicator in the gutter will navigate the caret to the super class's implementation of a method:

```
3 class RenameMe {
4     RenameMe foo() {
5         return new RenameMe();
6     }
7 }
8 class SubRenameMe extends RenameMe {
9     @Override
10    public RenameMe foo() {
11        return super.foo();
12    }
13 }
14
```

#### JavaDoc viewing

Hovering on a Groovy method field or class will bring up the JavaDoc/GroovyDoc comment for that declaration. For more information, see the [Groovy eclipse and javadoc hovers](#) blog post.

#### Generated getter/setter navigation and JavaDocs

Hovering over a getter or setter generated by a property node will show the JavaDoc/GroovyDoc of the originating field:

```

class GroovyClass {
    /**
     * This is the {@link GroovyClass#x} field of
     * {@link GroovyClass}
     */
    def x
}

new GroovyClass().getX()

```

Object foo.GroovyClass.x  
This is the [GroovyClass.x](#) field of [GroovyClass](#)

And pressing F3 or CTRL-Click when the generated getter or setter is selected will navigate the cursor to the property node's definition (i.e., the field).

#### Basic content assist options

There are now some basic options to control how content assist proposals are entered into the Groovy editor. You can control whether or not parentheses are used for entering method proposals. And you can also control whether closure arguments are entered with an empty closure `{ }` or as a regular parameter.

Groovy Content assist options to make your content assist Groovier.

- Do not use parens around methods with arguments
- Use brackets for closure arguments

#### Issues fixed

Errors were reported by the JIRA trusted connection.

- BAD\_REMOTE\_IP; Request not allowed from IP address: {0}; ["63.246.24.165"]

Issues addressed for RC1 (0 issues)										
Type	Key	Summary	Assignee	Reporter	Priority	Status	Resolution	Created	Updated	Due

**If the above doesn't show for you, go directly to the [jira issue tracker](#)**

Thanks

Thank you to the community for diligently trying out new releases, raising bugs, suggesting fixes, and discussing features. If you want to participate in the discussion, join the [mailing list](#) and raise bugs in our [issue tracker](#).

#### Groovy Eclipse 1.5.7 Release Notes

The time has come to release the Groovy Eclipse plugin to the world. First off congratulations to the Groovy Eclipse development team, in particular those who worked on the Refactoring feature. These are the features that have been resolved for this release:

⚠ This release does not yet contain the joint compilation feature, as such we still need to lumber on with using both the Groovy Compiler and the Eclipse JDT Java builder on our projects. As a consequence, the bin-groovy library will need to remain there for the time being. This is a top priority for the next release. Still note that there are a few complications that were uncovered during the last attempt, that need to be resolved.

GRECLIPSE-164 Synchronize access to Groovy Projects inside the GroovyProject class

GRECLIPSE-162 Multiple UnsupportedClassVersionError in Eclipse Plugin

GRECLIPSE-159 output directory not created

GRECLIPSE-155 Adding multiline block comments with Ctrl+Shift+/ removes start tags from regular comments

GRECLIPSE-153 NPE when creating new project.

GRECLIPSE-146 Support auto format

GRECLIPSE-141 Groovy rebuild of a project will not acknowledge cancellation or acquire workspace lock

GRECLIPSE-139 Adding Groovy to a Java project leads to strange errors

GRECLIPSE-129 Eclipse Groovy Plugin not working for Eclipse Ganymede (Launched on 25th June)

GRECLIPSE-122 Update to Groovy 1.5.6

GRECLIPSE-109 New Class Wizards fail if superclass is empty or does not exist

GRECLIPSE-108 New Class Wizards fail if superclass is empty or does not exist

GRECLIPSE-106 org.codehaus.groovy\bin should not be under svn control

GRECLIPSE-100 Provides the JavaDocs for the Groovy dependency offered by the Groovy Classpath Container

GRECLIPSE-93 Go to file from test case is off by one line

GRECLIPSE-91 Extension of the GroovyCompiler class to generate an AST without resolving the classes

GRECLIPSE-84 Unremovable Groovy JAR

GRECLIPSE-43 Provide refactoring capabilities (eclipse)

See more on refactoring on the following [wiki page](#).

GRECLIPSE-41 Implement code folding for GroovyEclipse

GRECLIPSE-30 Category org.eclipse.debug.ui.category.debug used for command does not exist

GRECLIPSE-26 ClassCastException during code completion

GRECLIPSE-14 Breakpoints on some lines don't hit

## Eclipse Plugin FAQ

- [Requirements and Installation](#)
  - [Q. Where can I install Groovy-Eclipse?](#)
  - [Q. What version of Eclipse do I need?](#)
  - [Q. How do I install from an update site?](#)
  - [Q. How do I get started?](#)
  - [Q. Existing groovy projects](#)
  - [Q. So you modify the JDT Compiler - will that affect my pure Java projects?](#)
  - [Q. What version of groovy does Groovy-Eclipse use?](#)
  - [Q. How do I change the compiler version in Groovy-Eclipse?](#)
  - [Q. Before we upgrade to the new groovy eclipse-plugin \(V1.x\), should we remove the prior Groovy plugin?](#)
  - [Q. I am using Aptana Studio and cannot install Groovy-Eclipse.](#)
  - [Q. I am running Linux and cannot install Groovy-Eclipse.](#)
- [Feature set](#)
  - [Q. Why is groovy-aware syntax highlighting not working?](#)
  - [Q. What is the theme of the first release?](#)
  - [Q. Can I use this compiler on the command line?](#)
  - [Q. Does this release support joint Groovy-Java projects?](#)
  - [Q. Does it support incremental compilation?](#)
  - [Q. Does it support JUnit?](#)

- [Q. Is there any integration with Maven?](#)
- [Q. Does it support the AST transformations included with Groovy?](#)
- [Q. Does it support custom AST transformations?](#)
- [Q. What can I expect in terms of code assist, navigation, debug and other eclipse facilities?](#)
- [Q. How does debugging work?](#)
- [Q. Where is the Groovy Console?](#)
- [Q. When I load a Groovy file, why is the outline view blank?](#)
- [Q. Where can I get Grails support?](#)
- [Q. Whoa, those editor colors are psychedelic - can I have Java colours?](#)
- [Troubleshooting](#)
  - [Q. Help! The Eclipse installer \(p2\) won't let me Groovy-Eclipse!](#)
  - [Q. Help! The Eclipse installer \(p2\) won't let me upgrade my plugin!](#)
  - [Q. I am getting a MissingPropertyException when running inside of Eclipse, but not when I run outside. What gives?](#)
- [Raising issues and asking questions](#)
  - [Q. Where should I report issues?](#)
  - [Q. Where can I raise questions?](#)
  - [Q. What are the open bugs?](#)

#### Requirements and Installation

## Q. Where can I install Groovy-Eclipse?

A. The source code for the new version is all in codehaus.

### Groovy-Eclipse update site

To install 2.0.0 in Eclipse 3.5.x, add the following URL to your Eclipse update manager:

<http://dist.springsource.org/release/GRECLIPSE/e3.5/>

To install in Eclipse 3.4.2, use the following URL:

<http://ci.repository.codehaus.org/greclipse/snapshot/e34/>

Note that 3.4 support is available, but we are no longer actively working on it.

## Q. What version of Eclipse do I need?

A. You will need Eclipse 3.6.x for the 3.6 version of the plugin, Eclipse 3.5.x the 3.5 version of the plugin and Eclipse 3.4.2 for the 3.4 version of the plugin. Since Groovy-Eclipse relies on Eclipse's Java tooling (the JDT), almost any flavor of Eclipse that includes the JDT will be compatible. This includes the Java IDE, Classic, RCP, JavaEE, etc. Flavors such as Eclipse for PHP developers are not compatible because it does not include the JDT. If you are interested in Grails tooling, then you must install the [SpringSource Tool Suite](#).

## Q. How do I install from an update site?

A. Navigate to "Help > Software Updates". On the "Available Software" tab, click "Add Site" and enter the update site URL appropriate for your version of Eclipse: <http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.6> or <http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.5> then click OK. Now the update site will be on the list, open it and mark the entry 'Groovy-Eclipse plugin'. Finally, in the top right, click "Install" and follow the dialogs to complete installation.

## Q. How do I get started?

A. Once installed you can create new Groovy projects in a similar way to how you can create Java projects. Or if you have an existing Java project which you want to add groovy code too, you can select the project in the package explorer, right click and navigate to "Configure" > "Convert to Groovy Project". This action adds the Groovy nature to your project. Once this happens, the icon decorator for your project will change to reflect the new Groovy nature. And any .java files, the builder for the project will also build .groovy files. Finally, if you have some older groovy projects created with the previous version of the plugin, they must be migrated to exploit the new plugin. See the next question:

## **Q. Existing groovy projects**

A. Existing projects that are based on the old groovy plugin need to be migrated. There is a migration tool included with the plugin. To use the tool, select the projects that you want to migrate, right click, select the Groovy sub-menu, and then choose "Convert legacy groovy projects". Note that this option will only appear if you are selecting projects that have the old groovy nature.

## **Q. So you modify the JDT Compiler - will that affect my pure Java projects?**

A. The changes to the JDT compiler are minimal, and all the extra extensions added to it are only executed if a project has the groovy nature. Any adverse affects on building pure Java projects should be considered a bug.

## **Q. What version of groovy does Groovy-Eclipse use?**

A. Groovy-Eclipse ships with Groovy 1.7.x. It is possible to install a Groovy 1.8 compiler, but you must install from the development update site. See [Compiler Switching within Groovy-Eclipse](#) for more information on this.

## **Q. How do I change the compiler version in Groovy-Eclipse?**

You can change the version by going to Preferences -> Groovy -> Compiler. If this does not work for you, follow the instructions in [Compiler Switching within Groovy-Eclipse](#). Currently, the 1.7.x and 1.8.x streams are supported.

## **Q. Before we upgrade to the new groovy eclipse-plugin (V1.x), should we remove the prior Groovy plugin?**

A. Yes, there may be some unexpected interactions between the two versions of the plugin. However, once you are using a 2.x version of the plugin, you do not need to uninstall before upgrading (see also previous question).

## **Q. I am using Aptana Studio and cannot install Groovy-Eclipse.**

A. Aptana Studio does not come with the JDT (Java Development Tools) pre-installed. This is something that Groovy-Eclipse is built on top of and is therefore a requirement. You must therefore install JDT first. Here's how:

1. Go to your update manager
2. Select the Galileo - <http://download.eclipse.org/releases/galileo> update site (or the Helios or Indigo update site, whichever is relevant)
3. Choose the Category Programming Languages -> Java Development tools
4. Install
5. (No need to restart)

6. Now install Groovy
7. Restart and things should be working

## **Q. I am running Linux and cannot install Groovy-Eclipse.**

You may be running into [GRECLIPSE-498](#). Make sure that you are using an Eclipse version downloaded from Eclipse.org, and not one that has come from your Linux distribution's package manager. Using a repackaged Eclipse may not work because when Linux distributions repackage Eclipse, they typically change the feature structure so that it is not possible to install the JDT feature patch.

Feature set

## **Q. Why is groovy-aware syntax highlighting not working?**

A. The most likely cause for this is that you are editing your groovy file in a Java editor. Look at the editor icon. Is it a groovy icon or a java icon? If it is a Java icon, you can change to a groovy editor like this:

1. close the editor.
2. right click on the file
3. Select Open With -> Groovy Editor

Also, please note that lines with multiple slashes are sometimes interpreted as regular expressions when they shouldn't be (see [GRECLIPSE-124](#)).

## **Q. What is the theme of the first release?**

A. See the [New and Noteworthy](#) for a list of features.

## **Q. Can I use this compiler on the command line?**

A. We have not yet released the batch compiler for Groovy, but we expect to do so for the 2.0.1 release (see the [Groovy-Eclipse Roadmap](#)).

## **Q. Does this release support joint Groovy-Java projects?**

A. Yes. You may have references between Groovy and Java files within or across projects.

## **Q. Does it support incremental compilation?**

A. Yes. If you make small changes to either java or groovy files in a mixed project and save them, they will be compiled together with any directly affected by the change.

## **Q. Does it support JUnit?**

A. Yes. All of your Groovy unit tests can be run using the standard JUnit launcher.

## **Q. Is there any integration with Maven?**

A. Yes. There are two options: GMaven and the groovy-eclipse-compiler. Each has different advantages. See the full discussion at [Groovy-Eclipse compiler plugin for Maven](#).

## **Q. Does it support the AST transformations included with Groovy?**

A. Yes. Although any changes introduced by the transformations (eg. new methods) are not currently visible to code in .java files, so you will not get code completion on them or be able to directly write code in .java files to call them. However, these methods will be available via content assist in Groovy files.

## Q. Does it support custom AST transformations?

A. Custom Ast transforms are supported and will be picked up from the project classpath during a build. If you wish to define and consume transforms within the same project, there is a restriction: the transform must be written in Java (not groovy) and it cannot have dependencies on groovy source in that same project. This restriction exists because Eclipse does not have a notion of separate source folders that can be separately compiled - in eclipse the source within all source folders is compiled together.

## Q. What can I expect in terms of code assist, navigation, debug and other eclipse facilities?

A. Content assist and navigation are available and take advantage of Groovy-Eclipse's [inferencing engine](#) and so can provide fairly sophisticated results. Many other Eclipse facilities, such as setting breakpoints, refactoring and code formatting are available. Debug support is also available. Please see the [Groovy-Eclipse Roadmap](#) for a discussion on what to expect in the future (and please leave your own comments).

## Q. How does debugging work?

A. Basic debugging works as it does in Java. You can set a line breakpoint by double clicking on the gutter marker next to where you want the breakpoint to go. You can right click to enable/disable the breakpoint. You can edit the breakpoint properties to make it a conditional breakpoint (conditions must be specified in Java and may require explicit casting). The instruction pointer will stop at breakpoints, and you can use the standard step into, step over, and step outcommands from Java. The variables window will show all variables that are in scope, just like when in Java. However, you will find that some of the runtime types of the variables are not what you may expect. For example, variables declared inside closures will be of type groovy.lang.Reference. Method entry and class loading breakpoints are not yet supported.

If you want a more complete debugging experience, you should install the [Grails Tooling](#), which also includes some enhanced debugging capabilities for Groovy code. Grails Tooling provides support for evaluating code snippets in the context of the currently executing program. This allows the display view, the expressions view, and inline evaluations to work as expected. See [New Groovy Debug Support in STS 2.5.1](#).

## Q. Where is the Groovy Console?

A. The Groovy Console feature of the old plugin has been removed. So far, no one has asked for this functionality. If you would like to see it, raise a jira issue (see below) for it or mention it on the mailing list (see below).

## Q. When I load a Groovy file, why is the outline view blank?

A. The outline receives structural information from the Groovy compiler. If there is a syntax problem that is preventing the compiler from parsing the Groovy file, then the outline view will remain empty until because the compiler cannot provide any information about the file's contents.

## Q. Where can I get Grails support?

A. Grails support is not included in this plugin. However, it is available in the SpringSource Tools Suite (STS). You can download STS [here](#). For a basic description of how to install Grails into STS, create a new project or import an

existing project into STS, [here is a tutorial](#) This tutorial also covers how to use the pop-up grails console to invoke grails commands.

## Q. Whoa, those editor colors are psychedelic - can I have Java colours?

A. To switch to Java colors, open the eclipse preferences, go to Groovy > Editor and click 'Copy Java Color Preferences'

### Troubleshooting

## Q. Help! The Eclipse installer (p2) won't let me Groovy-Eclipse!

A. Groovy-Eclipse cannot be installed into a read-only directory. The Eclipse installation must be writable by the current user. *Shared installs* are not supported. This is because Groovy-Eclipse ships with a feature patch for JDTC and feature patches cannot be installed into shared installs. This includes protected directories on Windows, like `C:\Program Files`. For more information on this problem see [Eclipse Bug 395516](#).

As a workaround, you can temporarily provide write access to the current user in the Eclipse directory, restart Eclipse, and install Groovy-Eclipse. Once Groovy-Eclipse is installed, you no longer need write access to the install location.

## Q. Help! The Eclipse installer (p2) won't let me upgrade my plugin!

It is likely that you are hitting this [Eclipse bug](#), which should be fixed in Eclipse 3.6 and beyond.

Try the following:

- Use the *Install new Software* menu item instead of *Check for updates*.
- Uninstall your current version of the plugin and then reinstall the new version.

## Q. I am getting a MissingPropertyException when running inside of Eclipse, but not when I run outside. What gives?

A. You could be hitting this issue: <http://jira.codehaus.org/browse/GRECLIPSE-664>.

It could be that you have the Groovy Libraries on the classpath twice. This may happen, for example, if you originally imported a Maven project and then added the Groovy Nature separately. The solution is to remove one version of the Groovy Libraries from your classpath.

### Raising issues and asking questions

## Q. Where should I report issues?

A. <http://jira.codehaus.org/browse/GRECLIPSE>

Be sure to specify version 2.0 of the plugin.

## Q. Where can I raise questions?

A. Subscribe to the groovy eclipse plugin user's mailing list:

<http://xircles.codehaus.org/lists/eclipse-plugin-user@groovy.codehaus.org/subscribe>

There is also a developer's mailing list:

<http://xircles.codehaus.org/lists/eclipse-plugin-dev@groovy.codehaus.org/subscribe>

## Q. What are the open bugs?

A. See below:

Errors were reported by the JIRA trusted connection.

- BAD\_REMOTE\_IP; Request not allowed from IP address: {0}; ["63.246.24.165"]

JIRA Issues (0 issues)										
Type	Key	Summary	Assignee	Reporter	Priority	Status	Resolution	Created	Updated	Due

### Groovy-Eclipse 2.0.0M1 New and Noteworthy

- [Overview](#)
- [Core](#)
- [User Interface](#)
- [Run/Debug](#)
- [Refactoring/Formatting](#)
- [Issues addressed for M1](#)

#### Overview

The Groovy-Eclipse team is proud to announce the first milestone release of the Groovy-Eclipse plugin. In this release, we have focused on ensuring that the basic edit-compile-run/test/debug loop works as smoothly as possible. Unlike version 1 of the plugin, this version is based upon a new approach to integrating the groovy compiler into eclipse. We have extended the Eclipse JDT compiler to be pluggable by other compilers for Java-like languages. We have plugged in `groovyc` so that the JDT can build and manipulate both Java and Groovy files in the same project (for more information, please read our earlier blog post, [A Groovier Eclipse Experience](#)). Importantly, this approach enables incremental compilation to work for a multi-language project. We have also put significant effort into ensuring that Groovy developers can naturally use the same standard Eclipse Java facilities that Java developers have become reliant on. This includes facilities like: JUnit testing, debugging, content assist, early error indication, open-type, and outline views. Full feature details are below.

The update site (and all future milestone update sites) is here:

**<http://dist.springsource.org/milestone/GRECLIPSE/e3.5/>**

#### Core

#### Cross language dependencies

Full support for any kind of dependency between `.java` and `.groovy` files is supported.

```
A.java
1
2 public class A {
3     public Object method() { return "A"; }
4 }
5

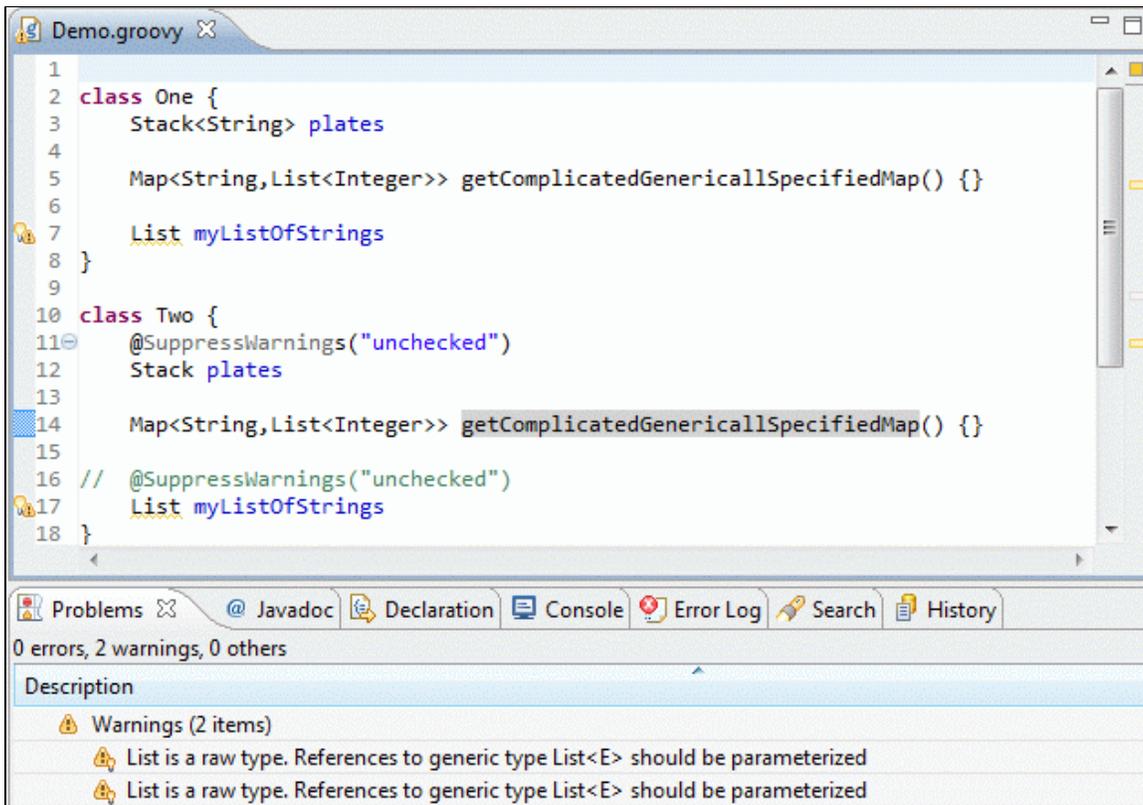
B.groovy
1
2 class B extends A {
3     def method() { return 'B' }
4 }

C.java
1
2 public class C extends B {
3     @Override
4     public Object method() {
5         return "C";
6     }
7
8 }

D.groovy
1
2 class D extends C {
3
4     def method() { return 'D' }
5
6     static void main(args) {
7         print new D().method();
8     }
9 }
```

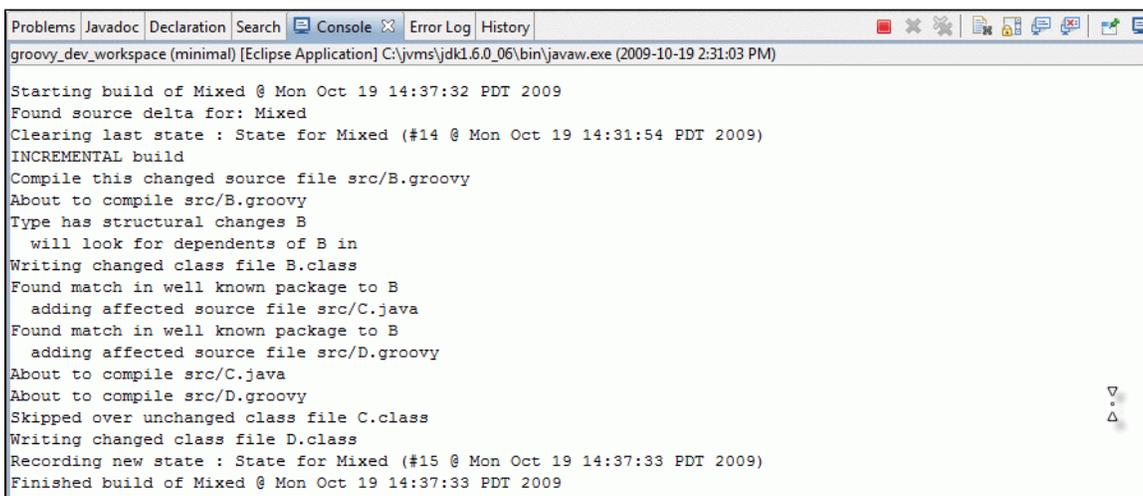
### Generics Checking

Since JDT can now understand groovy source (to a degree), it applies some generics checking to the groovy code and will report correct warnings that groovyc would not normally show:



### Incremental compilation

Changes made to either .java or .groovy files will result in an incremental compilation of the containing project. Even if the dependencies are in a different language, the impact of the change will correctly propagate.



### AST transforms

Projects may exploit AST transforms that are accessible on the project classpath. Transforms like `@Singleton` that are defined by groovy are immediately accessible due to the groovy libraries being on every groovy project classpath. To exploit transforms provided by third parties, the jars containing the transform implementation must be added to the project classpath prior to usage. As a temporary feature (still under evaluation) in this version, transforms defined in .java files **may** be exploited in the same project they are defined, although on changing the transform it may be necessary to do a full build of the project to pick up the change.

### Compiler level switching

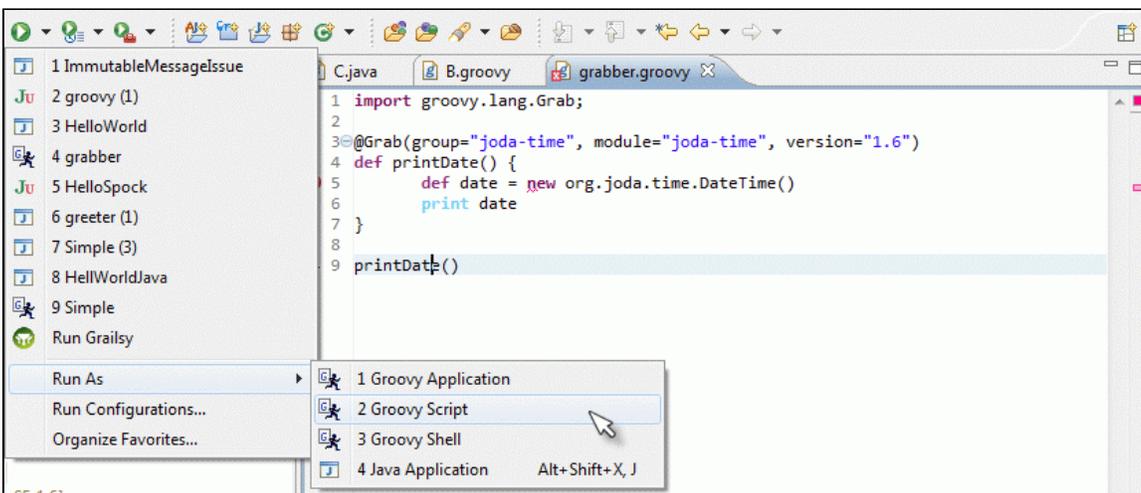
The Groovy-Eclipse plugin M1 ships with both the 1.7-beta2 and 1.6.5 versions of the groovy compiler. Currently, the only way to switch between the two versions is outside of Eclipse. Here is how:

1. Shut down Eclipse
2. Go to eclipse/configuration/org.eclipse.equinox.simpleconfigurator
3. Make a backup copy of bundles.info
4. Open bundles.info in a text editor
5. Find the line for org.codehaus.groovy\_1.7.0 and delete
6. Restart eclipse

To re-enable the 1.7 compiler, simply reinstate the backed up copy of bundles.info.

#### **@Grab annotation**

Basic support for Grape and the Grab annotation is included, but the code using the annotation may show a compile error (due to a missing dependency), however it will run as a groovy script:



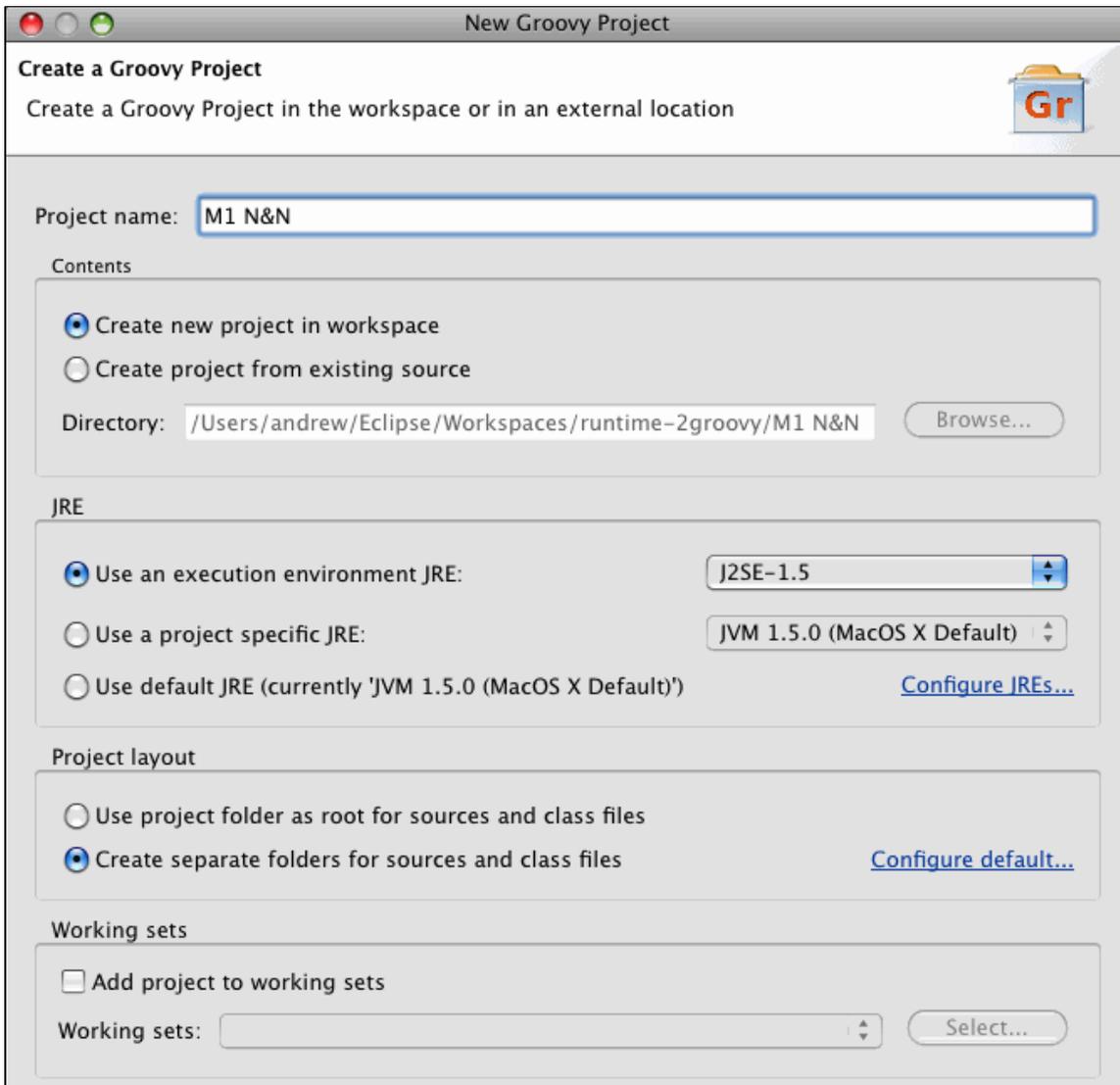
#### **User Interface**

##### **Wizards**

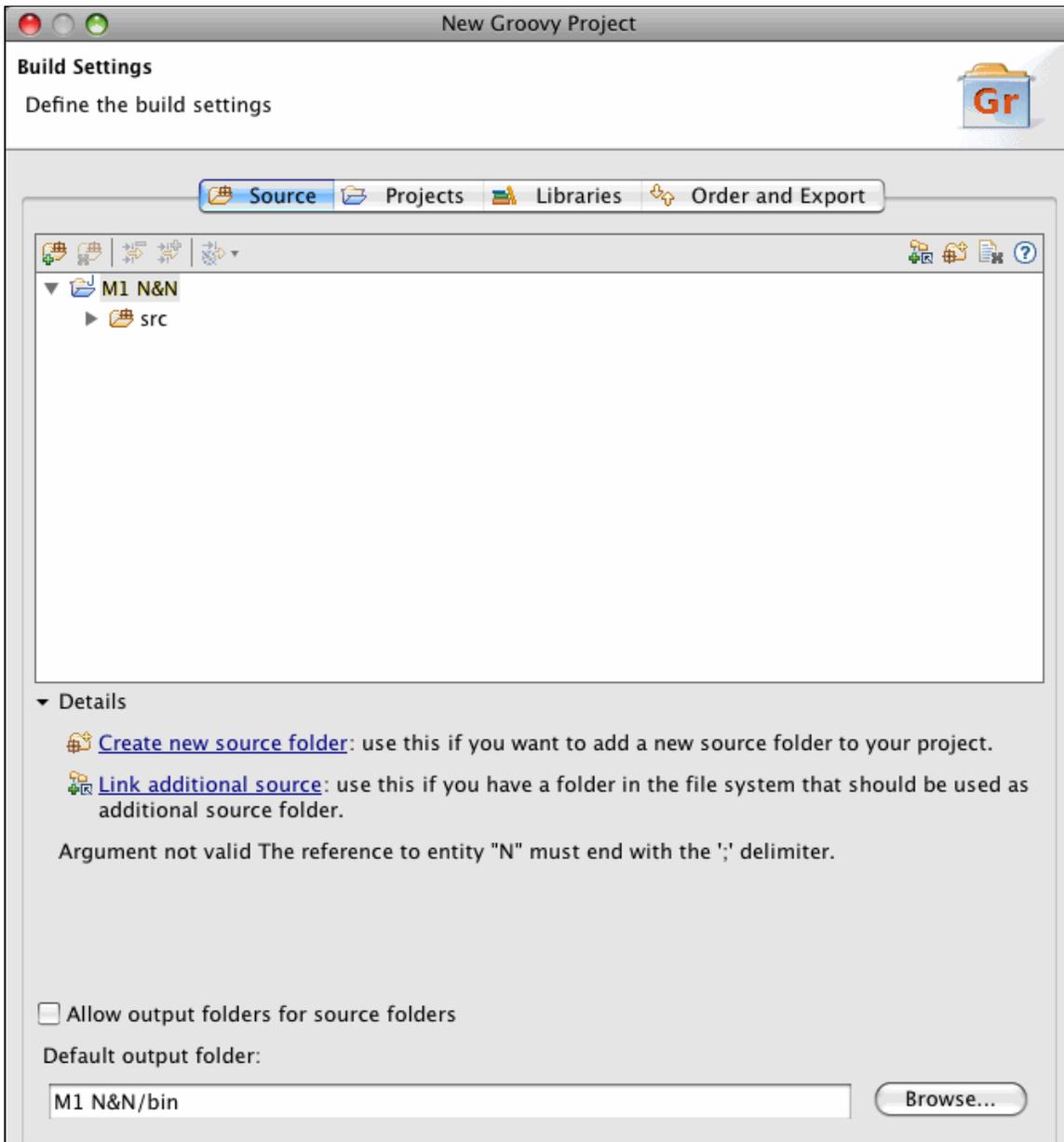
The wizards available mimic their Java counterparts.

## **New Groovy Project Wizard**

The new Groovy project wizard allows you to create an Eclipse project pre-configured with Groovy support:

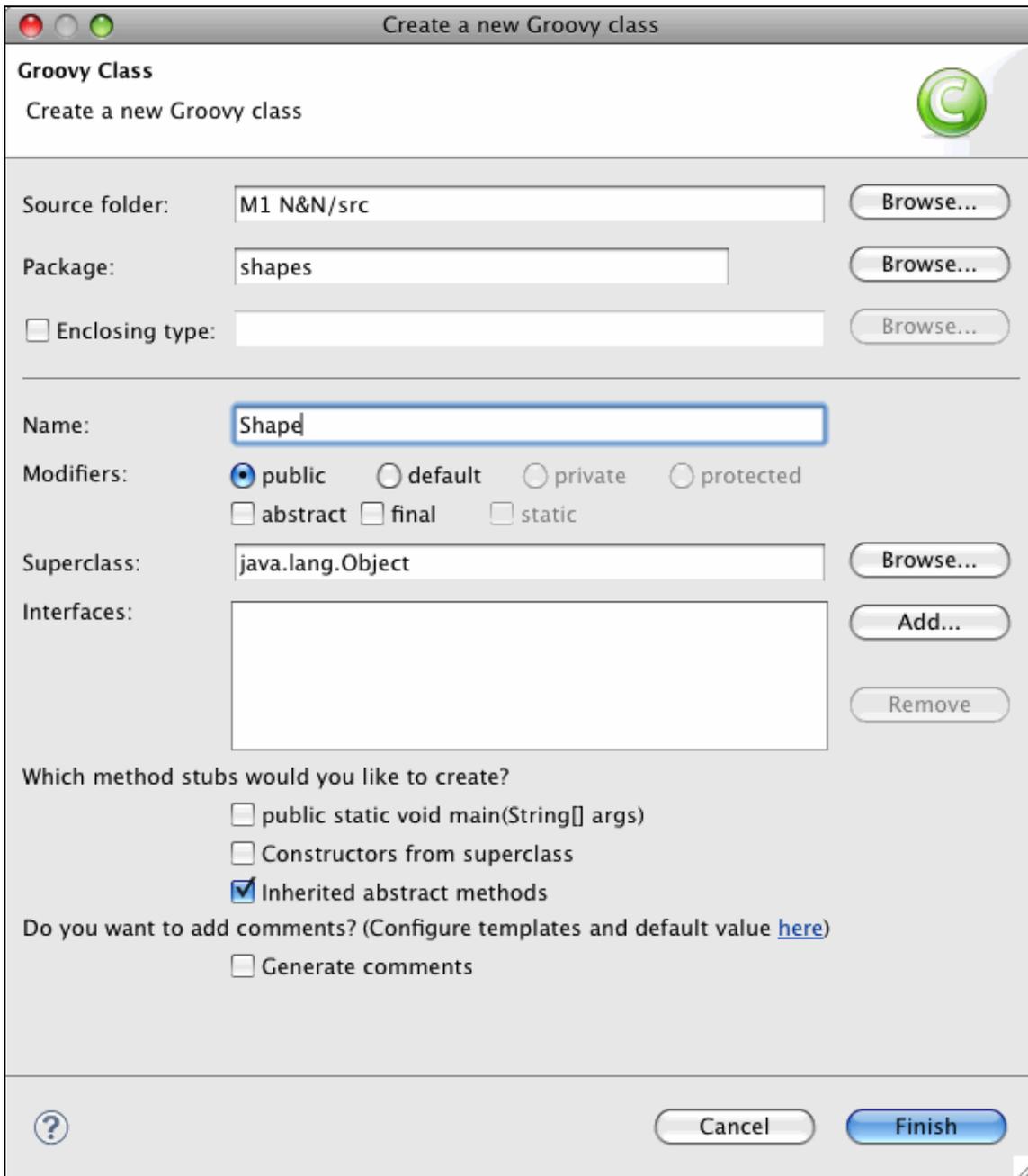


It supports all of the options available for Java projects:



## New Groovy Class Wizard

This wizard creates a new Groovy class, with many of the same options available for the Java Class creation wizard:



## New Groovy Test Case Wizard

This wizard assists you in creating a new Groovy JUnit 3 or JUnit 4 test case:

New Groovy JUnit test case

### JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

New JUnit 3 test  New JUnit 4 test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

setUpBeforeClass()  tearDownAfterClass()  
 setUp()  tearDown()  
 constructor

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Class under test:

#### Convert Legacy projects

On startup, you are reminded to convert any legacy groovy projects in your workspace:

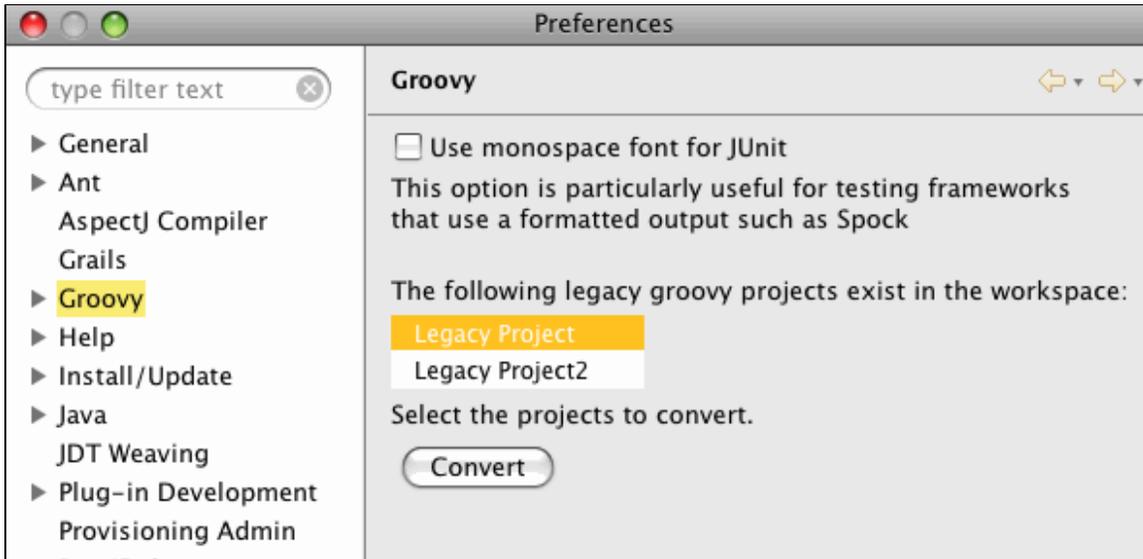
Convert legacy Groovy Projects

 Some of your Groovy projects appear to be incompatible with the new version of the Groovy plugin. Should they be converted now?

They can be converted later from the Groovy Preferences page

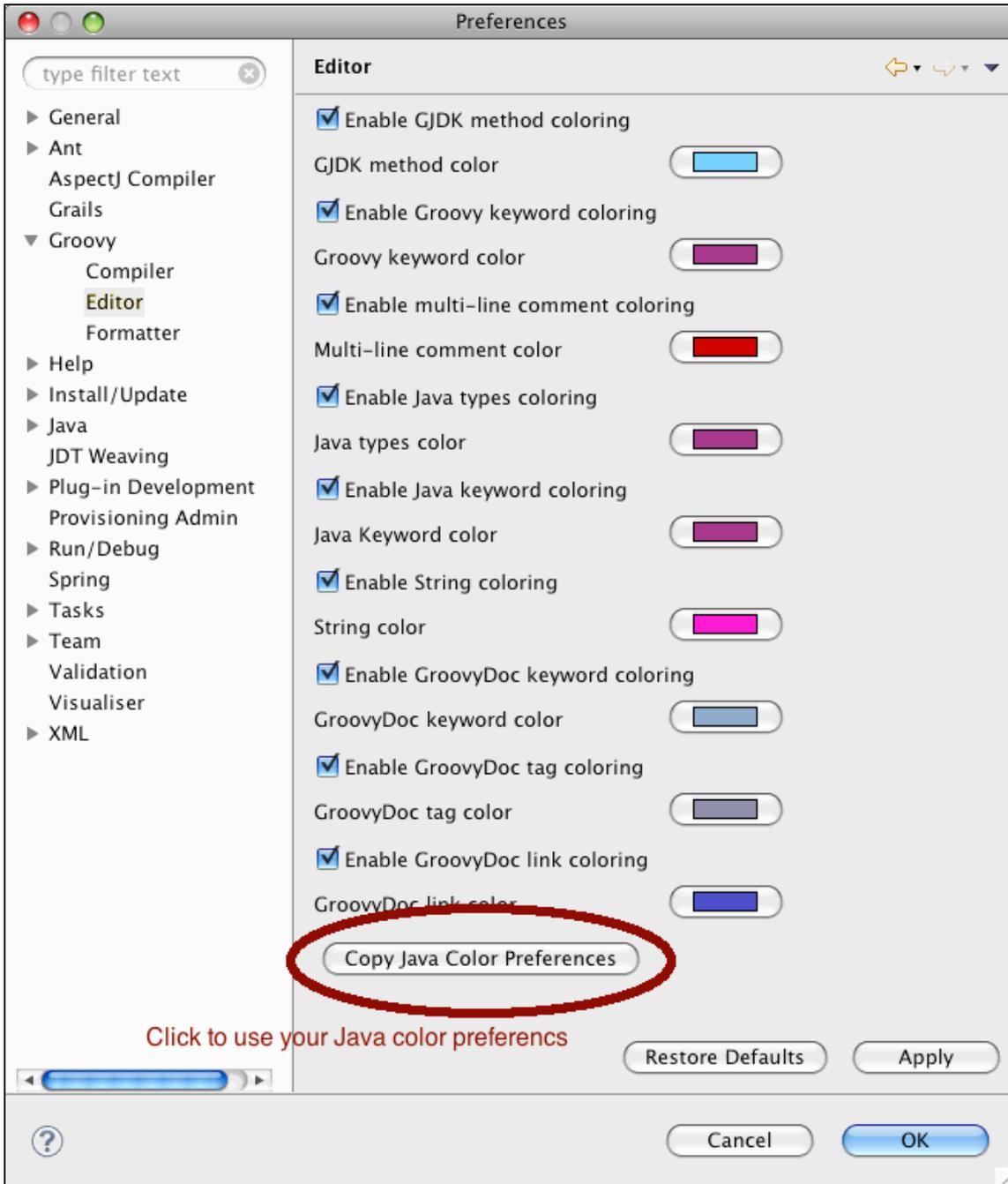
Don't show this message again.

If you can decide to permanently dismiss this dialog box, you can still do the conversion from the preference pages:



***Groovy Editor***

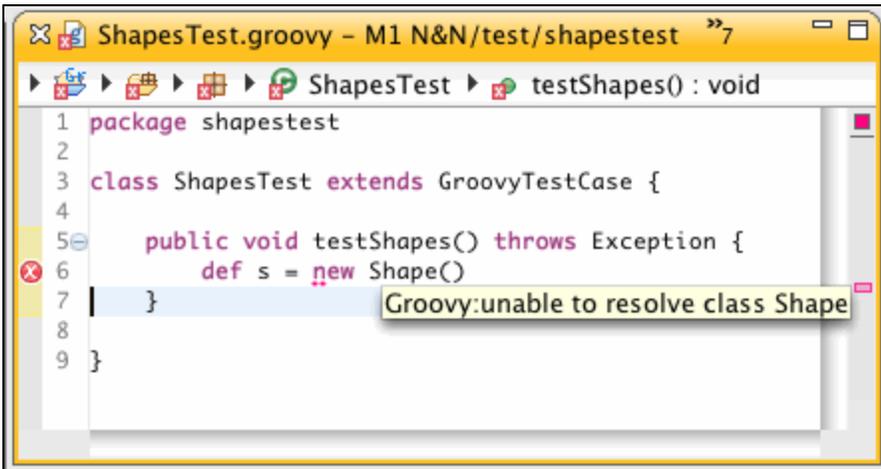
As much as possible, we try to provide the same kind of behavior in the Groovy editor as you would expect in the Java editor. By default, the editor uses a different set of colors from the Java editor in order to emphasize the difference between the two languages. However, if you prefer standard Java syntax coloring, you can select it on the following preference page:



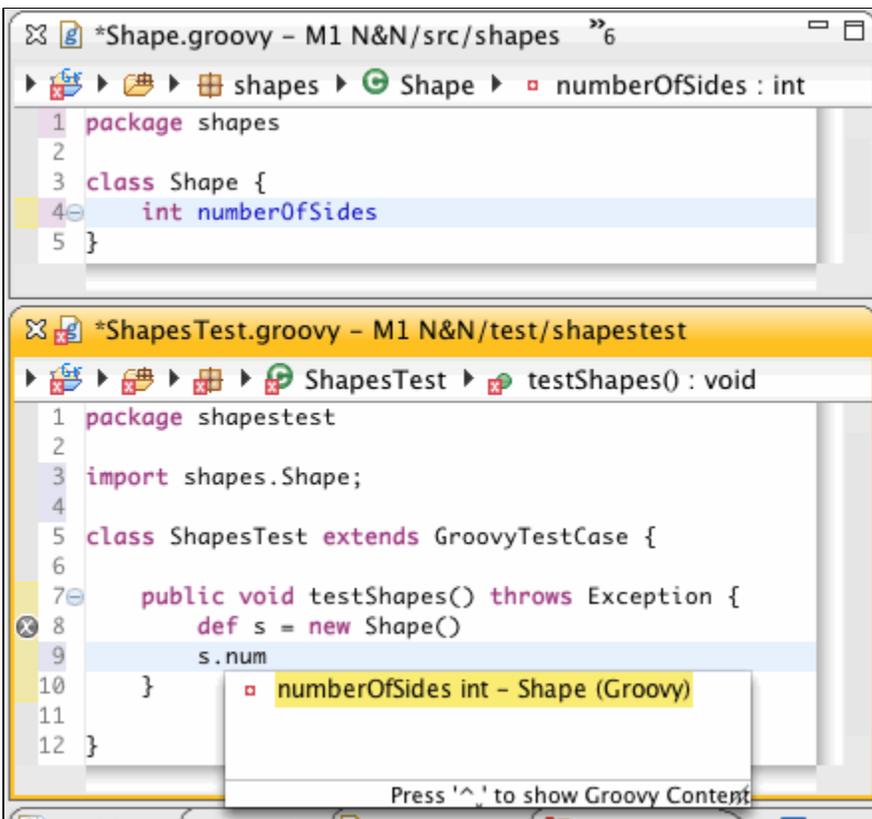
Additionally, the Groovy editor provides the following capabilities.

## Early error indication and reconciling

Errors are highlighted as you type. This functionality is provided by hooking into JDT's reconciling capability:



An additional benefit of reconciling means that content assist proposals that come from files with unsaved changes will appear where expected. In the following example, the Shape class the definition of the numberOfSides property y has not been saved to disk yet, but it appears in the content assist proposals list of the ShapesTest class:



## Content assist

Inferencing content assist has been available since earlier versions of the Groovy-Eclipse plugin. We are continuing to improve on this. For M1, we have focused on reliability and availability of content assist. For M2, we will focus on completeness of results. And for the final release, we will work on some performance tuning.

In following example, notice that there are two levels of inferencing happening. The editor correctly determines that someObject is an int numberOfSides is an int:

```

public void testShapes() throws Exception {
    def s = new Shape()
    def someObject = s.numberOfSides
    someObject.intValue()
}

```

intValue() : int - Number (Groovy)

Returns the value of the specified number as an int. This may involve rounding or truncation.

Returns: the numeric value represented by this object after conversion to type int.

And now, the inferring engine is able to determine that the type of someObject is now a String:

```

import shapes.Shape;

class ShapesTest extends GroovyTestCase {

    public void testShapes() throws Exception {
        def s = new Shape()
        def someObject = s.numberOfSides
        someObject = "I am a shape"
        someObject.indexOf()
    }
}

```

indexOf(int ch) : int - String (Groovy)

indexOf(String str) : int - String (Groovy)

## Navigation

Typing F3, or CTRL-click (CMD-click on Macs) will navigate to the definition of a reference. Navigation hooks into the same inferring engine being used for content assist. So, pressing F3 on the following:

```

public void testShapes() throws Exception {
    def s = new Shape()
    def x = s
    x.numberOfSides
}

```

will open an editor on the declaration:

```

package shapes

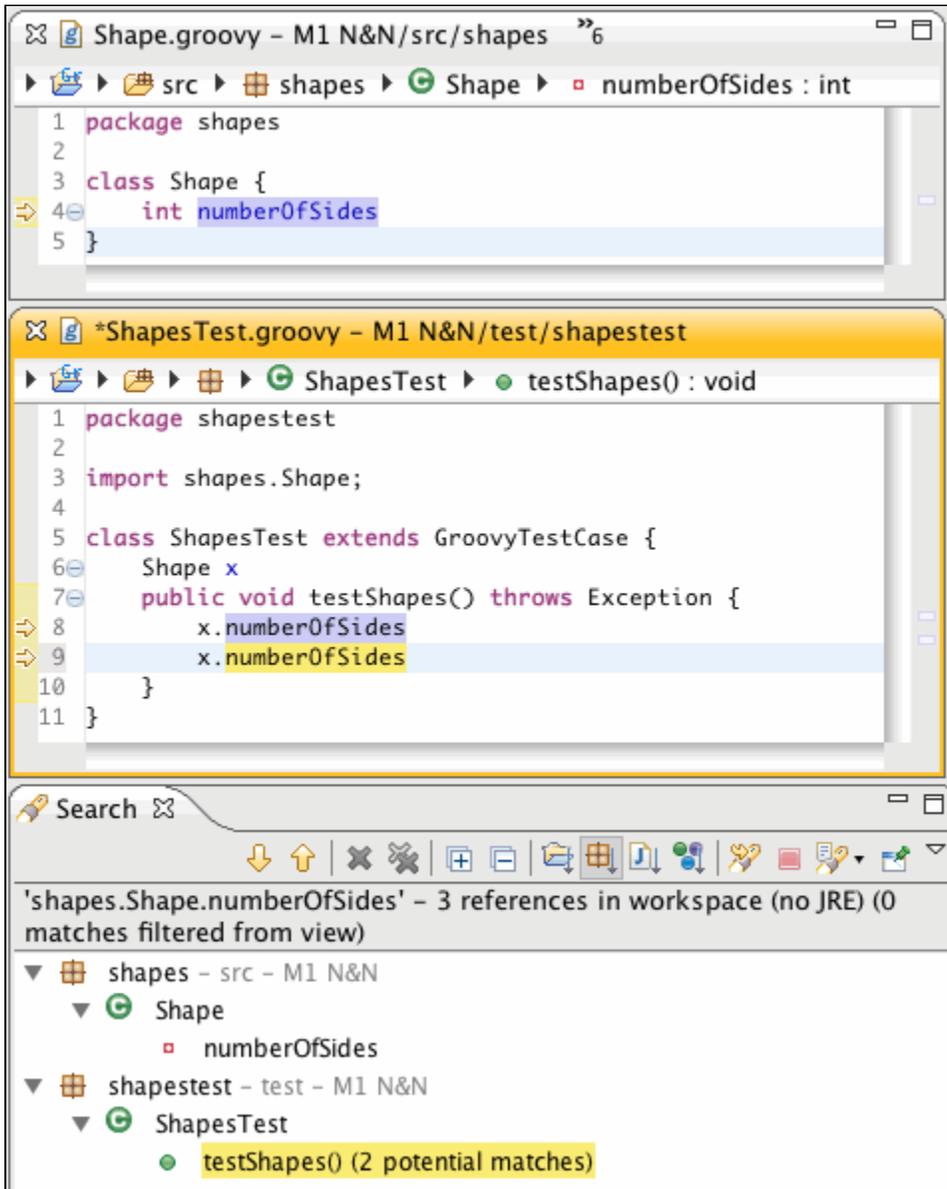
class Shape {
    int numberOfSides
}

```

You can also notice reconciling at work here because the Shape.groovy has not yet been saved since the numberOfSides property has been added.

## Search

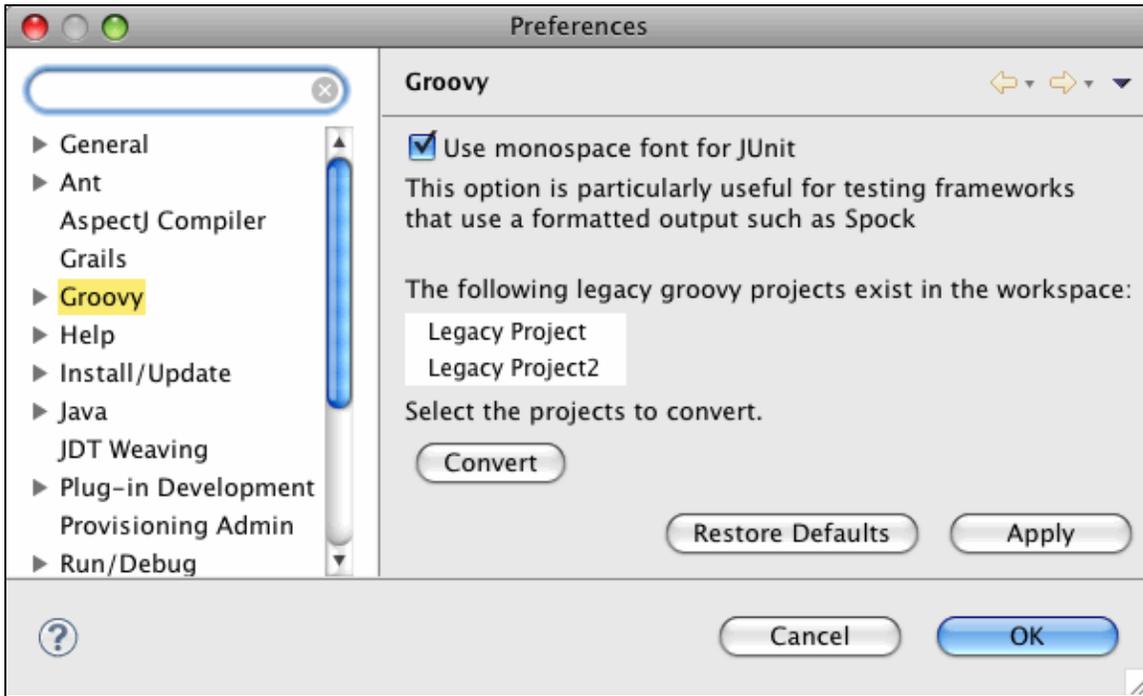
We have introduced some basic search functionality for M1. Currently, search will be able to locate method, field, and type references if they are explicitly typed. Local variable references cannot be found yet. This is a feature that is of primary importance for M2 and will be significantly improved in the coming snapshot builds.



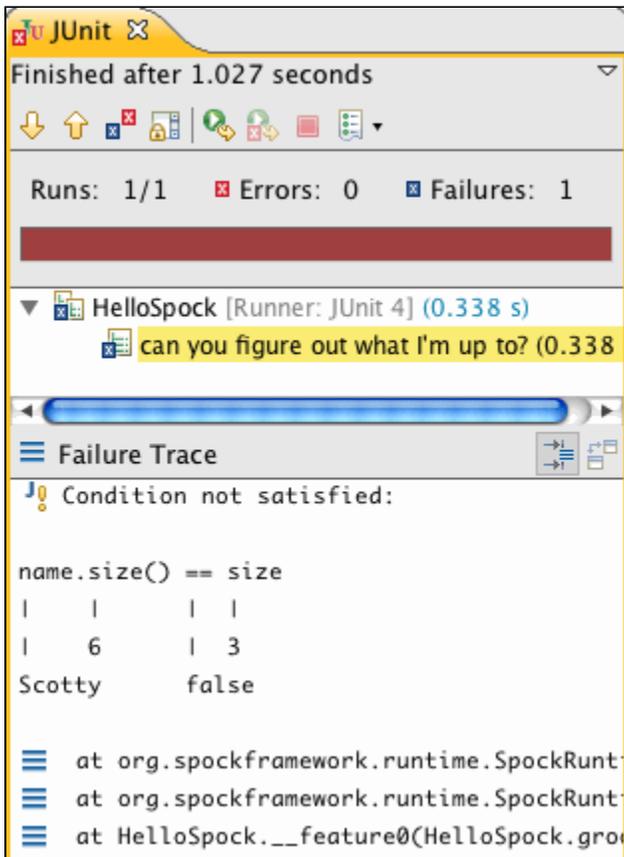
Again, notice the affect of reconciling in that the search results view contains both references to `numberOfSides` even though the text has not yet been written to disk.

## JUnit Monospace font

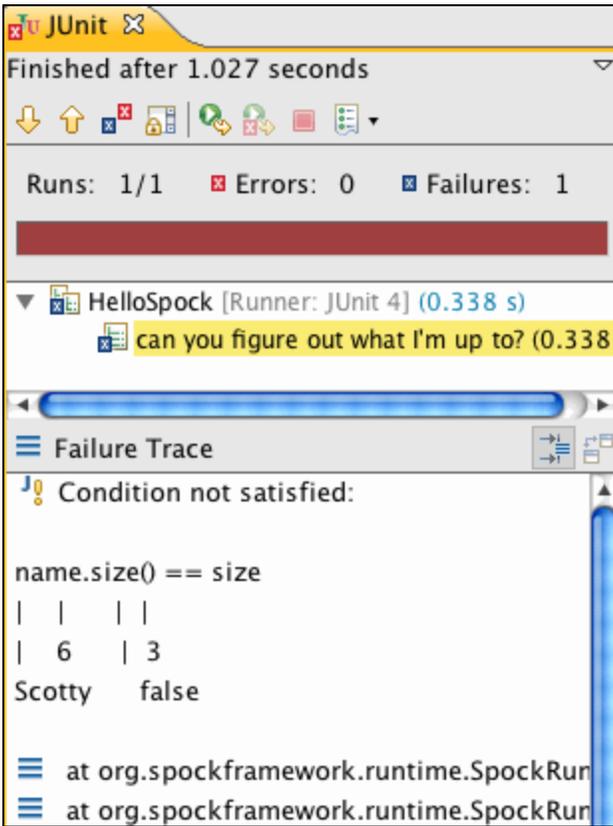
There is an option in the Groovy Preferences page to show the JUnit results pane in monospace font:



This is helpful for viewing the results of Spock tests. With monospace font, the results pane is much easier to read:



than it is without:

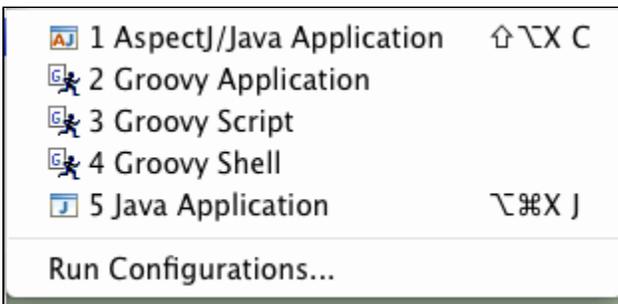


**Run/Debug**

There are two facets to M1's debug support: launching and debugging.

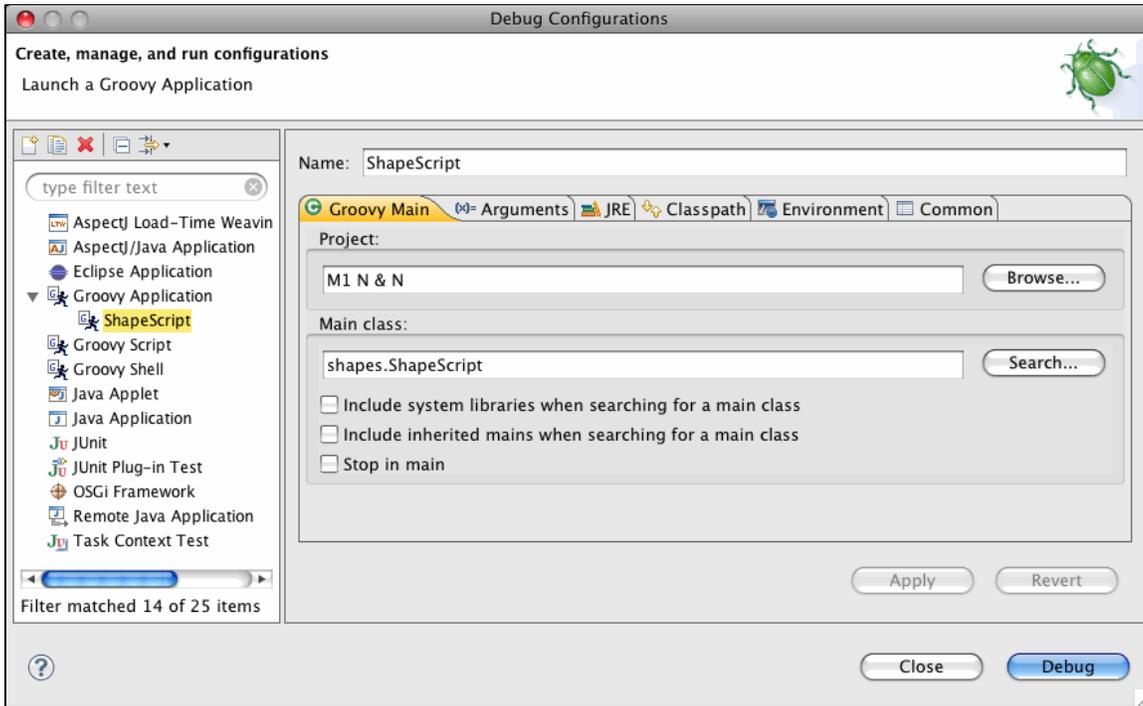
**Launching**

There are three launch options for your Groovy files and projects:



## Groovy Application

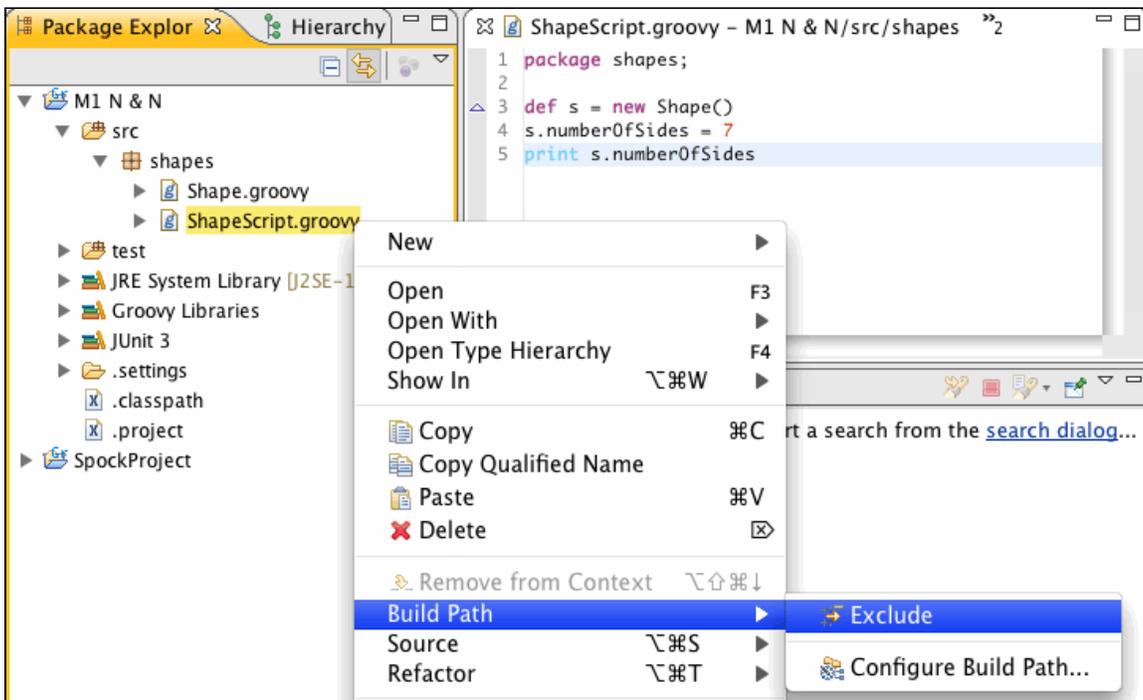
This is similar to launching Java applications. A main method of a Groovy or Java class is launched and you are provided with options similar to launching a standard Java application:



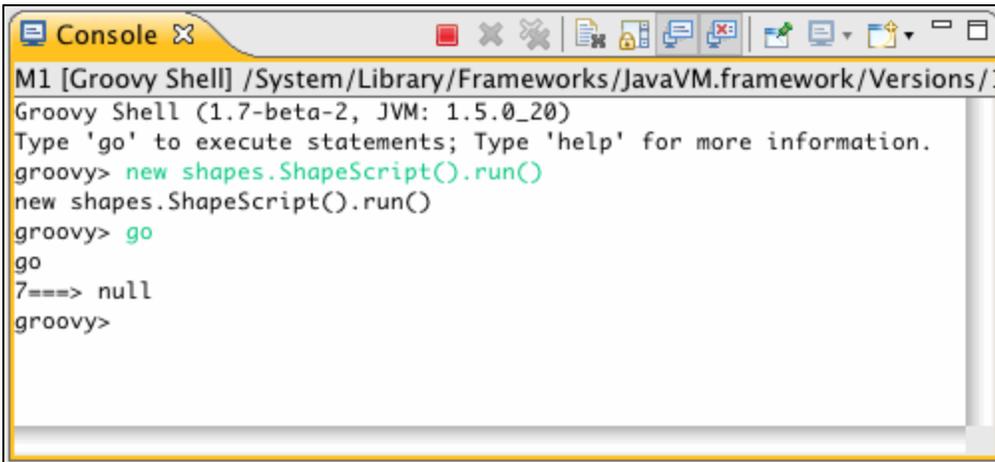
Actually, this launch configuration is so similar to the *Java Application* launch configuration that it may disappear in future versions of the plugin. Every launchable class file generated from a Groovy script is a valid Java program. And so, all Groovy scripts should be launchable as a Java application in Eclipse. This is currently possible, but needs some further testing before we can safely remove the Groovy application launch configuration.

## Groovy Script

This launch configuration launches a \*.groovy file as a script. It is identical to running the `groovy` command from the command line. This launch configuration does not require class files to execute. Only \*.groovy files are necessary. As an example of this, it is possible to explicitly remove a Groovy file from the build:





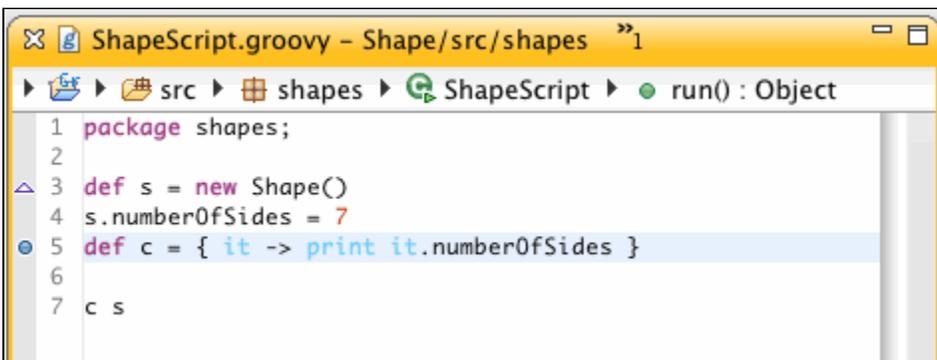


```
M1 [Groovy Shell] /System/Library/Frameworks/JavaVM.framework/Versions/...
Groovy Shell (1.7-beta-2, JVM: 1.5.0_20)
Type 'go' to execute statements; Type 'help' for more information.
groovy> new shapes.ShapeScript().run()
new shapes.ShapeScript().run()
groovy> go
go
7====> null
groovy>
```

Note that there is a bug on windows where the `groovy>` prompt appears twice on each line. See [GRECLIPSE-434](#).

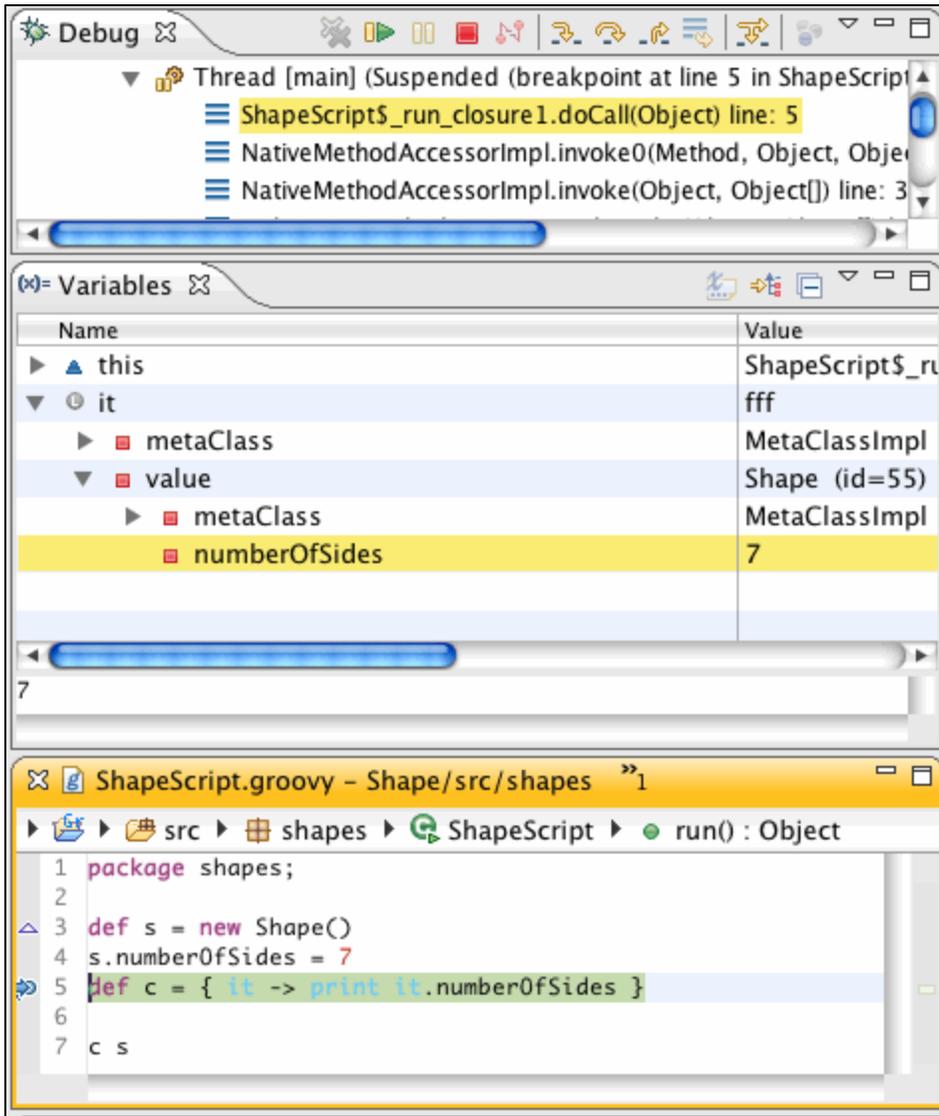
### ***Breakpoints and debugging***

The Groovy-Eclipse plugin allows you to set breakpoints on any statement in a Groovy file, including inside closures, by clicking in the gutter bar next to the line where you want the breakpoint to occur:



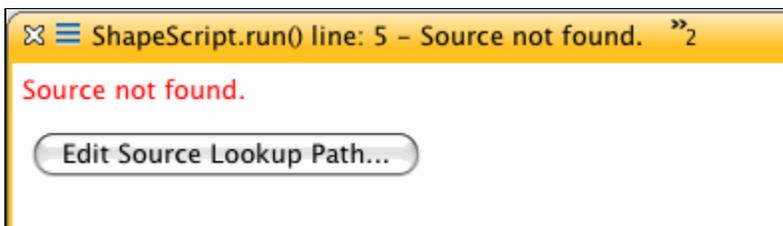
```
ShapeScript.groovy - Shape/src/shapes 1
src ▸ shapes ▸ ShapeScript ▸ run() : Object
1 package shapes;
2
3 def s = new Shape()
4 s.numberOfSides = 7
5 def c = { it -> print it.numberOfSides }
6
7 c s
```

When launched in debug mode as a Groovy Application or Groovy Script, execution will break when a breakpoint is reached:

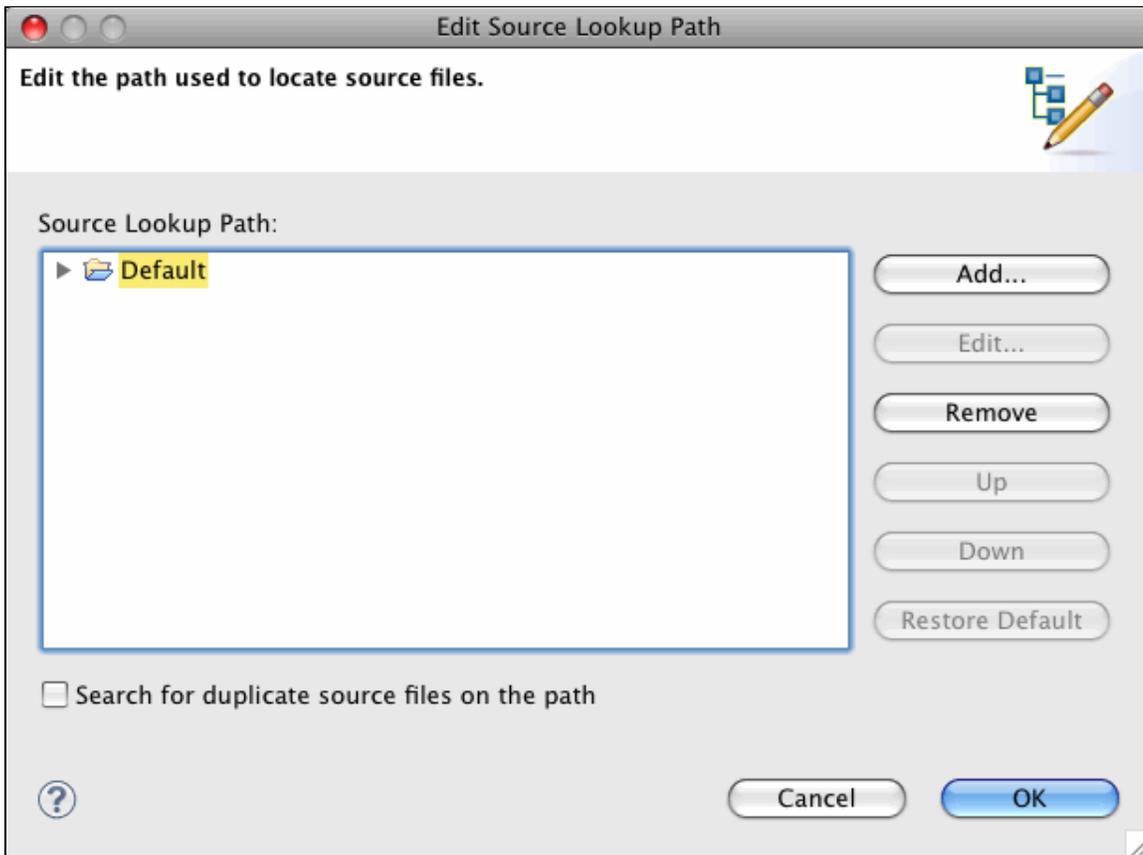


When there is ambiguity as to where the breakpoint should be placed, the debugger stops at both locations. As in the previous example, the debugger will stop at the definition of the closure `c` as well as whenever closure `c` is invoked.

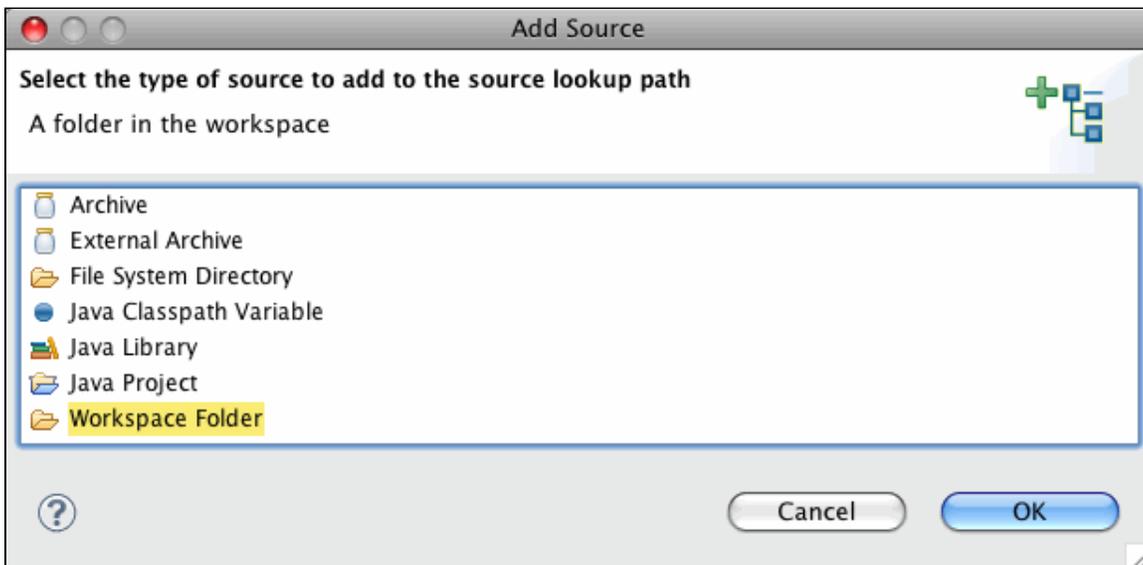
Debugging will even work if there is no class file available for the particular script being debugged. However, there is an extra step on how the debugger finds the source code for the script. When the debugger stops inside a script that does not have a corresponding class file, you will see this kind of editor:



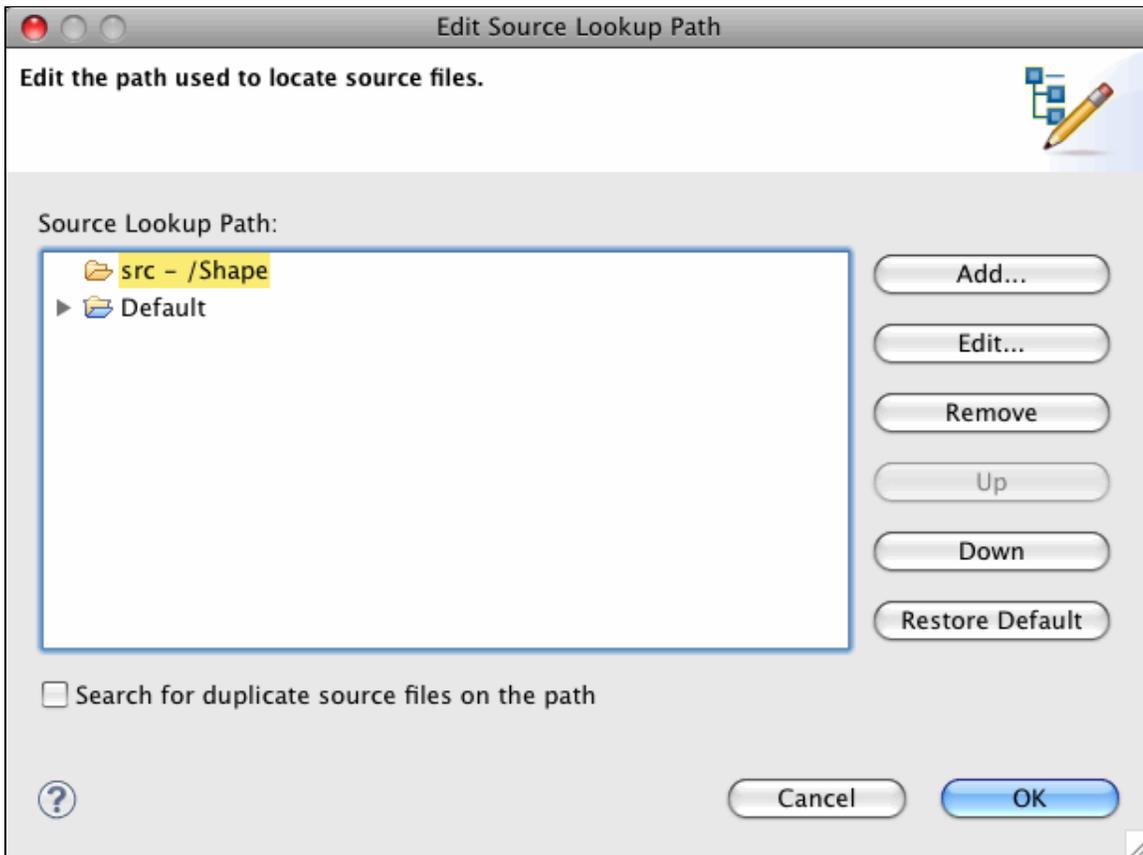
Click on the *Edit Source Lookup Path...* button and you will see the following dialog:



Click on *Add...* and the following dialog appears:



Choose *Workspace Folder* and select the source folder that contains your Groovy script. The end result should look something like this:



Now, the debugger knows how to find source code for scripts without class files. Please note that in future versions of Groovy-Eclipse, we will include functionality to perform this source lookup automatically.

#### ***Viewing variables***

Once a breakpoint is reached, (as in Java code) you can evaluate variables:

The image shows two parts of an IDE interface. The top part is the 'Variables' window, which displays a tree view of the current state of the program. The bottom part is a code editor showing Groovy code.

Name	Value
▶ this	ShapeScript (id=21)
▼ s	Shape (id=30)
▶ metaClass	MetaClassImpl (id=41)
▪ numberOfSides	7
▼ c	ShapeScript\$_run_closure1 (id=34)
▶ delegate	ShapeScript (id=21)
▪ directive	0
◆ maximumNumber	1
▪ metaClass	ClosureMetaClass (id=73)
▶ owner	ShapeScript (id=21)

groovy.lang.MetaClassImpl@de0034[class shapes.Shape]

```

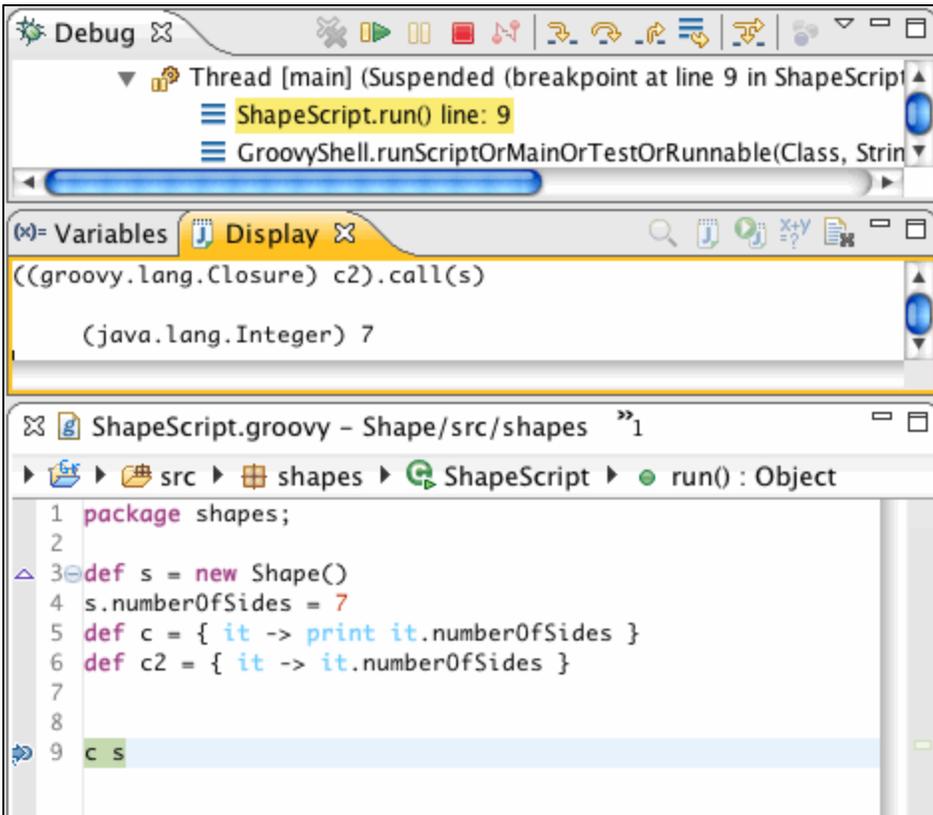
ShapeScript.groovy - Shape/src/shapes  ”1
▶ src ▶ shapes ▶ ShapeScript ▶ run() : Object
1 package shapes;
2
3 def s = new Shape()
4 s.numberOfSides = 7
5 def c = { it -> print it.numberOfSides }
6
7 c s

```

This lets you inspect the current state of the program and even look at the Groovy meta-classes and closures accessible in the current scope.

#### *Runtime evaluation of Groovy code*

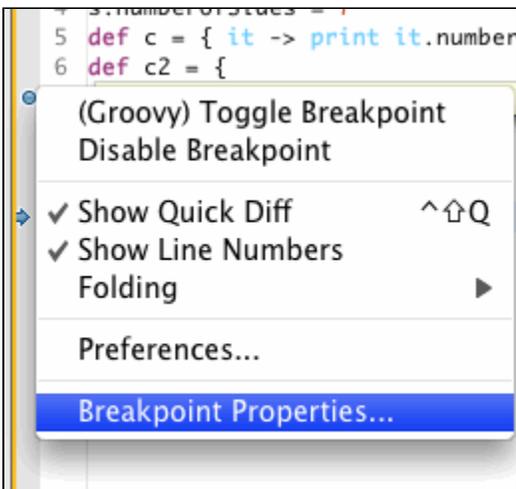
It is also possible to evaluate code snippets from the display view while in a Groovy scope:



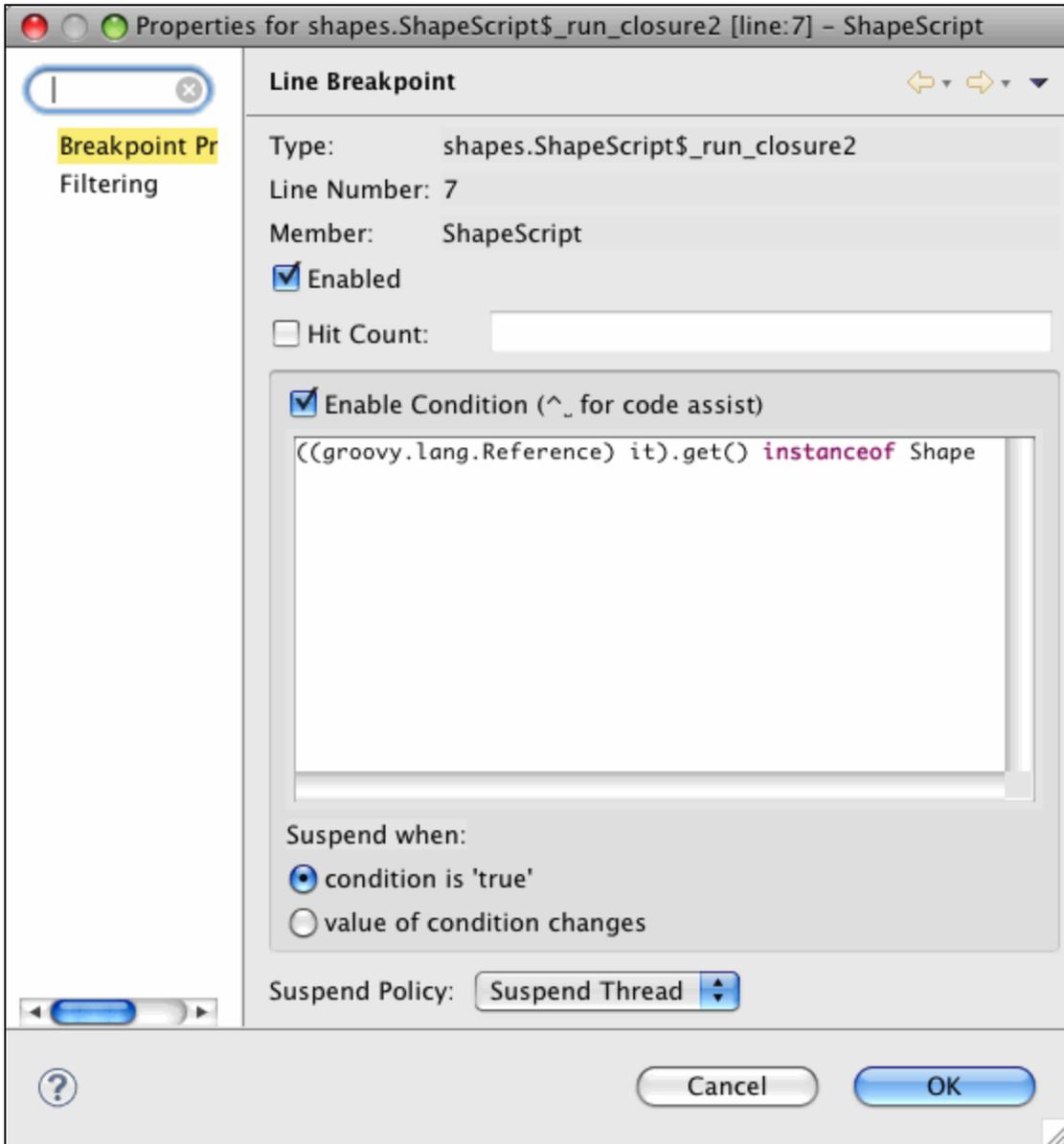
Java syntax is required for evaluating code snippets. What this means is that explicit casting of Groovy objects will be necessary for local variables and dynamically typed references.

#### Conditional breakpoints

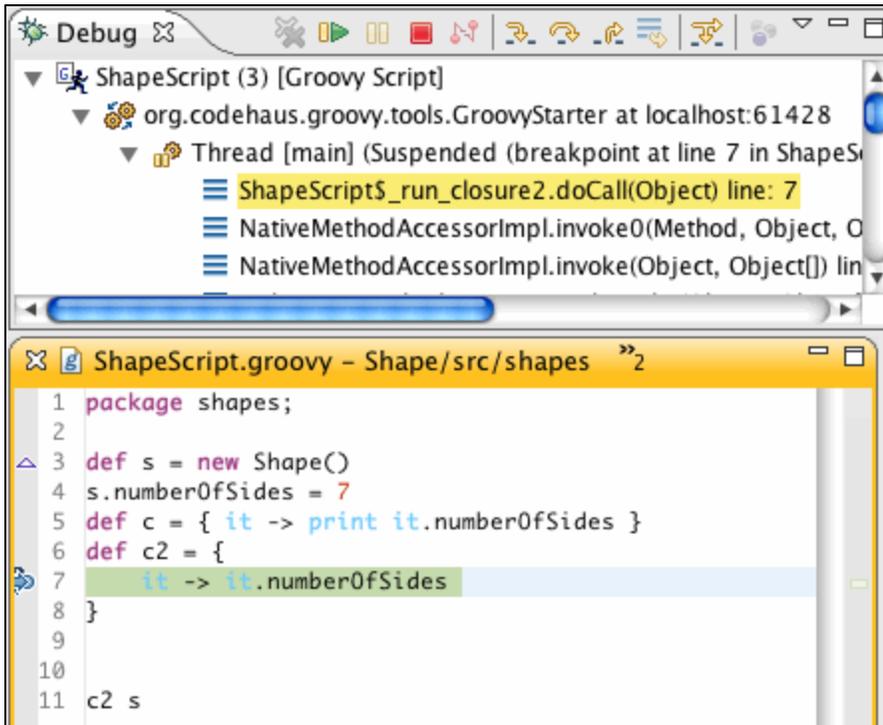
The Groovy debugging facility supports conditional breakpoints (Java syntax required). To create a conditional breakpoint, right click on the breakpoint definition and select *Breakpoint Properties...*:



The breakpoint properties dialog appears. Select *Enable Condition* and enter the condition to break at (again, only Java syntax is available, so it will be necessary to explicitly cast all variable references):



And now the debugger will stop at the defined location only when the condition is evaluated to true!



### *Refactoring/Formatting*

M1 contains several new refactoring and formatting facilities.

#### *Indentation & Formatting*

The Groovy editor will automatically indent poorly indented code on a paste. So, this:

```
def this_is(some) {
    very
    poorly
    indented
    "code"
}
```

Becomes this:

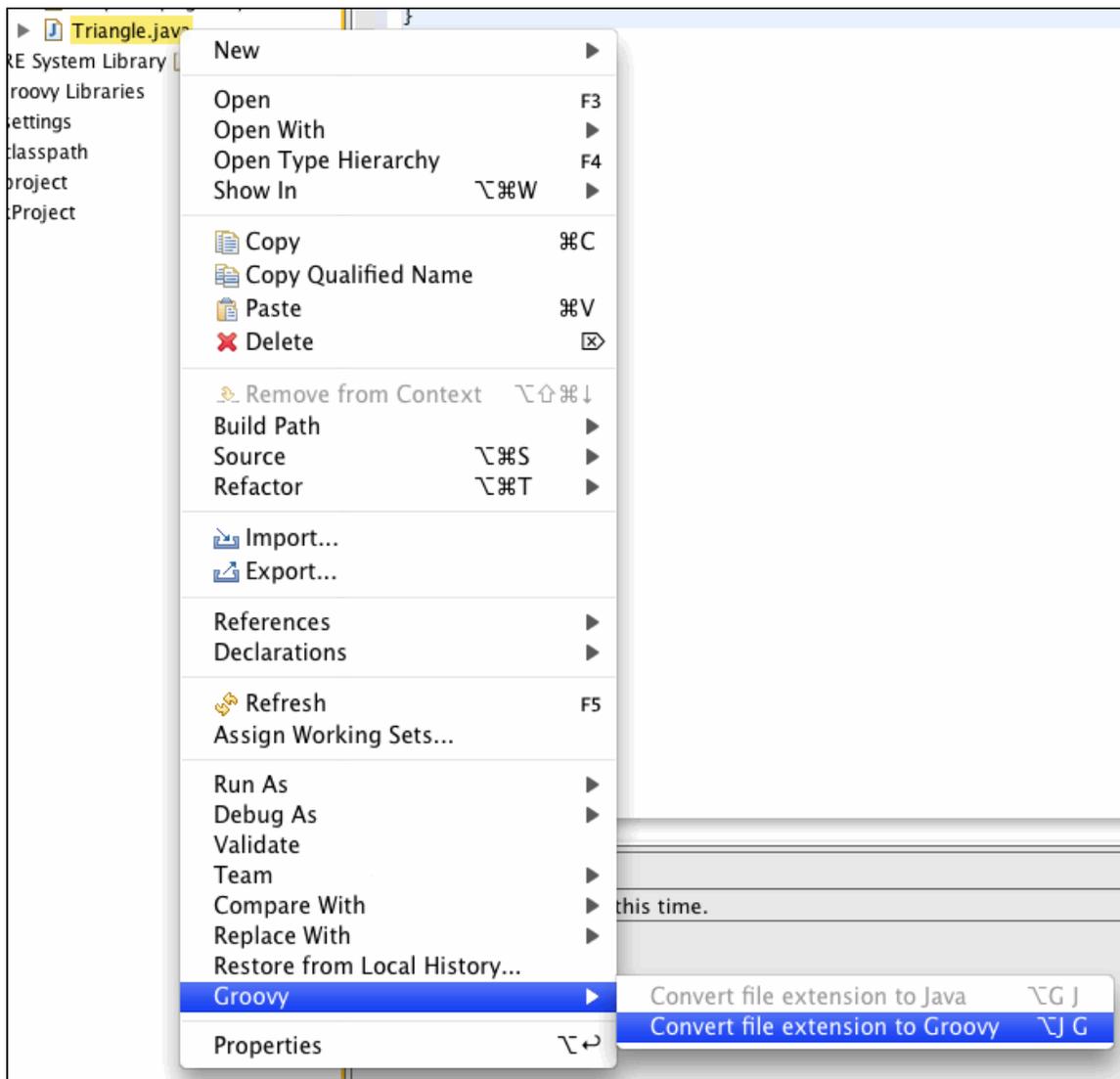
```
def x = 9
  print x

doSomething x

def this_is(some) {
  very
  poorly
  indented
  "code"
}
```

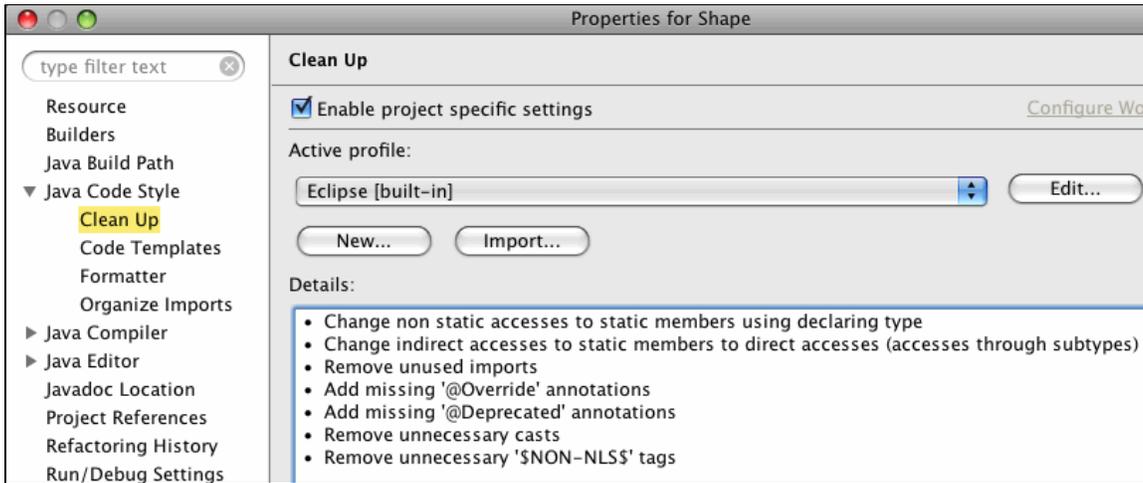
**Quick suffix switching**

It is simple to convert from a .java file to a .groovy file and back using the context menu:



**Organize imports**

The Groovy editor supports automatic organizing of imports (CTRL-Shift-O or CMD-Shift-O). Or you can set your project to automatically organize on save through the project properties page:



Note that only *Remove unused imports*, *format on save*, and *indent on save* are respected in Groovy editors. All other cleanup options are ignored.

### Issues addressed for M1

We have fixed over 130 issues for this milestone release:

Errors were reported by the JIRA trusted connection.

- BAD\_REMOTE\_IP; Request not allowed from IP address: {0}; ["63.246.24.165"]

### Issues addressed for M1 (0 issues)

Type	Key	Summary
------	-----	---------

### Groovy-Eclipse 2.0.0M2 New and Noteworthy

- [Overview](#)
- [Inferencing Engine](#)
- [Refactoring Support](#)
- [Task Tags](#)
- [Content Assist and Navigation in GStrings](#)
- [Extensibility for custom DSLs](#)
- [Issues fixed](#)
- [Thanks](#)

#### Overview

Today, the Groovy-Eclipse team has released milestone 2 of Groovy-Eclipse, tool support for the [Groovy](#) programming language built in [Eclipse](#). In this milestone release, we have focused on responsiveness and features in the user interface. This milestone is significantly more stable than milestone 1 as we have [fixed over 100 issues](#) in the bug tracker.

#### Groovy-Eclipse M2 update site

To install M2, add the following URL to your Eclipse update manager:

<http://dist.springsource.org/milestone/GRECLIPSE/e3.5/>

#### Inferencing Engine

The [inferencing engine](#) now forms the core of many of Groovy-Eclipse's most prominent features. The inferencing

engine is uses static analysis to infer the type of a single or a group of expressions of Groovy code. it is being used as the basis for content assist, navigation, search, and discovering unknown references. We have posted a [detailed description](#) of the inferencing engine.

#### Refactoring Support

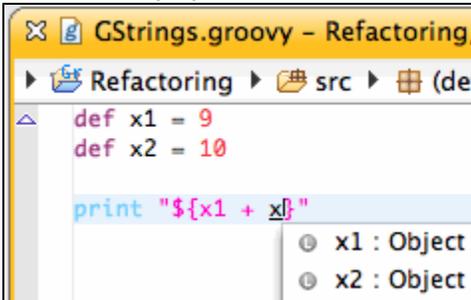
[Cross-language refactoring support](#) is now available in Groovy-Eclipse. Now, when renaming Java methods, fields, and classes, your Groovy code will be updated as well. And the reverse is true. You can find more information [here](#).

#### Task Tags

The Groovy compiler now parses task tags such as `TODO`, `FIXME`, and `XXX`. They are treated by Eclipse in the same way that Java task tags are. Task tags are configurable and can be viewed in the *Tasks* view. More detailed information can be found [here](#).

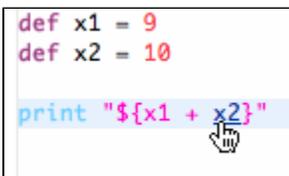
#### Content Assist and Navigation in GStrings

Content assist and navigation are now supported inside of GStrings. For example, in the following image, content assist will display both local variables:



Notice also that the `x` is underlined. This is because the `x` reference is undefined in the current scope. See the description of the [inferencing engine](#) for more information on this.

CTRL-Click (or CMD-Click on Mac) will follow a hyperlink to the definition of whatever is being hovered over (pressing F3 will have the same effect):



#### Extensibility for custom DSLs

Also included in M2 are extension points for custom DSLs to hook into Groovy-Eclipse's inferencing engine, content assist, and syntax highlighting. This is still basic work and will be more fully announced for RC1. This extensibility is particularly suited for implementing [Grails tool support in STS](#), but it is generic enough for any Groovy DSL to use.

#### Issues fixed

Errors were reported by the JIRA trusted connection.

- BAD\_REMOTE\_IP; Request not allowed from IP address: {0}; ["63.246.24.165"]

Issues addressed for M2 (0 issues)		
Type	Key	Summary

If the above doesn't show for you, go directly to the [jira issue tracker](#)

Thanks

Thank you to the community for diligently trying out new releases, raising bugs, suggesting fixes, and discussing features. If you want to participate in the discussion, join the [mailing list](#) and raise bugs in our [issue tracker](#).

## Groovy-Eclipse 2.0.0 New and Noteworthy

At the [start of May 2009](#) we announced we were working on a new approach to joint compilation for mixed Java/Groovy projects in Eclipse. We are pleased to now announce the final release of Groovy-Eclipse v2.0.0, based on that new technology. During the months of development we have rebased (and almost entirely rewritten) version 1 of the Groovy-Eclipse plugin, with the goal of offering first class Groovy support in the Eclipse IDE, comparable to the experience Java developers have in Eclipse. The driving themes for version 2.0.0 have always been to optimize around the common developer actions of editing, building, running and testing code:

- **Edit** We have an advanced inferencing engine and extensible content assist enabling us to provide highly accurate contextual aware editor suggestions. Task tags, javadoc hovers and even cross-language refactoring is also supported.
- **Build** As well as a 'stub-less' incremental joint compilation process, we use a modified groovy parser that is more recoverable than the standard parser - enabling us to be more stable in the face of partially correct code.
- **Run** Launching (and debugging) of applications or scripts is supported - even for code using transforms such as @Grab
- **Test** The plugin fully integrates with the JUnit support available in Eclipse.

More information on all these features can be found in the New and Noteworthy for the milestones and release candidates leading up to this release:

- [Groovy-Eclipse 2.0.0M1 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.0M2 New and Noteworthy](#)
- [Groovy-Eclipse 2.0.0RC1 New and Noteworthy](#)

### Groovy-Eclipse update site

To install 2.0.0 into Eclipse 3.5, you can update from the milestone update site you may already be using or define the following release URL to your Eclipse update manager:

<http://dist.springsource.org/release/GRECLIPSE/e3.5/>

What's next?

Some of the features we will be working on in the near term:

- Eclipse 3.6 support
- Better debugging experience
- Simpler user extensible content assist
- Picking up the latest Groovy 1.6/1.7 and later compiler versions

You can find more details on the [Groovy-Eclipse Roadmap](#), where you can comment on proposed features and see our release schedule.

We appreciate all the community support and feedback we have had through the version 2.0.0 development cycle. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you'd like to see), please raise them in our [issue tracker](#).

## Issues fixed since Groovy-Eclipse 1.5.7

Groovy-Eclipse 1.5.7 was the last public release and since then over 380 bugs have been fixed in the run-up to version 2:

Errors were reported by the JIRA trusted connection.

- BAD\_REMOTE\_IP; Request not allowed from IP address: {0}; ["63.246.24.165"]

<a href="#">Issues addressed for 2.0.0 since last release (0 issues)</a>										
Type	Key	Summary	Assignee	Reporter	Priority	Status	Resolution	Created	Updated	Due

If the above doesn't show for you, go directly to the [jira issue tracker](#).

## Eclipse Plugin Refactoring

### *Refactoring Support for Groovy-Eclipse Plugin*

As result of a bachelor thesis at the HSR University of Applied Sciences Rapperswil, Institute for Software, three students had implemented refactoring features for the Groovy-Eclipse plugin. The refactoring features are available in version 1.5.7 of the Groovy-Eclipse Plug-in. The most needed Refactorings are implemented yet. If you find bugs, or have ideas if improvement please [create an issue in Jira](#) for the component "Refactoring"

#### New Features

- **Refactorings**
  - Extract Method
  - Inline Method
  - Rename Local
  - Rename Method
  - Rename Class
  - Rename Field
- **Tools**
  - Source Code Formatter
  - Hierarchy View (experimental)

#### Download

The refactoring features are now available in the Groovy-Eclipse Plug-in version 1.5.7

More information about this project can be found on the project website of this project <http://groovy.ifs.hsr.ch>

#### *Demo*

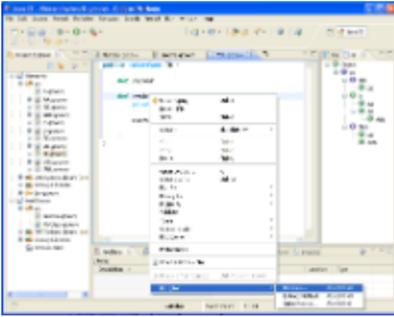
Demonstration of the key features are available as Flash Videos [Flash Videos Refactoring in Groovy-Eclipse](#)

The refactoring features ere presented at the OOPSLA 2008 in Nashville. Here is the [submission](#) for this demo.

#### *Documentation*

A simple documentation of the Refactoring features can be found in chapter 9.7 in the appendix of our [documentation](#).

#### Screenshot



## Create Your First Groovy Project

### TUTORIAL OVERVIEW

**Task** - Create a Groovy project in Eclipse.

**Level** - Basic. The task is simple if you have created a Java project in Eclipse.

**Prerequisites** - GroovyEclipse v2.0, Eclipse 3.4.2, 3.5, or 3.5.1.

### ALL GROOVY-ECLIPSE TUTORIALS

[Install Groovy-Eclipse Plugin](#)

Create Your First Groovy Project

### QUICK FACTS FOR EXPERIENCED ECLIPSE USERS

Create a Groovy project in Eclipse the same way you do a Java project, but use the Groovy wizards – **File > New > Groovy Project** and **File > New > Groovy Class** – instead of the Java equivalents.

**NOTE:** In this tutorial, the Groovy-Eclipse plugin is running on Eclipse 3.5.1. The user interface and task flow may vary somewhat in Eclipse 3.4.2.

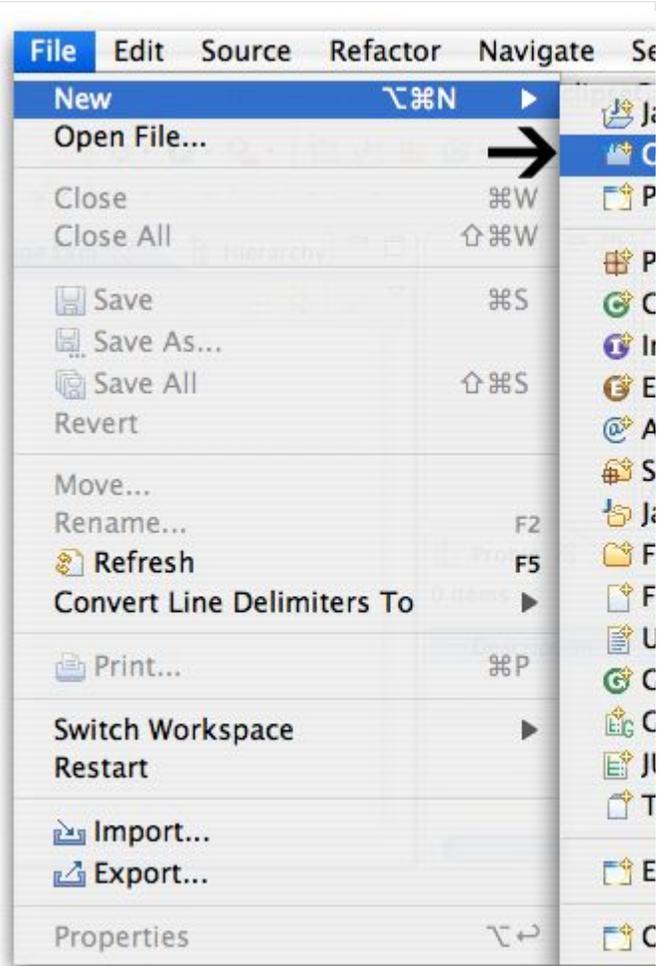
**GETTING HELP** – If you have problems, send a message to [http://xircles.codehaus.org/lists/eclipse-plugin-user@groovy.codehaus.org](mailto:http://xircles.codehaus.org/lists/eclipse-plugin-user@groovy.codehaus.org).

**Do**

**See**

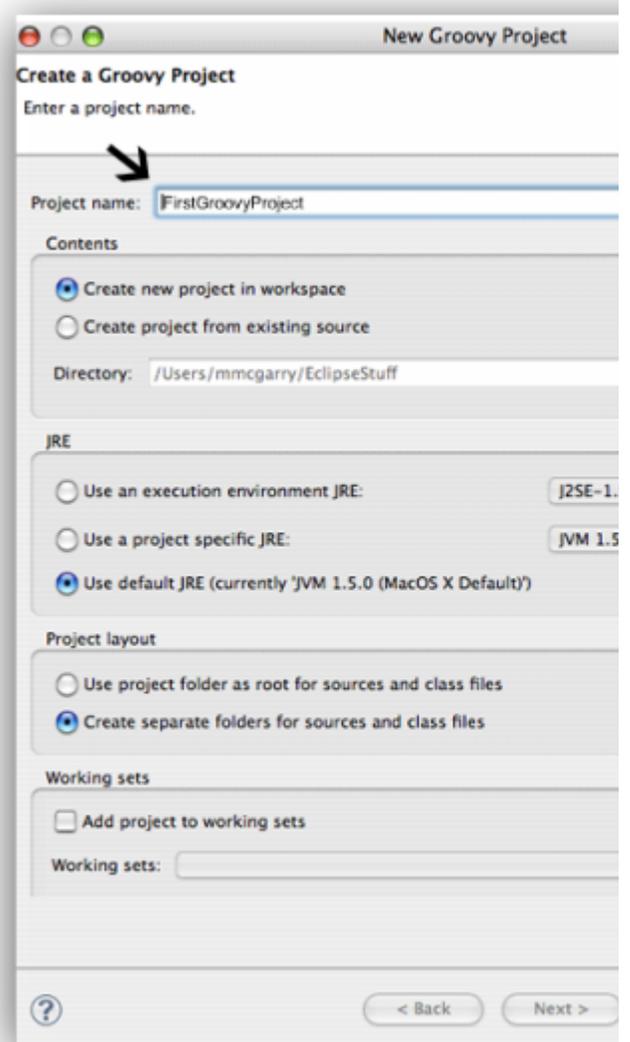
In Eclipse, click **File > New > Groovy Project**.

*If **Groovy Project** does not appear in the drop-down list, then select **Other** and search for **Groovy Project** in the dialog box.*



In the **New Groovy Project** wizard, type a name in the **Project Name** box, and click **Next**.

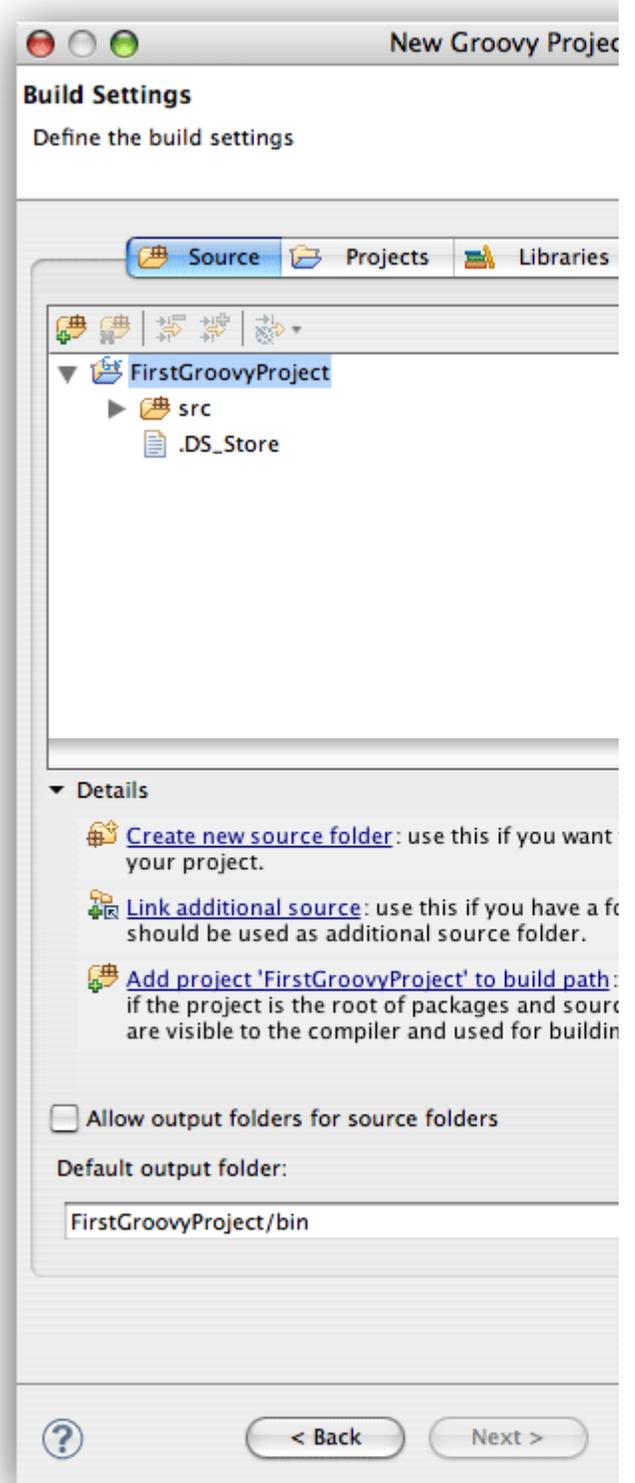
*GroovyEclipse creates the project and a folder structure for it. A folder with the same name as the project appears in the Package Explorer. The project directory contains a `src` and a `bin` folder, each empty.*



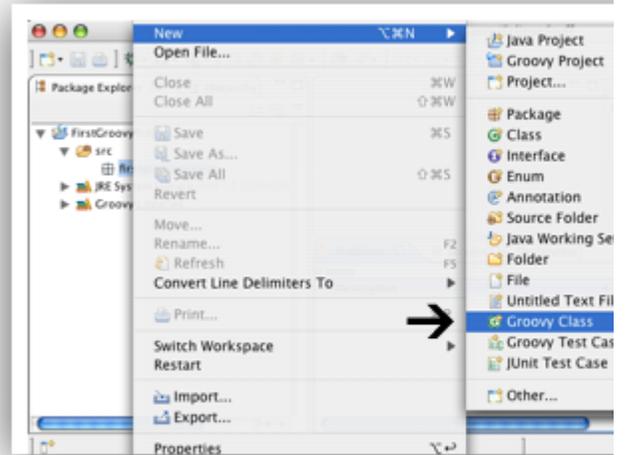
The tabs and options in the **Build Settings** window are identical to those in the the Java Development Tool (JDT).

Click **Finish** to proceed without customizing the build settings.

*For information about build options, see Eclipse help for the **New Java Project Wizard**; it applies to Groovy projects as well.*



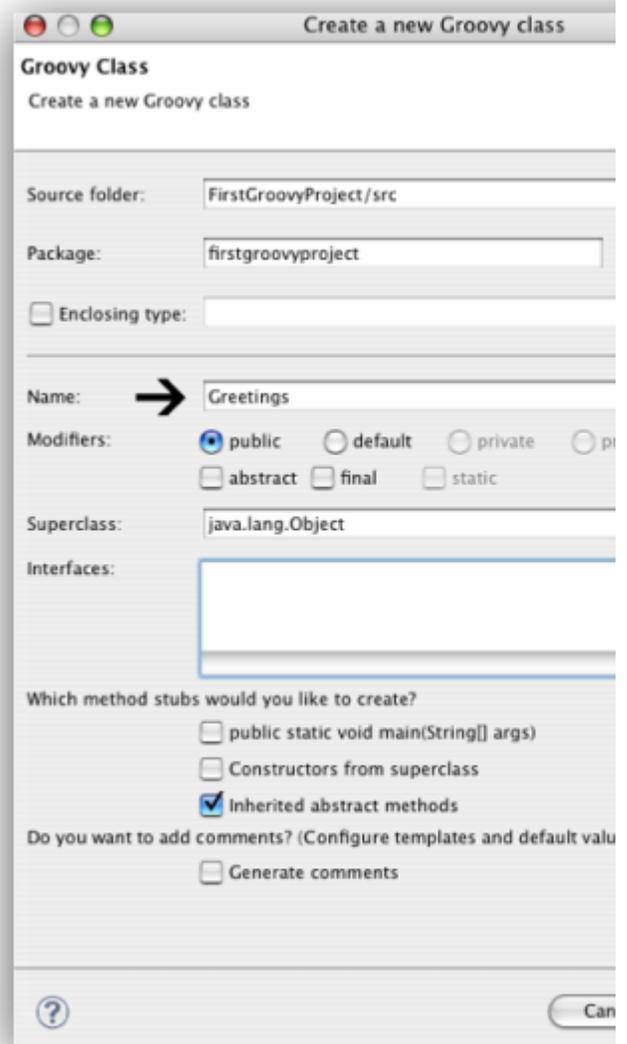
Select the project in the Package Explorer, and click **File > New > Groovy Class**.



In the **Groovy Class** wizard, type a name for the class in the **Name** box, a name for the package in the **Package** field, and click **Finish** to create the class and package.

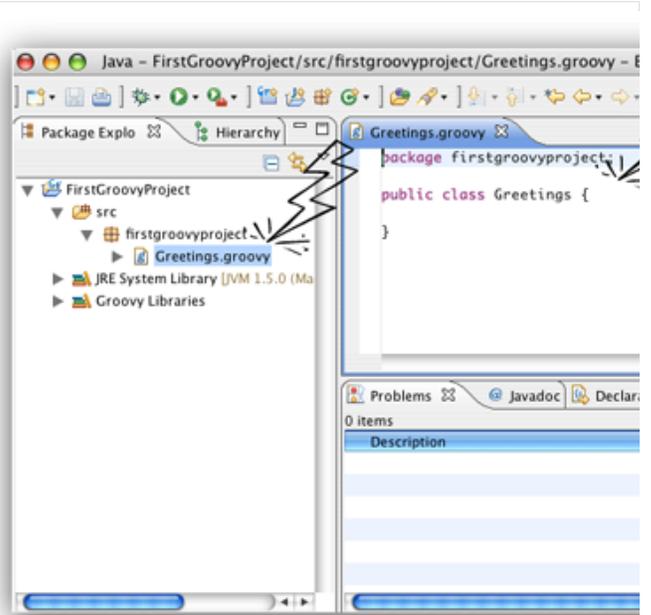
*If you type the name of an existing package in the **Package** field, the class is created there.*

*Just like in the JDT, you can create a package as a separate step with **File > New > Package**, or when you use the create the first class*



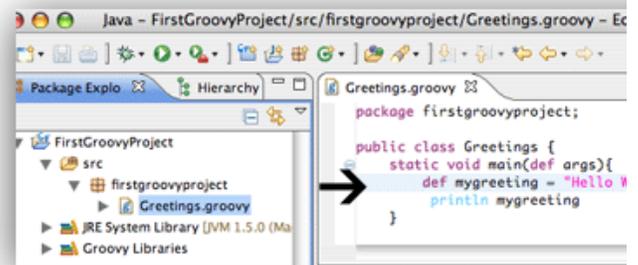
GroovyEclipse creates the new class. The new class, `Greetings.groovy` in this example, is listed in the Package Explorer, and opened in the editor.

*Note that, as generated by GroovyEclipse `Greetings.groovy` already contains the package statement and class declaration.*



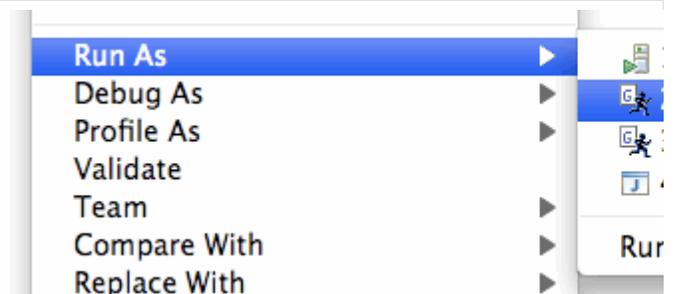
Paste this code into the file:

```
static void main(def args) {  
    def mygreeting = "Hello World"  
    println mygreeting  
}
```

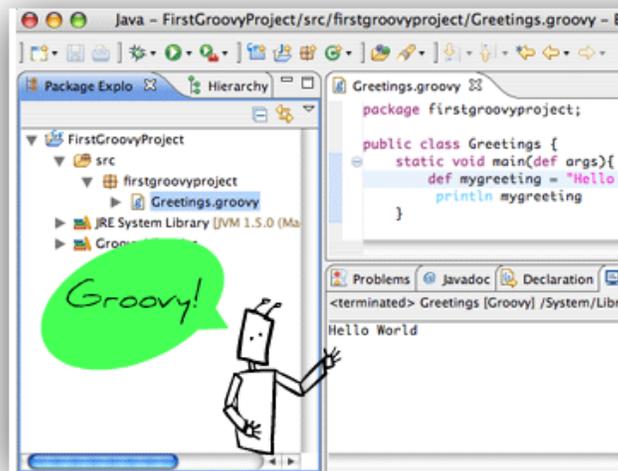


Right click anywhere in the editor and then **Run > Run As > Groovy Script** or **Java Application**.

*Running as an Application will launch the compiled \*.class files, whereas running as a Script will launch the uncompiled \*.groovy files. In general, the results will be the same, but there are some subtle differences between the two.*



The greeting is issued in the **Console** tab.



### Install Groovy-Eclipse Plugin

#### TUTORIAL OVERVIEW

**Task** - Install GroovyEclipse Plugin.

**Level** - Very basic.

**Prerequisites** - Eclipse 3.4.2, 3.5, 3.5.1 or 3.6.0.

#### ALL GROOVY-ECLIPSE TUTORIALS

Install GroovyEclipse Plugin

[Create Your First Groovy Project](#)

### QUICK FACTS FOR ECLIPSE EXPERTS

GroovyEclipse update site for Eclipse 4.2 (Juno),  
Release update site

<http://dist.springsource.org/release/GRECLIPSE/e4.2/>

GroovyEclipse update site for Eclipse 3.7 (Indigo),  
Release update site

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

GroovyEclipse update site for Eclipse 3.6.n (Helios),  
snapshot build with recent fixes

<http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.6/>

GroovyEclipse update site for Eclipse 3.5.n (Galileo), M1 build	<div data-bbox="865 138 1442 302" style="border: 1px solid black; padding: 5px;"> <a href="http://dist.springsource.org/milestone/GRECLIPSE/e3.5/">http://dist.springsource.org/milestone/GRECLIPSE/e3.5/</a> </div>
GroovyEclipse update site for Eclipse 3.5.n (Galileo), snapshot build with recent fixes	<div data-bbox="865 342 1442 541" style="border: 1px solid black; padding: 5px;"> <a href="http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.5/">http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.5/</a> </div>
GroovyEclipse update site for Eclipse 3.4.2 (Europa), snapshot build with recent fixes	<div data-bbox="865 583 1442 747" style="border: 1px solid black; padding: 5px;"> <a href="http://ci.repository.codehaus.org/greclipse/snapshot/e34/">http://ci.repository.codehaus.org/greclipse/snapshot/e34/</a> </div>
GroovyEclipse update site for Eclipse 3.4.2 (Europa), M1 build	<div data-bbox="865 793 1442 921" style="border: 1px solid black; padding: 5px;"> <p>available soon</p> </div>

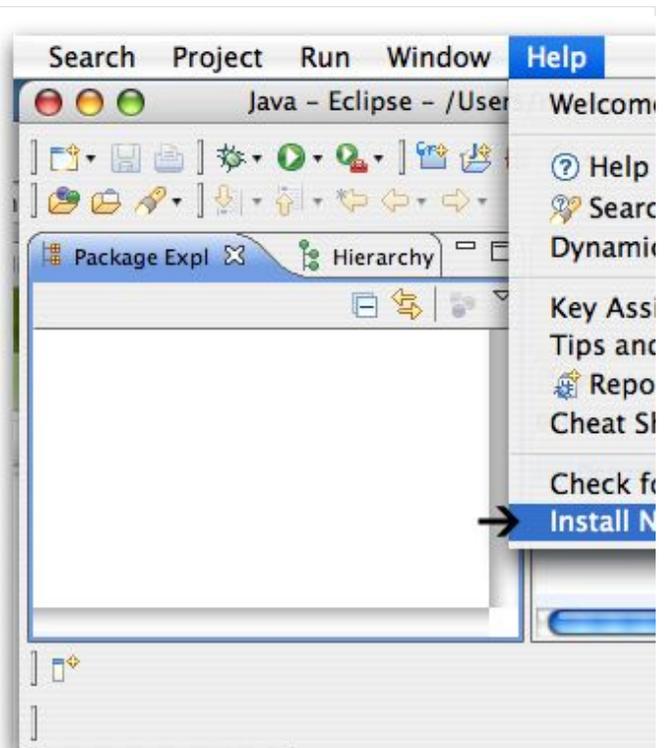
**GETTING HELP** – If you have problems, send a message to [http://ircles.codehaus.org/lists/eclipse-plugin-user@groovy.codehaus.org](mailto:ircles.codehaus.org/lists/eclipse-plugin-user@groovy.codehaus.org).

<b>Do</b>	<b>See</b>
-----------	------------

In Eclipse 3.5.x, click **Help > Install New Software**.

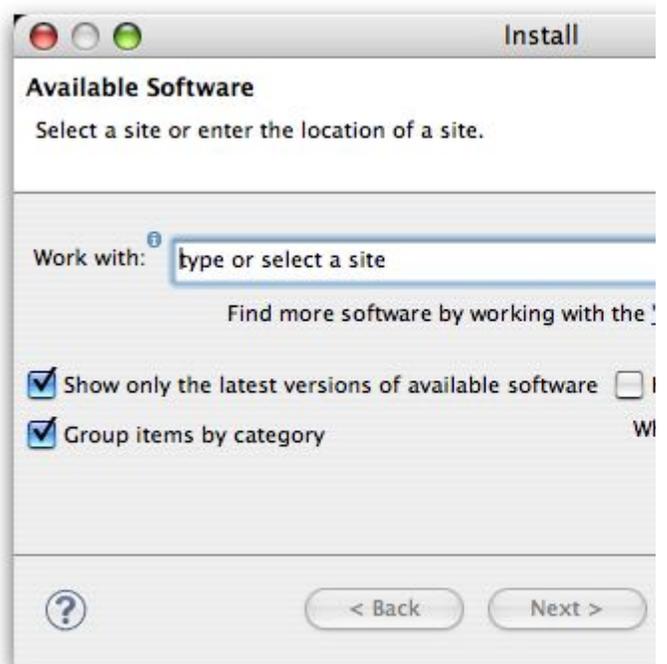
*You use Eclipse's standard installation process to download and install GroovyEclipse. The Eclipse user interface for software installation and update varies between 3.4 and 3.5.*

*Eclipse 3.4.2 users: click **Help > Software Updates** and follow the dialogs to work with or add the GroovyEclipse update site - see step 3 for the URL.*



To add the GroovyEclipse plugin download site to your **Available Software Sites**, click **Add** on the **Available Software** page.

*If you have previously added the site, click the blue down arrow and select it from the drop-down list of your available sites.*

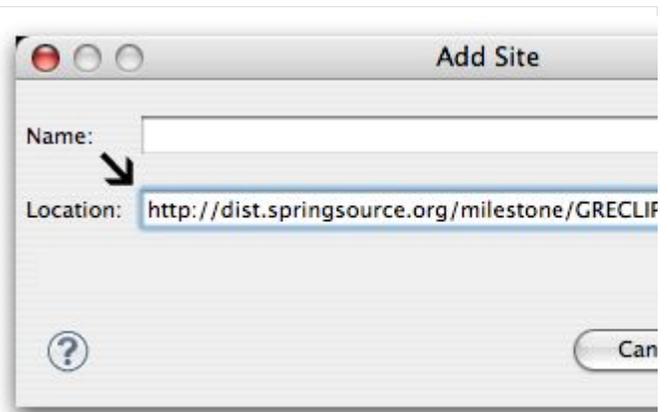


On the **Add Site** page, type a name in the **Name** box to assign a name of your preference to the site, or leave it blank to use a default site name constructed from repository metadata.

In the **Location** box type the download URL the URL that corresponds to your version of Eclipse, and click **Next**.

*If you use 3.4.2 -  
"http://ci.repository.codehaus.org/greclipse/snapshot/e34" or*

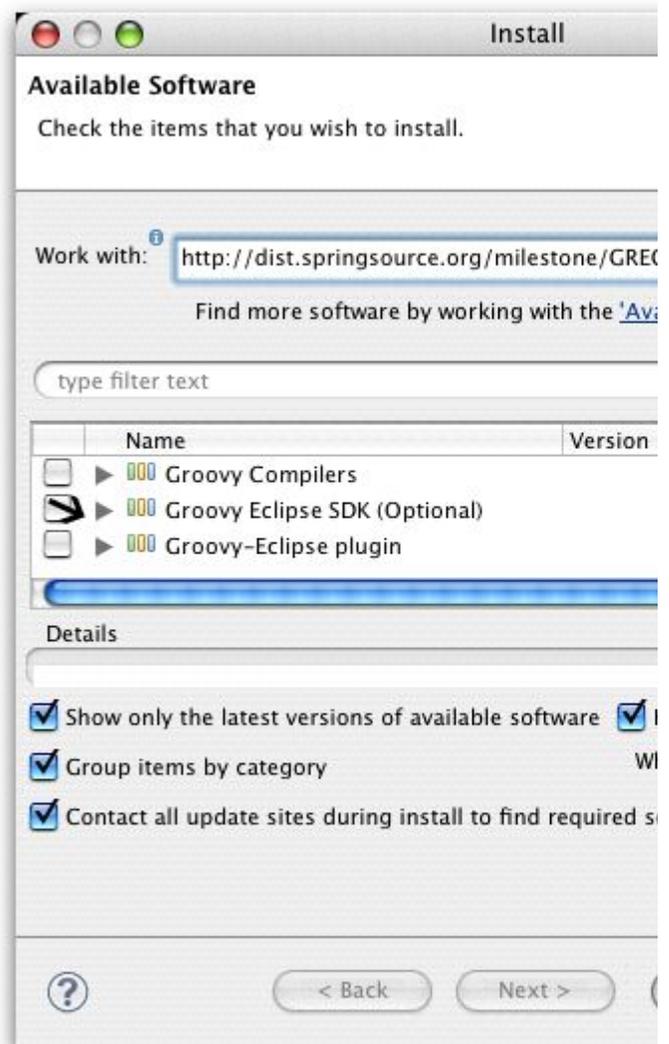
*If you use 3.5.x -  
"http://dist.springsource.org/milestone/GRECLIPSE/e3.5/"*



On the **Available Software** page, checkmark **Groovy-Eclipse Plugin**, and if you wish to work with the plugin sources, **Groovy Eclipse SDK**.

Click **Next**.

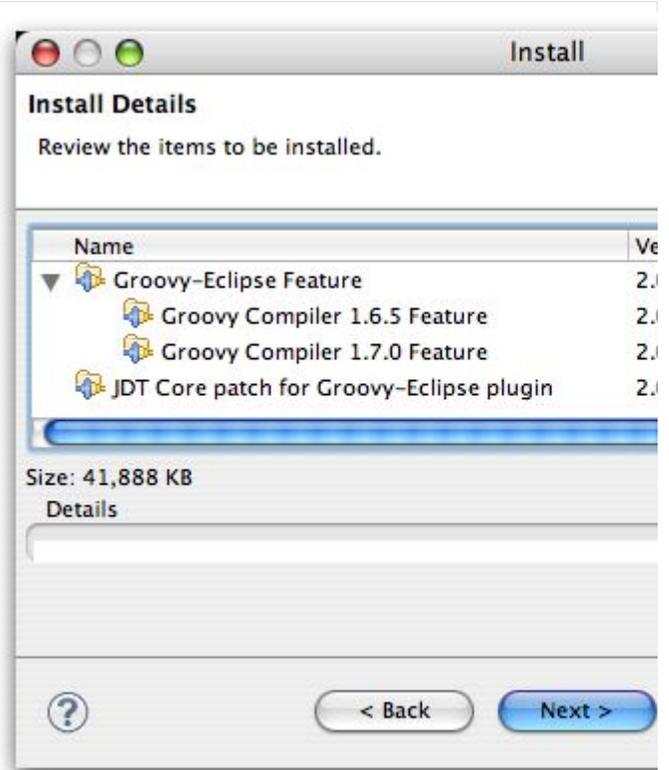
*Don't checkmark **Groovy Compilers** - they are included in the GroovyEclipse Plugin.*



You can review the components to be installed on the **Install Details** page.

*As noted above, the GroovyEclipse Plugin includes two versions of the Groovy compilers: v1.7-beta2 (enabled) and v1.6.5.*

*The JDT Core patch is an update to the Eclipse Java Development Tool (JDT) that enables Groovy/JDT integration.*



Accept the license agreement and click **Finish**.



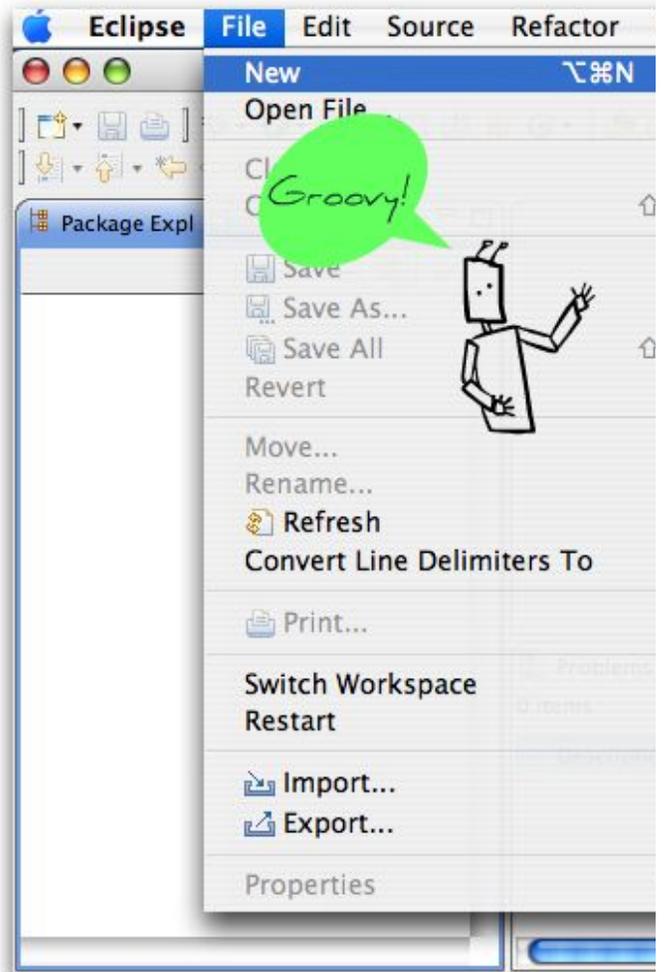
Eclipse prompts you to restart.



After Eclipse restarts, you can see that Eclipse's **File > New** menu now includes wizards for creating Groovy projects, classes, and test cases.

You have successfully installed GroovyEclipse.

For step-by-step instructions on how to create your first Groovy program using GroovyEclipse, see [Create Your First Groovy Project](#).



### Groovy-Eclipse 2.0.1 New and Noteworthy

The Groovy-Eclipse team is pleased to release 2.0.1. Groovy-Eclipse 2.0.1 is primarily a service refresh of [2.0.0](#). We have fixed over [40 issues](#) in this release.

### **i** Groovy-Eclipse update site

To install 2.0.1 into Eclipse 3.5, you can update from the milestone update site you may already be using or define the following release URL to your Eclipse update manager:

<http://dist.springsource.org/release/GRECLIPSE/e3.5/>

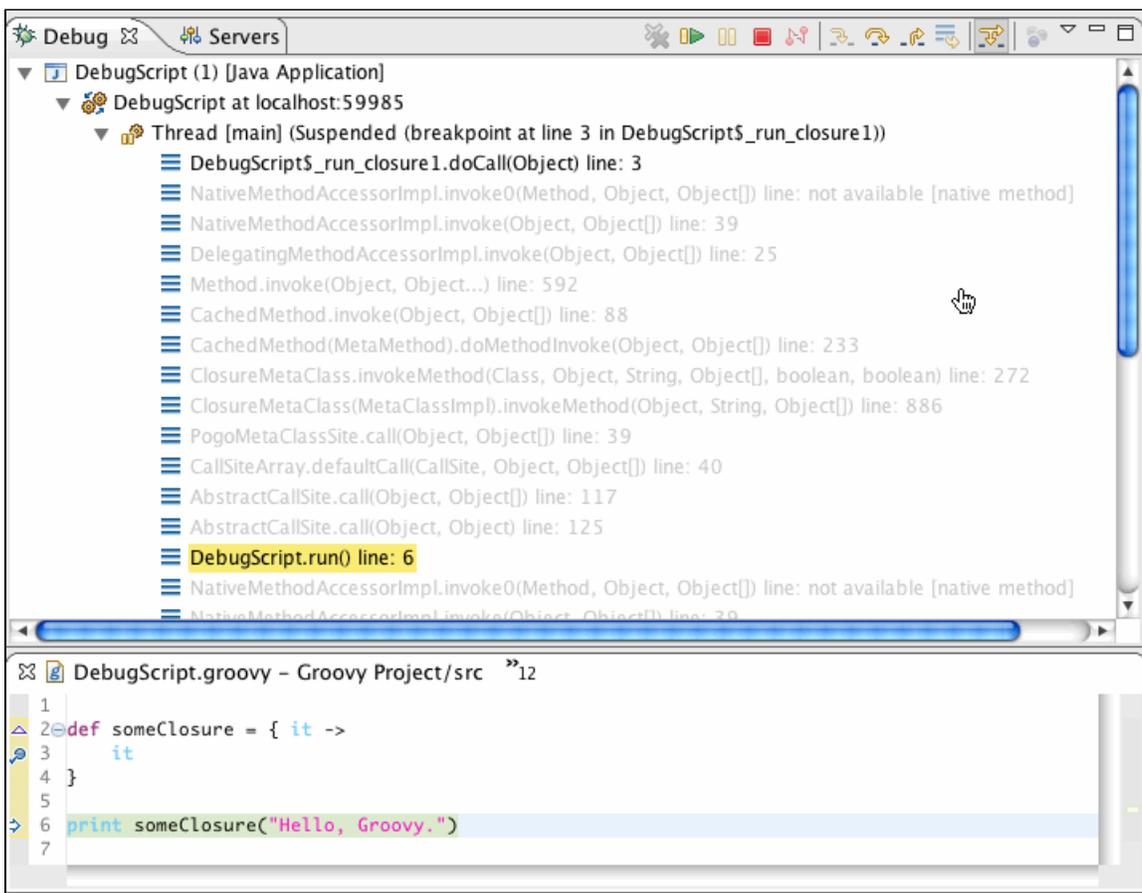
## Support for Eclipse 3.5.2

This release of Groovy-Eclipse can be installed into Eclipse 3.5.2.

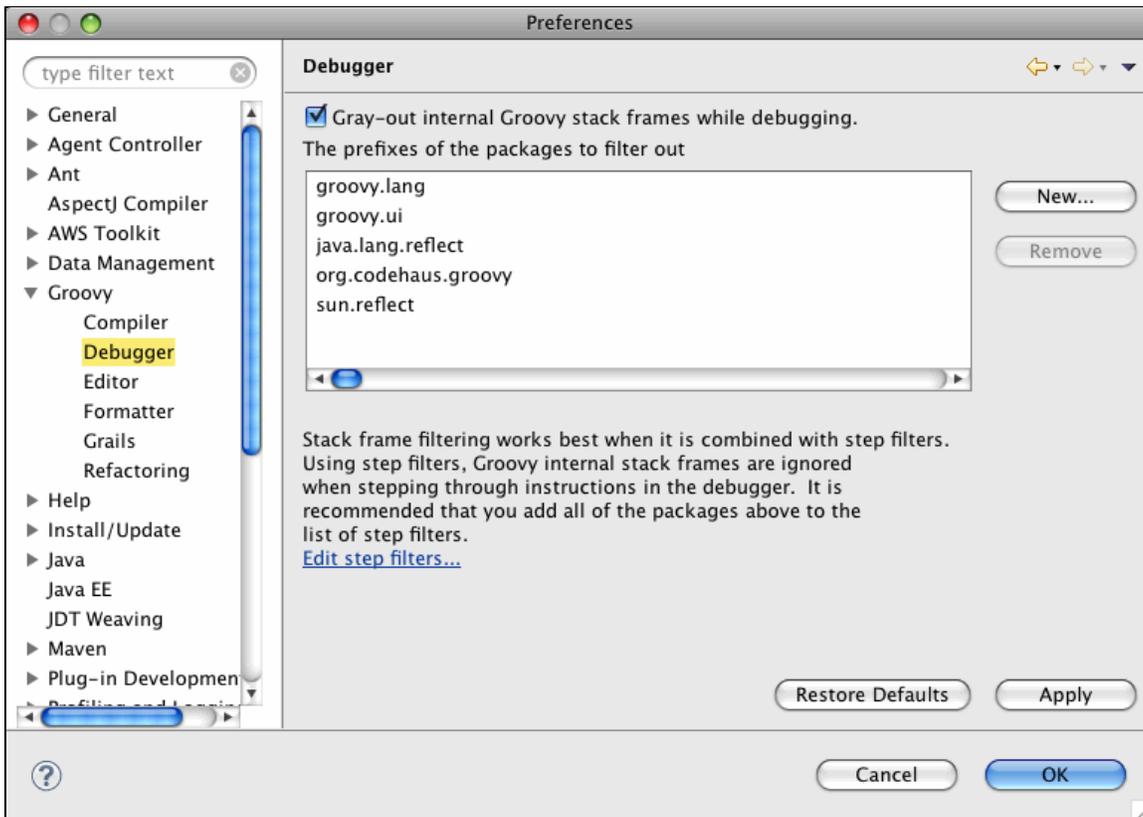
There is one significant new feature that should be described for this release.

## Filter Groovy internal stack frames

Under the covers, Groovy uses reflection to achieve much of its dynamicism. This can be problematic when using the Eclipse debugger since all of the internal stack frames are exposed to the user. There is a new option in Groovy-Eclipse to de-emphasize these stack frames and highlight show user stack frames:



Notice how the user stack frames are in black. This option can be enabled/disabled and tweaked from the Groovy -> Debugger preferences page:



#### What's next?

You can find more details on the [Groovy-Eclipse Roadmap](#), where you can comment on proposed features and see our release schedule.

We appreciate all the community support and feedback we have had through the version 2.0.1 development cycle. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you'd like to see), please raise them in our [issue tracker](#).

#### Groovy-Eclipse coding conventions

This page contains some coding conventions for the Groovy-Eclipse project. This is still a work in progress. If you feel there are some conventions that are missing from here, please raise a jira issue for it.

We are only trying to coordinate some high level conventions that maintain the quality of the project across releases and when there are multiple developers. For anything that is not specified here, use your common sense. If you think that there is a major issue of concern, send a message to the [eclipse-plugin-dev@groovy.codehaus.org](mailto:eclipse-plugin-dev@groovy.codehaus.org) mailing list.

#### Copying code from Eclipse and JDT

In general, we want to avoid copying code from JDT and Eclipse. Instead, the preferred way to extend or borrow Eclipse functionality is to use sub-classing and reflection. Where this approach is not possible or it is unwieldy, it may be reasonable to copy an entire file. Here are the guidelines for doing this:

- Add the copied file to the `jdt_patch` source folder in the plugin. (If this doesn't exist yet, then create it.)
- Change the package name by prefixing 'greclipse.' to the package. Do not change the type name.
- First line of the file should describe where it comes from (fully qualified name, plugin, etc).
- If there are visibility problems that mean you cannot change the package name, then leave the package as is and suffix the type name with `PATCH`. This should be a last resort.

- Surround all changes (apart from the package statement and imports) with "// GROOVY start" and "// GROOVY end"
- If suitable, add a textual description of why you are doing what you are doing after "// GROOVY start"
- If you have been **really keen** include the bugzilla entry raised against JDT related to the change you are making "prNNNNNN"
- Where you are replacing existing code with new code, use this format:

```
// GROOVY start: ...reason for the change..
pr131072
/* {
...old code...
*/
...new code...
// GROOVY end
```

This ensures comparisons and merges are easier because the old code will precisely match what is in the original file. If the old code includes `/* */` comments just apply common sense and think what will be easiest to merge again later.

In some cases, you may want to copy code from JDT as a starting point, but you have no intention of keeping the versions synchronized across Eclipse releases. This situation is fine, but you must keep the original EPL license at the top of the file.

#### **Code formatting, templates, and code cleanups**

The Groovy-Eclipse plugins use a set of pre-defined code formatting, templates, and code cleanups. They are set in each project. Also, copies of these preferences are included in the `org.codehaus.groovy.eclipse.core` project. When creating a new project, be sure to use these preferences.

#### **Task tags in code**

Do not use TODO tags. Instead use something like FIXADE (where instead of ADE, use your initials). And after the FIXADE tag, you can optionally include a version number that this change should be applied to (eg- `// FIXADE (2.1.0)` ). You may also want to add the tag as one that is recognized by the JDT compiler. Go to: Preferences -> Java Compiler -> Task Tags and add what you need.

#### **Error handling**

In general, unless there is a good reason not to, errors should be logged to the error log. To do this, in the catch clause, add the following code:

```
GroovyCore.logException(String, Throwable)
```

And when in the `org.eclipse.jdt.groovy.core` project, use this instead (or one of the variants of the method):

```
org.eclipse.jdt.internal.core.util.Util.log(Throwable)
```

#### **Jira**

The version field of an issue should be the version of Groovy-Eclipse that the issue was raised against. If unknown, choose the most recently released version. If raised against the latest dev build, also choose the most recently released version.

Only assign an issue to yourself if you intend to work on it in the near future.

Mark the issue as "In Progress" once you are starting to write code.

### ***(Deprecated) Getting Started With Groovy-Eclipse Source Code on Ubuntu***

This page is now deprecated. Groovy-Eclipse has moved to git. See [Getting Started With Groovy-Eclipse Source Code](#) instead.

- 1 [Step 1: Install Eclipse](#)
- 2 [Step 2: Install Subversion on your machine](#)
- 3 [Step 3: Install Eclipse support for Subversion](#)
- 3.1 [Step 3.a: Subversive Install](#)
- 3.2 [Step 3.b: Subclipse Install](#)
- 3.3 [Step 3.c: Eclipse Java Library Path](#)
- 4 [Step 4: Install Groovy-Eclipse into your Eclipse](#)
- 5 [Step 5: Checkout Source from SVN](#)
- 6 [Step 6: Run the tests](#)

#### **Building Groovy Eclipse in Eclipse (on Ubuntu 10.04)**

Below is a step-by-step list of instructions that takes you through the process of setting up an Eclipse environment for building the Groovy-Eclipse plugin.

These instructions were written while going through the process on an Ubuntu 10.04 machine. So in places they are specific to that environment.

If someone tries these instructions in a different environment (say Mac OS X or Windows) it would be nice to provide a copy of this page with appropriate modifications for those environments.

#### ***Step 1: Install Eclipse***

Install one of

- Eclipse for RCP/Plug-in Developers (183 MB)
- Ubuntu note: Don't install Eclipse from the Ubuntu repositories download it manually from [eclipse.org](http://eclipse.org) instead.
- [Eclipse](#) or [STS](#) pick and install the package appropriate for your OS.

I prefer to install Eclipse RCP instead of STS. The main reason is that the RCP plugin environment comes with all the source code for JDT plugins, which is quite useful while working on Groovy-Eclipse, which heavily subclasses from JDT.

#### ***Step 2: Install Subversion on your machine***

We'll need to checkout the Groovy sources from SVN. There are two options available for installing Eclipse support for subversion.

- Subversive from the [Eclipse.org](http://eclipse.org) repositories (recommended)
- Subclipse from <http://subclipse.tigris.org/>

**Note:** For Subversive, it is possible to skip this step (OS install of subversion) entirely as there is a "Full Java" version of the backend libraries. I recommend this option since it is the easiest, most painless install process. If you choose this path, skip to Step 3.

### In Ubuntu (10.04):

Open Synaptic and search for subversion.

I found and installed both of the following:

- subversion 1.6.6
- libsvn-java 1.6.6

#### *Step 3: Install Eclipse support for Subversion*

If you have something already installed, you may just want to go with that. Otherwise choose either Subversive or Subclipse and follow instructions below.

## Step 3.a: Subversive Install

To install subversive, you go through the Galileo update site (under "Help >> Install New Software"). You don't need to add it, it should already be there (with a standard install of Eclipse or STS). Under the Collaboration category find and install:

- Subversive SVN Team Provider
- Subversive SVN Integration for the Mylin project (optional)

After installing these and restarting Eclipse, you will be prompted to install an SVN connector package. Pick a version that matches your install of subversion (e.g. I installed the JavaHL 1.6 connector to match the version installed in my OS). If you chose to go with the "pure java" option rather than native libraries, pick the most recent "full java" version (1.1.7 at the time of this writing, for Eclipse 3.6).

## Step 3.b: Subclipse Install

As an alternative to Subversive you can install Subclipse instead. (Note: It is probably a bad idea to install both of them at the same time)

Before installing Subclipse, check which version you need to install. Essentially it depends on which version of subversion and the JavaHL library you have installed in your OS. It is recommended to get the more recent version of JavaHL and SVN to be able to install the more recent version of subclipse plugin.

With version 1.6.6 of the JavaHL (libsvn-java) we need to install the [corresponding Subclipse](#) version: 1.6.x.

This can be installed from the Subclipse 1.6 update site:

[http://subclipse.tigris.org/update\\_1.6.x](http://subclipse.tigris.org/update_1.6.x)

## Step 3.c: Eclipse Java Library Path

If you are on Linux, and you did not choose the "pure Java" install, you may still need to tell Eclipse (or STS) about the installed JavaHL libraries. **If you chose a pure java SVN install you can skip this step.**

### Ubuntu 10.04 with Eclipse RCP

Add the library path into the **eclipse.ini** file in the eclipse installation directory. In my case, I manually installed

Eclipse into my home directory so I added

```
-Djava.library.path=/usr/lib/jni
```

to the end of `~/Applications/eclipse/eclipse.ini`

Make sure this line is placed somewhere after the `-vmargs` line.

### **Ubuntu (10.04) with STS (2.3.2)**

Add the library path into the **STS.ini** file in the STS installation directory. In my case, I manually installed Eclipse into my home directory so I added

```
-Djava.library.path=/usr/lib/jni
```

to the end of `~/Applications/springsource/sts-2.3.2.RELEASE/STS.ini`

Make sure this line is placed somewhere after the `-vmargs` line.

You may want to check whether the JavaHL library is properly installed: in Eclipse Preferences >> Team >> SVN >> SVN Connector it should display the version of the library if it is properly installed.

Note: The "win32 only" comment in the version info seems untrue, things work just fine under Linux, so just ignore it.

#### *Step 4: Install Groovy-Eclipse into your Eclipse*

Some of the projects contain Groovy code, so we need to have Groovy support installed.

**Preferably, the latest version of Groovy Eclipse should be installed, which can be obtained**

**from the update site for the latest snapshot:**

<http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.6/>

**Alternately, the following update site can be used to get the latest release, but this might be a bit outdated:**

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

In [STS](#) it can also be installed via the Dashboard from the Extensions Tab.

#### *Step 5: Checkout Source from SVN*

In SVN repository explorer: Add SVN repository location: <https://svn.codehaus.org/groovy/eclipse>

- trunk
  - base (contains plugins for compiler integration)
    - org.codehaus.groovy
    - ...
  - base-test (contains plugins that test the compiler integration)
    - org.eclipse.jdt.groovy.core.tests.builder
    - ...
  - extras (contains Groovy-Eclipse m2eclipse integration and the groovy-eclipse-compiler plugin for maven as well as some test projects.)
    - ...
  - ide (contains ui plugins)
    - ...

- ide-test (contains ui test plugins)
  - ...
- jdt-patch (contains the JDT core patch plugins...only check out the branch corresponding to the eclipse version you have as your target environment)
  - e36 (JDT patch and related plugins targeting Eclipse 3.6)
    - ...
  - e37 (JDT patch and related plugins targeting Eclipse 3.7)
    - ...

Each of the directories base, base-test, extras and ide contains a number of eclipse projects. Check out all the projects except those in extras. **You must select only the projects (e.g. org.codehaus.groovy), but not the directories base, base-test, ... themselves.**

This can be done easily as follows: You can select multiple projects by control-clicking on each project. When all projects are selected do a "check out as project" (subclipse) or "check out" (subversive) to check them out all at once.

*Step 6: Run the tests*

Do **Run As->JUnit plugin test** on **org.codehaus.groovy.alltests**.

**Note 1:** There are two separate sets of tests in alltests plugin:

- AllGroovyTests.java
- GroovyJDTTests.java

At the time when I wrote this, \*running both test suites at once resulted in a number of test failures\* in the GroovyJDTTests suite. Running both the suites separately I got no such failures.

**Note:** If you get an Out of Memory Error like this (I got this in STS)

```

java.lang.OutOfMemoryError: PermGen space
at java.lang.ClassLoader.defineClass1(Native
Method)
at
java.lang.ClassLoader.defineClassCond(ClassL
oader.java:632)
at
java.lang.ClassLoader.defineClass(ClassLoade
r.java:616)
...

```

You can increase the size of PermGen space in "Run As -> Run Configurations"

Add this to VMArguments:

**-XX:MaxPermSize=256m**

### **Getting Started With Groovy-Eclipse Source Code**

- 1 [Download and Install Eclipse](#)
- 2 [Install git](#)
- 3 [Install eGit \(Optional\)](#)
- 4 [Install Groovy-Eclipse](#)
- 5 [Import Groovy-Eclipse source code into your workspace \(the simple way\)](#)
- 6 [Import Groovy-Eclipse source code into your workspace \(the complicated way\)](#)
- 7 [Run the tests](#)
- 8 [Now, look for some interesting bugs to work on](#)

### **Building Groovy Eclipse in Eclipse**

Below is a step-by-step list of instructions that takes you through the process of setting up an Eclipse environment for building the Groovy-Eclipse plugin. These instructions were written while going through the process on a Mac OS Lion machine. Except for choosing an Eclipse and installing git, the process is similar no matter which OS and architecture you are using

These instructions are meant for Eclipse 3.7 (Indigo) or Eclipse 4.2 (Juno).

#### **Download and Install Eclipse**

Download and unzip Eclipse for RCP/Plug-in Developers from <http://eclipse.org/downloads>. Choose the appropriate version for your architecture.

*Ubuntu note:* Don't install Eclipse from the Ubuntu repositories download it manually from [eclipse.org](http://eclipse.org) instead.

#### **Install git**

Groovy-Eclipse is available from github. To work with github, you need to have git installed on your machine. [Github b has some good documentation](#) for a gentle introduction into git and it explains how to install the command line tools.

#### **Install eGit (Optional)**

[eGit](#) is the Eclipse tool support for working with git source code repositories. This tooling is helpful for running git operations inside of Eclipse. Things like viewing history and diffs are much more easily done than with command line tools. Also, having good integration with Eclipse makes simple operations like committing, pushing and, pulling possible without leaving the IDE. However, for more complex operations like rebasing and complex merges are typically more easily done on the command line.

To install eGit, use the following update site:

 Eclipse 4.2 (Juno) - <http://download.eclipse.org/releases/juno>

Eclipse 3.7 (Indigo) - <http://download.eclipse.org/releases/indigo>

Choose to install **eGit** (the visual tooling for git) and **jGit** (a java-based implementation of git),

No need to restart.

#### *Install Groovy-Eclipse*

Some of the projects contain Groovy code, so we need to have Groovy support installed.

**Preferably, the latest version of Groovy Eclipse should be installed, which can be obtained from the update site for the latest snapshot:**

 <http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.7>

or

<http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e4.2>

#### **Now, restart.**

##### *Import Groovy-Eclipse source code into your workspace (the simple way)*

The canonical Groovy-Eclipse repository is located at github. The easiest way to import the source code into your workspace is to use the following *Groovy-Eclipse team project set* [groovy-eclipse-project-set.psf](#). To use, go to File -> Import... -> Team project set and paste the URL. Importing the team project set will take a long time since the entire repository needs to be cloned. Currently, the psf file requires Eclipse 4.2.x.

##### *Import Groovy-Eclipse source code into your workspace (the complicated way)*

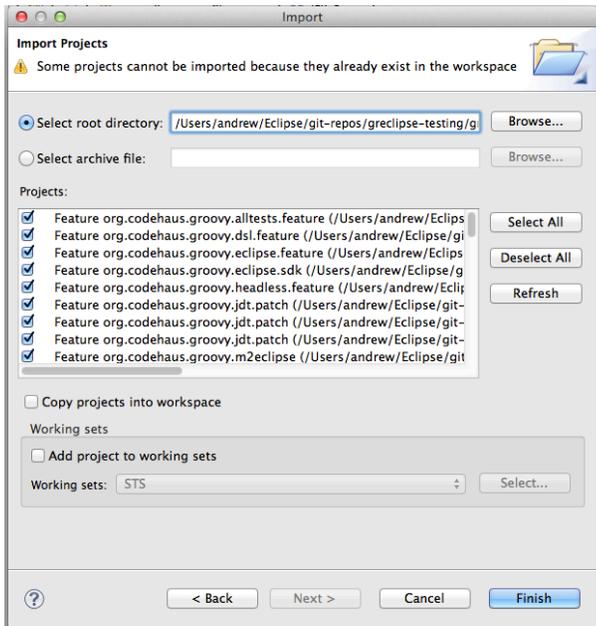
You can ignore this section if you imported the source code the simple way. This section provides a more detailed description of which projects to import and how to do it if you are not using the psf file above.

If you are running git from the command line, use the following command to clone the repository:

 `git clone git://github.com/groovy/groovy-eclipse.git`

Have a cup of coffee or a beer while the project is cloned.

After cloning is complete, you must import the projects into your Eclipse workspace. Click on File Import Existing projects into workspace. Choose the top-level git repository folder and search for all projects recursively. You will see something like this, with many, many projects available for import:



Or, you can clone from inside of eGit. Use the git repository url above and follow the [directions on the eGit documentation](#).

You do not need to import all projects, and some projects conflict with each other. Here are some suggestions on what to import:

- **IMPORTANT:** there are multiple *org.eclipse.jdt.core* projects. These projects contain the patched Eclipse Java compiler and are specific to a single version of Eclipse. Only import the single project that matches your Eclipse level (e.g., E36, E37, or E42). To determine which project is appropriate for your eclipse level, look at the path to each of these projects, and you will see a segment like: *jdt-patch/e42*. Match your eclipse level to the path. **If you do not follow this step, you will see compile errors in the imported projects.**
- The projects *org.codehaus.groovy16*, *org.codehaus.groovy18*, and *org.codehaus.groovy20* projects contain the groovy 1.6, 1.8, and 2.0 compilers respectively. The *org.codehaus.groovy* project contains the groovy 1.7 compiler. It is not necessary to import all of these projects. If you import more than one compiler project, then the latest one will take precedence over the earlier ones.
- The *org.codehaus.groovy.m2eclipse* project requires m2eclipse to be installed in your target Eclipse. If m2eclipse is not available, there will be compile errors. This project provides maven integration when running in an Eclipse workbench. You do not need to import this project unless you intend to work on maven integration.
- Any project that contains the word *test* contains unit tests. It is recommended to import these projects.
- Any project prefixed with *Feature* as well as the *Site Groovy* does not contain code and can be ignored unless you know that you need it.
- Any project prefixed with *groovy-eclipse-* is part of the maven compiler plugin for groovy-eclipse. You only need to import these projects if you plan on working on the maven support.
- The *org.codehaus.groovy.eclipse.pluginbuilder* project contains the releng code to build Groovy-Eclipse and publish the update site.

#### **Run the tests**

Optionally, you may want to run the tests. Do **Run As->JUnit plugin test** on **org.codehaus.groovy.alltests**.

**Note:** There are two separate sets of tests in alltests plugin:

- AllGroovyTests.java — the UI tests (content assist, navigation, refactoring, etc) takes ~35-45 minutes to run
- GroovyJDTTests.java — the core tests (parser, compiler integration, type inferencing, searching, etc) takes

~10-15 minutes to run

At the time when I wrote this, \*running both test suites at once resulted in a number of test failures\* in the GroovyJDTTests suite. Running both the suites separately I got no such failures.

**Note:** If you get an Out of Memory Error like this:

```
java.lang.OutOfMemoryError: PermGen space
at java.lang.ClassLoader.defineClass1(Native
Method)
at
java.lang.ClassLoader.defineClassCond(ClassL
oader.java:632)
at
java.lang.ClassLoader.defineClass(ClassLoade
r.java:616)
...
```

You can increase the size of PermGen space in "Run As -> Run Configurations"

Add this to VMArguments:

```
-XX:MaxPermSize=256m
```

*Now, look for some interesting bugs to work on*

See the list in jira. All the issues labelled [help requested](#) are a good place to start. Also, you can contact the [eclipse plugin dev mailing list](#) and we can help you out.

### Extending Groovy Eclipse

See [here](#) for how to extend Groovy-Eclipse to support domain specific languages. After reading the article, download the extension example, which is a plugin project to be added to your Eclipse development workspace and the extension example runtime project, which should be imported into your runtime workspace.

#### 1306: Status Date

3/11/2015 15:20:10  
1306: Status Date

1/11/2015 15:20:10  
1306: Status Date

## Groovy-Eclipse 2.0.2 New and Noteworthy

We have just released Groovy-Eclipse 2.0.2. This release includes a number of exciting new features, Groovy 1.7.3, and fixes for over [80 jira issues](#). For this release, we have two streams available: an Eclipse 3.5 (Galileo) stream and an Eclipse 3.6 (Helios) stream. To install 2.0.2, you can add one of the following release URLs to your Eclipse update manager:

### Groovy-Eclipse update site for Eclipse 3.5

<http://dist.springsource.org/release/GRECLIPSE/e3.5/>

### Groovy-Eclipse update site for Eclipse 3.6

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

The theme of this release is *Refactoring, Refactoring, Formatting*.

## Outline

- [Refactoring support](#)
  - [Extract Local Variable Refactoring](#)
  - [Extract Constant Refactoring](#)
  - [Extract Method Refactoring](#)
  - [Rename Refactoring](#)
- [Groovy Editor](#)
  - [Code Formatting and Indentation](#)
  - [GroovyDoc](#)
  - [Expand and contract selection](#)
- [Better AST Transform support](#)
- [What's next?](#)

### Refactoring support

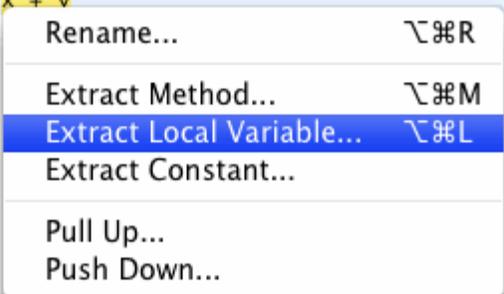
For this release, we have done some work on refactoring. In the past, there was a separate *Groovy Refactoring* menu, which was the only place Groovy refactorings could be invoked from. Now, we have integrated the refactorings into the JDT refactoring support. Now, Groovy refactorings can be invoked using the same keyboard commands and menu items as their Java counterparts.

### *Extract Local Variable Refactoring*

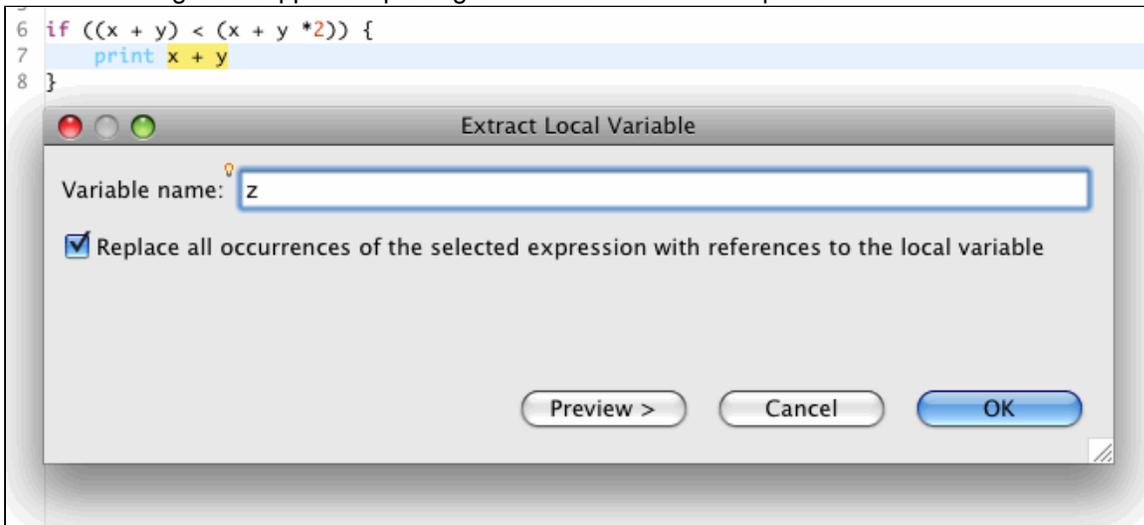
The extract local variable refactoring allows programmers to select an expression and convert it into a local variable:

- Initial code:

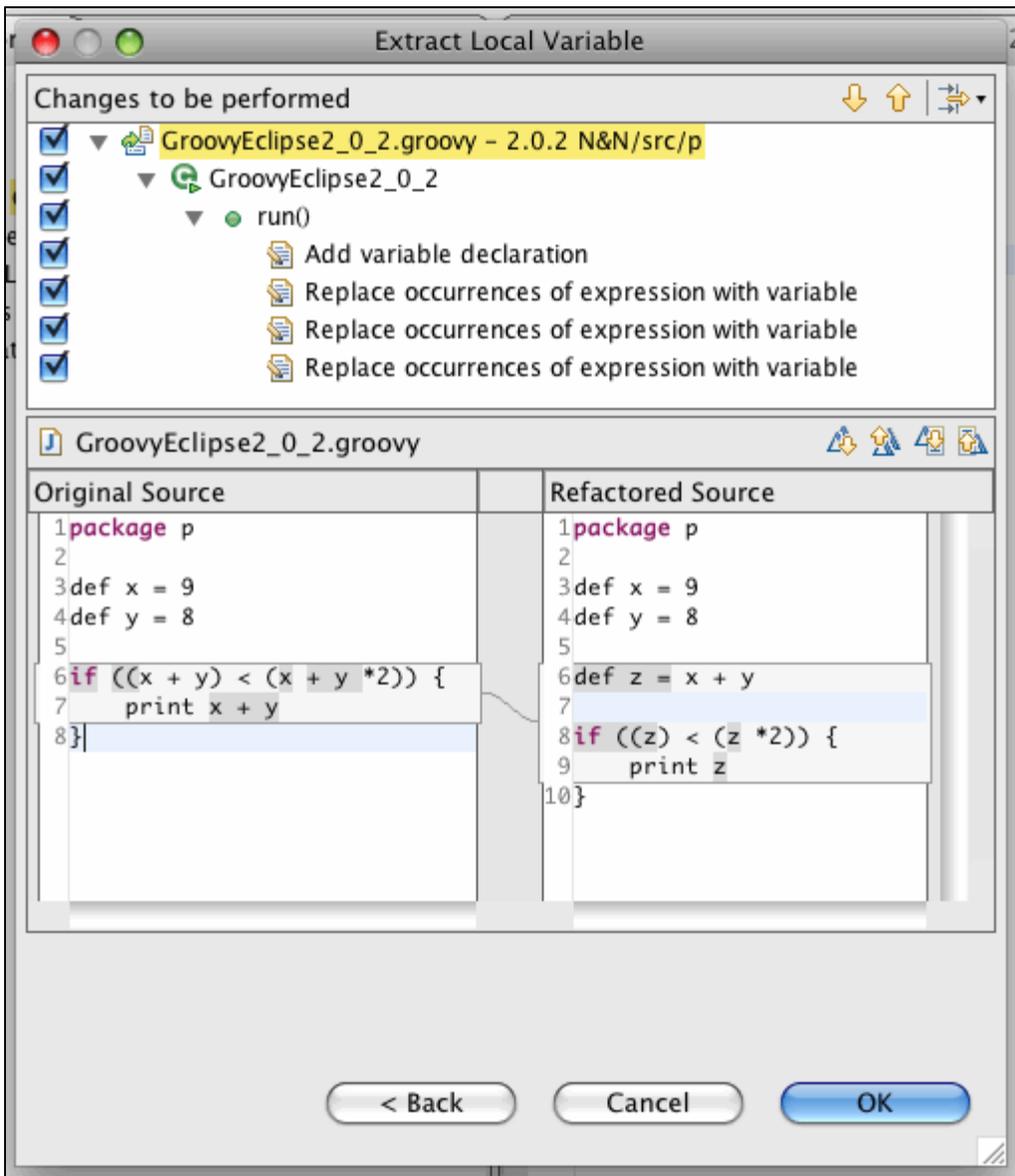
```
1 package p
2
3 def x = 9
4 def y = 8
5
6 if ((x + y) < (x + y *2)) {
7   print x + y
8 }
```



- This refactoring also supports replacing all occurrences of the expression:



- Preview:



- See the results:

```

1 package p
2
3 def x = 9
4 def y = 8
5
6 def z = x + y
7
8 if ((z) < (z * 2)) {
9     print z
10 }

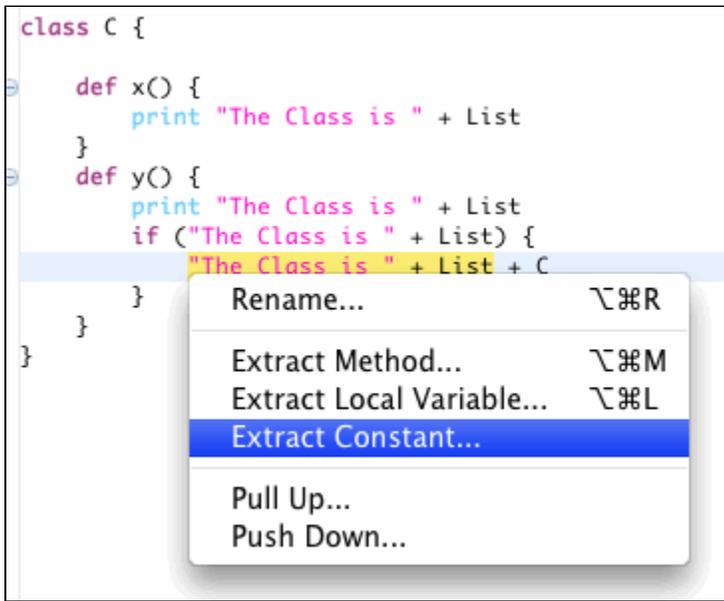
```

The keyboard binding for this refactoring is the same as it is for the Java variant of the refactoring (e.g., CMD-Alt-L on Macs).

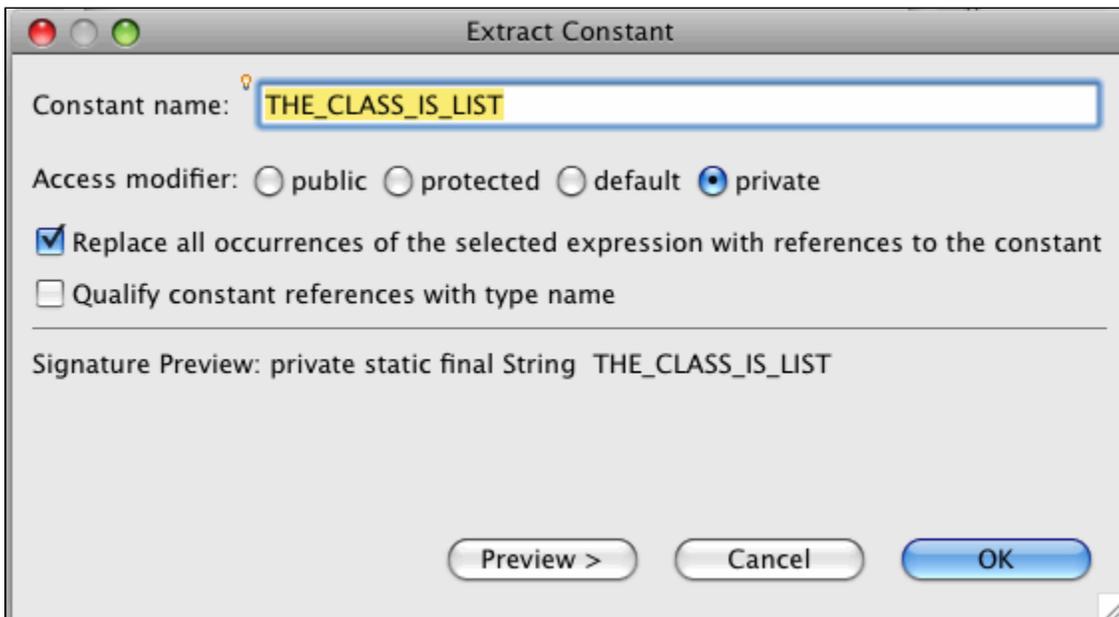
#### ***Extract Constant Refactoring***

Extract constant expression allows a programmer to select a static expression, and pull it out as a constant in the current class. This refactoring is also available from the refactoring context menu.

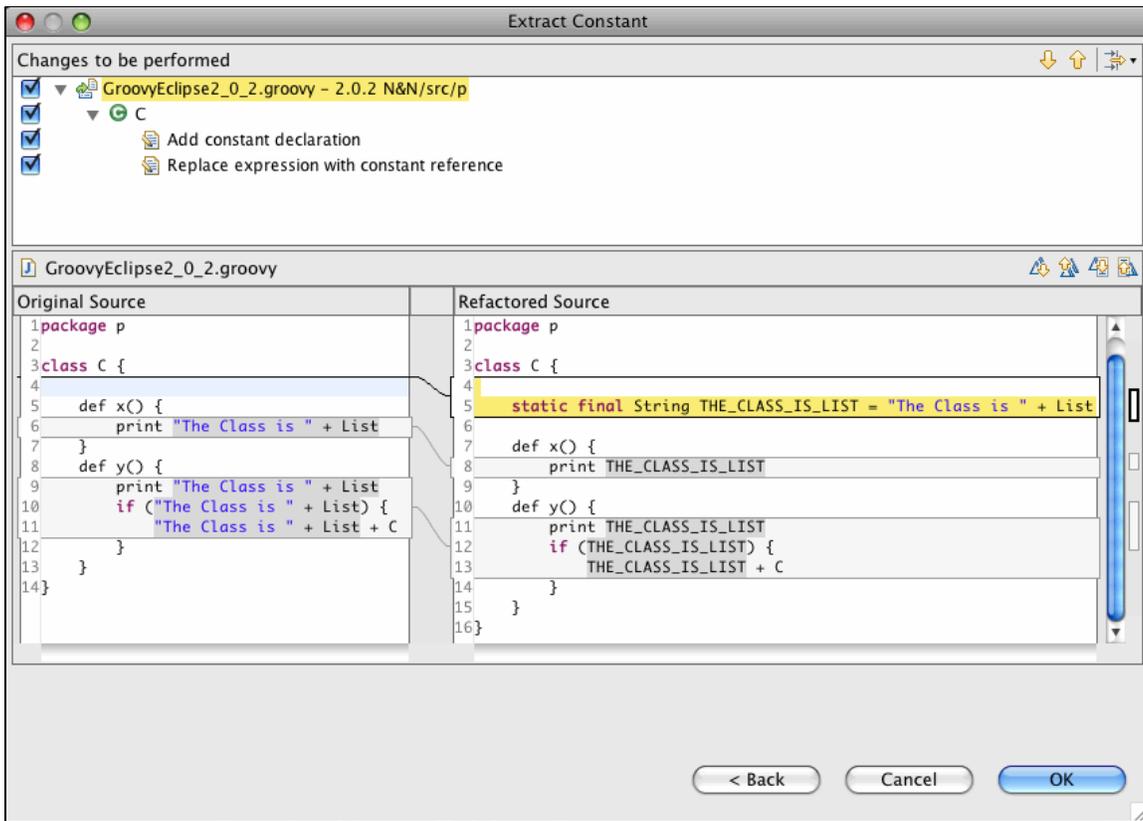
- To invoke this refactoring, select a static expression and select the *Extract Constant...* refactoring:



- In the refactoring wizard, you can choose to replace all occurrences with the constant, and you can also choose to qualify references to the constant with the class name. Also notice that the refactoring provides a reasonable suggestion for an initial name:



- As with extract local, you can see the preview:



- And the result is what you would expect with all text properly replaced:

```
class C {
    static final String THE_CLASS_IS_LIST = "The Class is " + List

    def x() {
        print THE_CLASS_IS_LIST
    }
    def y() {
        print THE_CLASS_IS_LIST
        if (THE_CLASS_IS_LIST) {
            THE_CLASS_IS_LIST + C
        }
    }
}
```

There are two caveats to be aware of:

- Extract constant inside of scripts is disallowed
- All sub-expressions of the expression to extract must be static

#### **Extract Method Refactoring**

Extract method refactoring was available in version 2.0.0 and earlier, but was removed for 2.0.1 because it depended on deprecated internal APIs that had been removed.

- For 2.0.2, we have reimplemented extract the method refactoring and linked it to the standard Java method refactoring in the editor:

```
1 package p
2
3 def x = 9
4 def y = 8
5
6 def g = x + y
7 def h = g * 2
8
9 print h
```

- This refactoring supports reordering and renaming of the parameters of the extracted method:



- As expected, the result properly extract the selected statements into a new method, taking into account new parameters and the return statement:

```

1 package p
2
3 def x = 9
4 def y = 8
5
6 def h = extracted(x, y)
7 |
8 print h
9
10 private extracted(int x, int y) {
11     def g = x + y
12     def h = g * 2
13     return h
14 }

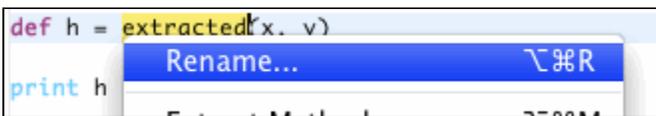
```

The limitations to this refactoring are the following:

- Only complete statements or groups of complete statements can be refactored.
- Multiple return values are not supported.

#### **Rename Refactoring**

Although originally introduced in the 2.0.1 version, rename refactoring is now available from the context menu:



And the menu bar:



or keyboard commands (e.g., Alt-CMD-R on MacOS).

#### **Groovy Editor**

##### **Code Formatting and Indentation**

In this release, we focused on fixing issues that directly affect a user's day-to-day experience, such as typing, copying and pasting. We have integrated a number of "smart" editing transformations into the Groovy Editor. In the past, we inherited some of the Java-like editing transformations, which would sometimes fail due to incompatibilities between the Groovy and Java grammars.

In this release we are providing a new groovy-specific implementations for three *smart editor* features:

- Smart paste: attempt to preserve relative indentation of code pasted into a Groovy editor.
- Smart newline: try to put the cursor in a sensible place after the user presses newline.
- Smart tab: correct the indentation of the current line when pressing tab at start of a line.

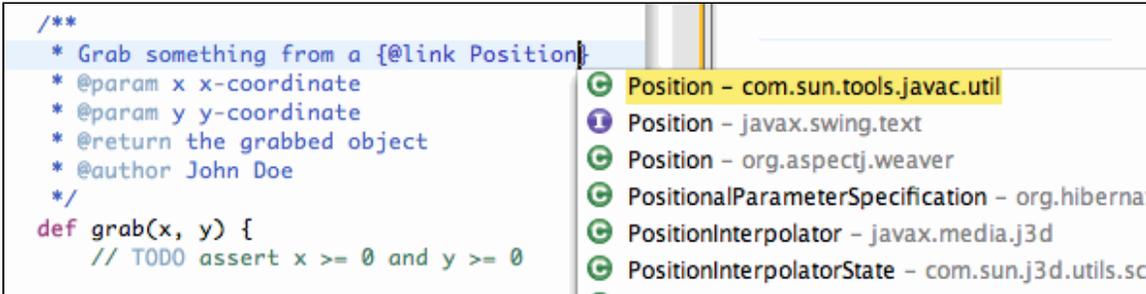
Like their Java counterparts, these implementations rely on heuristics rather than a full parser, so they should work even when the document in the editor is not parsable. These implementations are brand new and we are looking forward to getting your feedback on how to make them work better and polish them up.

Besides these "on-line" algorithms that get invoked while typing and pasting. The Groovy editor also provides an "off-line" formatter that can be invoked explicitly by pressing CTRL-I or CTRL-SHIFT-F. Numerous bugfixes and

small improvements have been made to the of-line Groovy formatter and indenter as well.

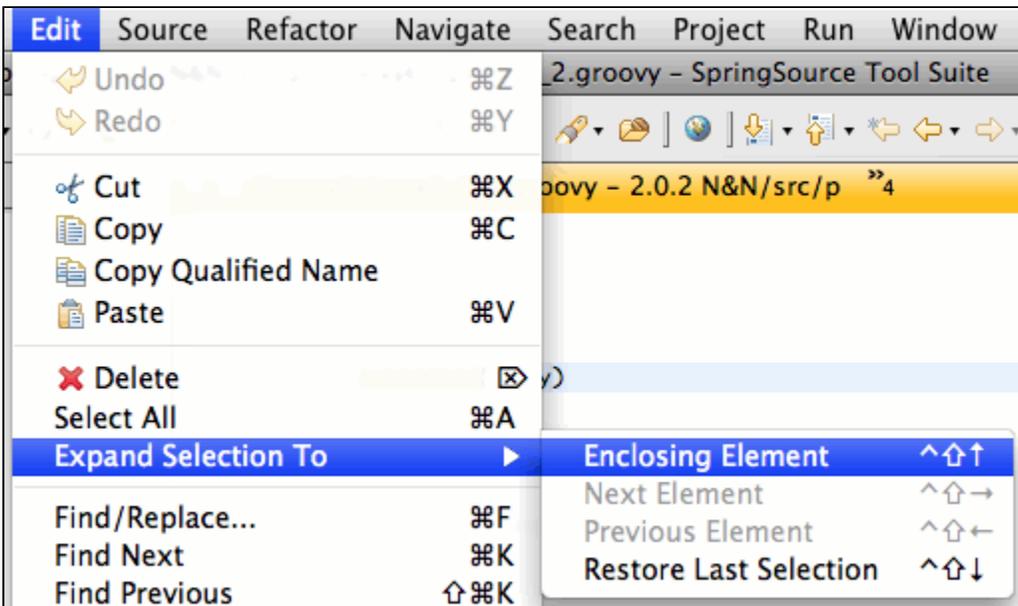
### GroovyDoc

The Groovy editor now properly recognizes GroovyDocs in Groovy code. This means you will now have the same support for highlighting and editing GroovyDoc as for JavaDoc in the Java editor. This includes content assist for GroovyDoc tags and Groovy classes.



### Expand and contract selection

An often-requested feature has been to implement expand and contract selection from within the Groovy editor. This is now available, and uses the same keyboard combination as expand/contract selection in the Java editor uses:



You can use the *Expand Selection to -> Enclosing Element* command to select the surrounding expression, statement or declaration, and then you can return to the previous selection by using the *Expand Selection to -> Restore Last Selection* command (keyboard shortcuts vary across OS's, and see previous screenshot for the shortcuts on Macs).

As an example:

```
def x = 7 + 9
```

**expand**

```
def x = 7 + 9
```

# restore

```
def x = 7 + 9
```

## Better AST Transform support

In previous versions of Groovy-Eclipse, AST Transforms would often mangle source code and source locations to the extent that the Groovy Editor could no longer find the lexical location of Groovy code elements. We have solved this problem in 2.0.2. For more information, see [GRECLIPSE-617](#).

## What's next?

The Groovy-Eclipse team will continue to work on improving the development experience of writing Groovy code in Eclipse as we move towards the 2.1.0 release in early Autumn. You can find more details on the [Groovy-Eclipse Roadmap](#), where you can comment on proposed features and see our release schedule.

We appreciate all the community support and feedback we have had through the version 2.0.2 development cycle. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you'd like to see), please raise them in our [issue tracker](#).

## Groovy-Eclipse 2.1.0RC2 New and Noteworthy

We have just released Groovy-Eclipse 2.1.0.RC2. This release includes a number of exciting new features and fixes for over 50 jira issues

([http://jira.codehaus.org/secure/IssueNavigator.jspa?reset=true&jqlQuery=project+%3D+GRECLIPSE+AND+resolution+in+\(Fixed,+%22Won't+Fix%22,+Duplicate,+Incomplete,+%22Cannot+Reproduce%22,+%22Not+A+Bug%22\)+AND+fixVersion+in+\(%222.1.0Release%22,+%222.1.0rc1%22\)+ORDER+BY+updated+DESC](http://jira.codehaus.org/secure/IssueNavigator.jspa?reset=true&jqlQuery=project+%3D+GRECLIPSE+AND+resolution+in+(Fixed,+%22Won't+Fix%22,+Duplicate,+Incomplete,+%22Cannot+Reproduce%22,+%22Not+A+Bug%22)+AND+fixVersion+in+(%222.1.0Release%22,+%222.1.0rc1%22)+ORDER+BY+updated+DESC)). For this milestone release, we have focussed on the Eclipse 3.6 stream of Groovy-Eclipse, and are continuing to maintain the 3.5 stream. You can download the milestone releases here:

### Groovy-Eclipse update site for Eclipse 3.6

<http://dist.springsource.org/milestone/GRECLIPSE/e3.6/>

### Groovy-Eclipse update site for Eclipse 3.5

<http://dist.springsource.org/milestone/GRECLIPSE/e3.5/>

In this release, we have focussed on the inferencing engine and content assist.

## Outline

- [Outline](#)
- [Compatibility](#)
- [Update site zips](#)
- [Generics-aware type inferencing](#)
- [Generics-aware hover support](#)
- [Improved content assist](#)
  - [Sorting](#)
  - [Extension point for proposal filtering and sorting](#)
- [Script folders](#)
- [What's next?](#)

## Compatibility

Groovy-Eclipse 2.1.0.RC1 includes Groovy 1.7.5 and is installable on Eclipse 3.5.2, 3.6.0, and 3.6.1.

#### Update site zips

Zipped versions of the update sites are available at:

#### **i** Zipped Groovy-Eclipse update site for Eclipse 3.6

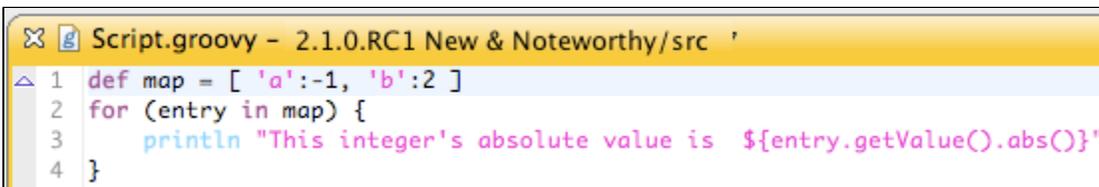
<http://dist.springsource.org/milestone/GRECLIPSE/e3.6/archive-2.1.0.xx-20101007-1100-e36-RC2.zip>

#### **i** Zipped Groovy-Eclipse update site for Eclipse 3.5

<http://dist.springsource.org/milestone/GRECLIPSE/e3.5/archive-2.1.0.20100926-2100-e35-RC1.zip>

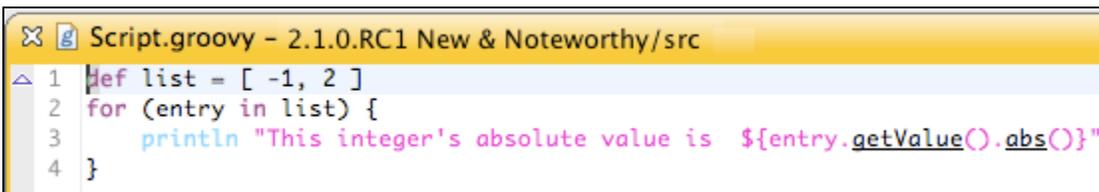
#### Generics-aware type inferencing

The Groovy-Eclipse inferencing engine is now aware of generic types. For example, the type of the loop variable in a for loop is inferred from the type of the collection expression:



```
Script.groovy - 2.1.0.RC1 New & Noteworthy/src
1 def map = [ 'a':-1, 'b':2 ]
2 for (entry in map) {
3     println "This integer's absolute value is ${entry.getValue().abs()}"
4 }
```

And, of course, the editor will notify you when you are using a loop variable incorrectly:



```
Script.groovy - 2.1.0.RC1 New & Noteworthy/src
1 def list = [ -1, 2 ]
2 for (entry in list) {
3     println "This integer's absolute value is ${entry.getValue().abs()}"
4 }
```

There are a couple of pieces not yet implemented in this area, that we are planning post 2.1.0. First, inferencing of the type of the closure variable inside of `DefaultGroovyMethods` is not available yet. For example:

```
def x = [1, 2]
x.each { print it.abs() }
```

In this situation, the type of `it` is `Integer` and the `abs()` method is valid, but it will be underlined in the Groovy editor.

Also, type inferencing for maps and lists is currently simple. The static type of the first element of the list or map is looked at. If the type can be determined, then it is used, otherwise `Object` is used.

#### Generics-aware hover support

Similarly, code hovers now display the generics information of the value being selected:

```
def map = [ 'a':-1, 'b':2 ]
for (entry in map) {
    println "T"
}
```

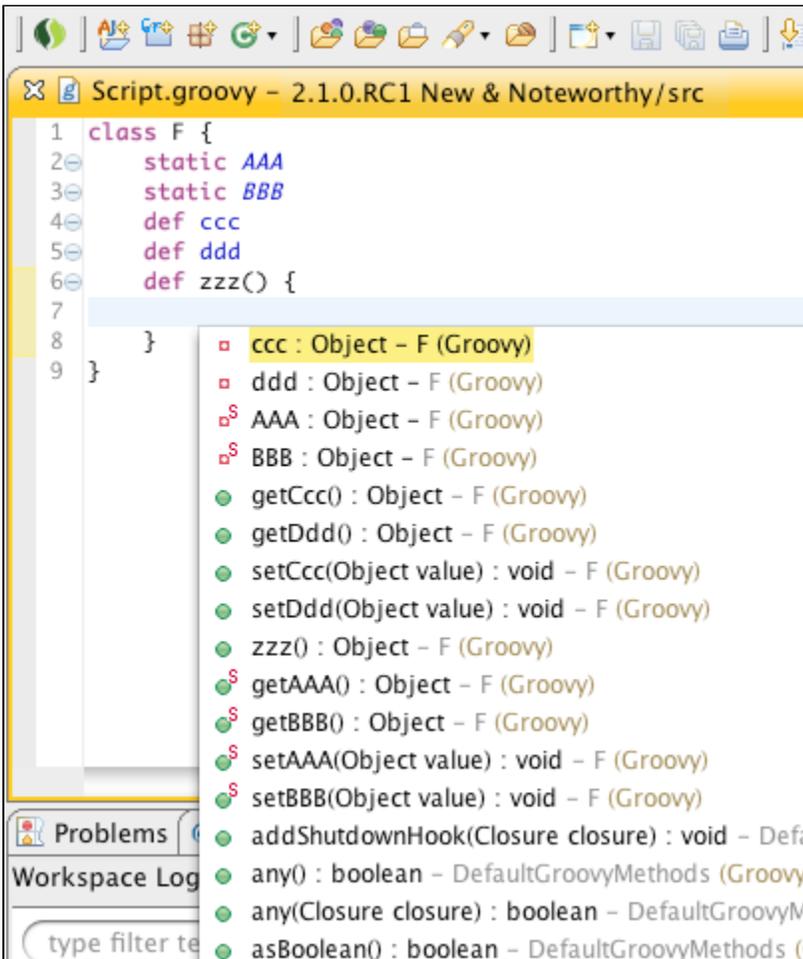
Map<String, Integer> map - Script.run()

### Improved content assist

We have made a few improvements to content assist to make it easier for users. In addition to the improved type inferencing described above, which will provide better proposals for lists and maps, we have worked on a better sorting algorithm for proposals.

## Sorting

In prior versions of Groovy-Eclipse, the most commonly used content assist proposals such as local variables often got lost amidst rarely used proposals such as Default Groovy Methods and type proposals. Now, we have reworked our relevance algorithm and we ensure that local variables are above, instance methods and fields are next, and type proposals are always last.



As you can see in the screenshot above, the ordering is non-static fields, static fields, non-static methods, static methods, default groovy methods, and finally types.

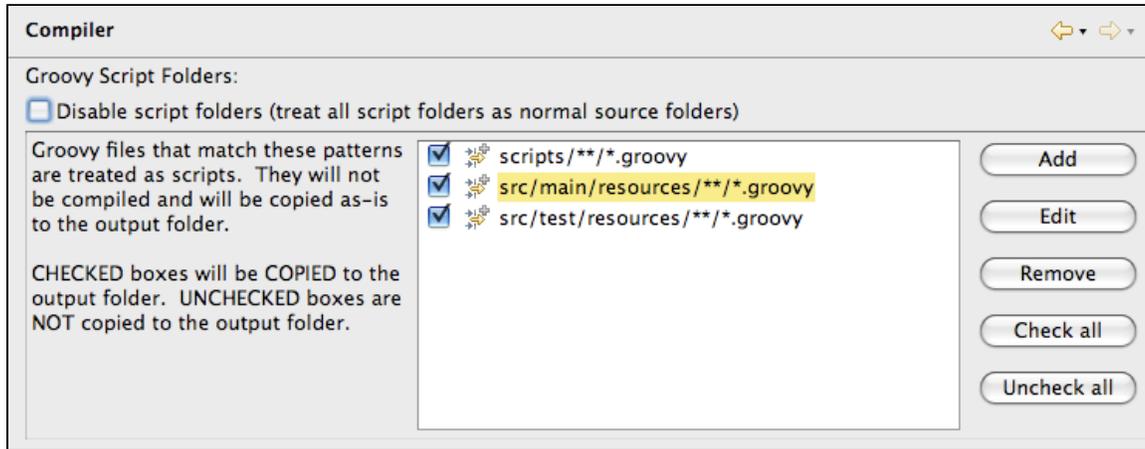
## Extension point for proposal filtering and sorting

For Groovy DSL implementors, we have provided a new extension point for Groovy-Eclipse that allows third-party plugins to provide additional sorting and filtering for Groovy content assist. For more information, see the [org.codehaus.groovy.eclipse.codeassist.completion.completionProposalFilter](#) extension point in the [org.codehaus.groovy.eclipse.codeassist.completion](#) plugin.

### Script folders

Groovy-Eclipse now supports the notion of *script folders*. A script folder is a source folder that is meant to contain Groovy scripts. Groovy scripts are not sent to the compiler, are optionally copied to the appropriate output folder, and have fully editing support enabled (e.g., content assist, hovers, underlining, etc.).

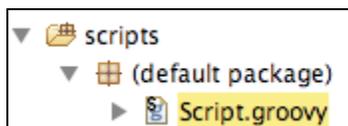
Script folders can be configured in Eclipse Preferences -> Groovy -> Compiler preference page:



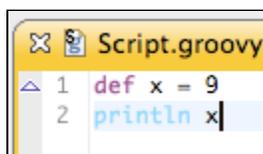
Several script folders are included by default, which can be changed if required. Each entry is a regular expression that captures a set of Groovy files to be treated as scripts. The syntax of these regular expressions are similar to the syntax of [Apache Ant patternSets](#). By default, script folders are *disabled* workspace-wide. Uncheck the "Disable script folders..." check box to enable them.

Also, each filter entry in the script folders list has a check box. If the box is *checked* then scripts matching that filter entry are copied to the output folder. If *unchecked*, the scripts are not copied to the output folder.

Groovy scripts are designated through a variant of the groovy icon in the Package Explorer:



and in the Editor



### What's next?

The 2.1.0 final release is scheduled for a week from now to coincide with the 2.5.0 release of the [SpringSource Tool Suite](#) before [SpringOne 2GX](#) in October. Because this version is a release candidate, only critical bug fixes will be made between now and the final release.

We appreciate community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you'd like to see), please raise them in our [issue tracker](#).

## Groovy-Eclipse 2.1.0 New and Noteworthy

We have just released Groovy-Eclipse 2.1.0.RC2. This release includes a number of exciting new features and fixes for over [50 jira issues](#). For this release, we have focussed on the Eclipse 3.6 stream of Groovy-Eclipse, and are continuing to maintain the 3.5 stream. You can download the milestone releases here:

### Groovy-Eclipse update site for Eclipse 3.6

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

### Groovy-Eclipse update site for Eclipse 3.5

<http://dist.springsource.org/release/GRECLIPSE/e3.5/>

In this release, we have focussed on the inferencing engine and content assist.

#### Outline

- [Outline](#)
- [Compatibility](#)
- [Update site zips](#)
- [Generics-aware type inferencing](#)
- [Generics-aware hover support](#)
- [Improved content assist](#)
  - [Sorting](#)
  - [Extension point for proposal filtering and sorting](#)
- [Script folders](#)
- [What's next?](#)

#### Compatibility

Groovy-Eclipse 2.1.0 includes Groovy 1.7.5 and is installable on Eclipse 3.5.2, 3.6.0, and 3.6.1.

#### Update site zips

Zipped versions of the update sites are available at:

### Zipped Groovy-Eclipse update site for Eclipse 3.6

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.1.0.xx-20101020-1000-e36-release.zip>

### Zipped Groovy-Eclipse update site for Eclipse 3.5

<http://dist.springsource.org/release/GRECLIPSE/e3.5/archive-2.1.0.20101020-2100-e35-release.zip>

#### Generics-aware type inferencing

The Groovy-Eclipse inferencing engine is now aware of generic types. For example, the type of the loop variable in

a for loop is inferred from the type of the collection expression:

A screenshot of the Eclipse IDE showing a Groovy script. The code is: 

```
1 def map = [ 'a':-1, 'b':2 ]
2 for (entry in map) {
3     println "This integer's absolute value is ${entry.getValue().abs()}"
4 }
```

And, of course, the editor will notify you when you are using a loop variable incorrectly:

getValue().abs()}" 4 }"/>A screenshot of the Eclipse IDE showing a Groovy script. The code is: 

```
1 def list = [ -1, 2 ]
2 for (entry in list) {
3     println "This integer's absolute value is ${entry.getValue().abs()}"
4 }
```

There are a couple of pieces not yet implemented in this area, that we are planning post 2.1.0. First, inferring of the type of the closure variable inside of `DefaultGroovyMethods` is not available yet. For example:

```
def x = [1, 2]
x.each { print it.abs() }
```

In this situation, the type of `it` is `Integer` and the `abs()` method is valid, but it will be underlined in the Groovy editor.

Also, type inferring for maps and lists is currently simple. The static type of the first element of the list or map is looked at. If the type can be determined, then it is used, otherwise `Object` is used.

#### Generics-aware hover support

Similarly, code hovers now display the generics information of the value being selected:

A screenshot of the Eclipse IDE showing a Groovy script. The code is: 

```
def map = [ 'a':-1, 'b':2 ]
for (entry in map) {
    println "T
}
```

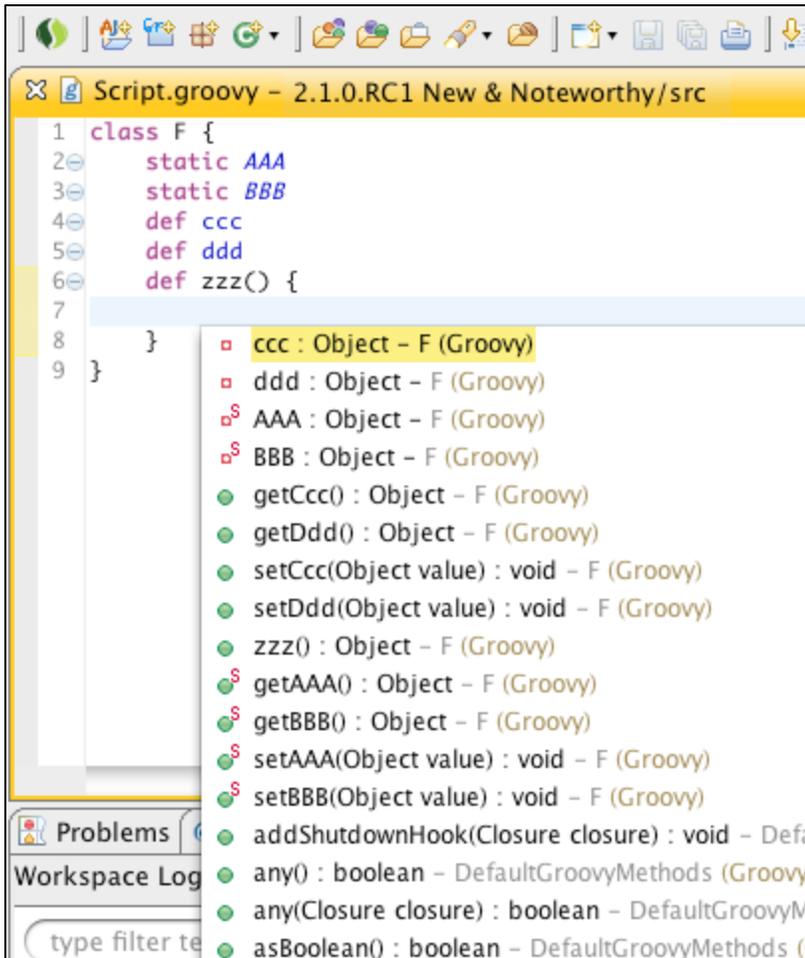
 A hover tooltip is visible over the `map` variable, displaying: `Map<String, Integer> map - Script.run()`

#### Improved content assist

We have made a few improvements to content assist to make it easier for users. In addition to the improved type inferring described above, which will provide better proposals for lists and maps, we have worked on a better sorting algorithm for proposals.

## Sorting

In prior versions of Groovy-Eclipse, the most commonly used content assist proposals such as local variables often got lost amidst rarely used proposals such as Default Groovy Methods and type proposals. Now, we have reworked our relevance algorithm and we ensure that local variables are above, instance methods and fields are next, and type proposals are always last.



As you can see in the screenshot above, the ordering is non-static fields, static fields, non-static methods, static methods, default groovy methods, and finally types.

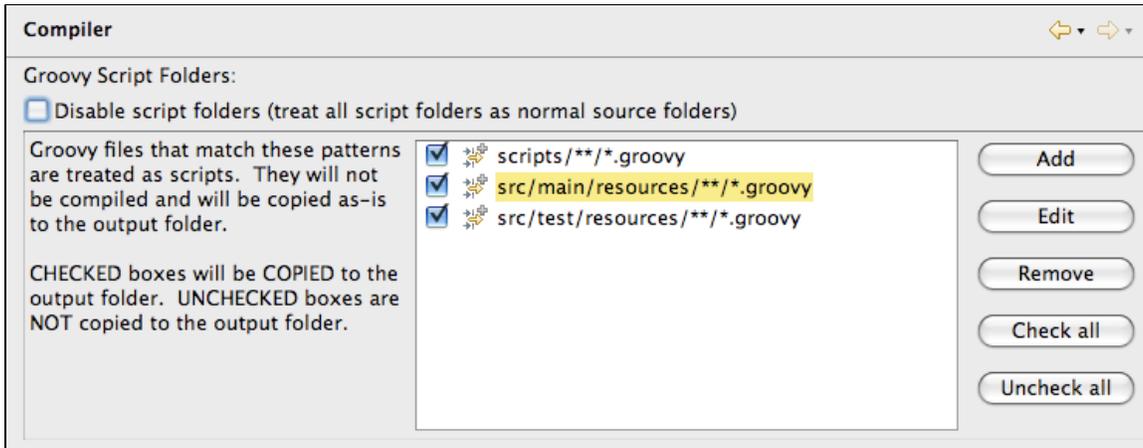
## Extension point for proposal filtering and sorting

For Groovy DSL implementors, we have provided a new extension point for Groovy-Eclipse that allows third-party plugins to provide additional sorting and filtering for Groovy content assist. For more information, see the [org.codehaus.groovy.eclipse.codeassist.completion.completionProposalFilter](#) extension point in the [org.codehaus.groovy.eclipse.codeassist.completion](#) plugin.

### Script folders

Groovy-Eclipse now supports the notion of *script folders*. A script folder is a source folder that is meant to contain Groovy scripts. Groovy scripts are not sent to the compiler, are optionally copied to the appropriate output folder, and have fully editing support enabled (e.g., content assist, hovers, underlining, etc.).

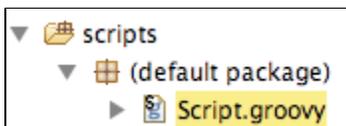
Script folders can be configured in Eclipse Preferences -> Groovy -> Compiler preference page:



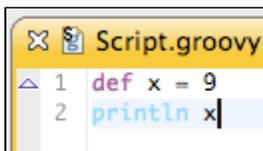
Several script folders are included by default, which can be changed if required. Each entry is a regular expression that captures a set of Groovy files to be treated as scripts. The syntax of these regular expressions are similar to the syntax of [Apache Ant patternSets](#). By default, script folders are *disabled* workspace-wide. Uncheck the "Disable script folders..." check box to enable them.

Also, each filter entry in the script folders list has a check box. If the box is *checked* then scripts matching that filter entry are copied to the output folder. If *unchecked*, the scripts are not copied to the output folder.

Groovy scripts are designated through a variant of the groovy icon in the Package Explorer:



and in the Editor



#### What's next?

The 2.1.0 final release is scheduled for a week from now to coincide with the 2.5.0 release of the [SpringSource Tool Suite](#) before [SpringOne 2GX](#) in October. Because this version is a release candidate, only critical bug fixes will be made between now and the final release.

We appreciate community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you'd like to see), please raise them in our [issue tracker](#).

#### Groovy-Eclipse 2.1.1RC1 New and Noteworthy

We have just released Groovy-Eclipse 2.1.1.RC1. Just 6 weeks after our previous release of 2.1.0, we are starting the release process again. There are many exciting new features to look forward to.

You can use the following update site to install this release.

### **i** Groovy-Eclipse update site for Eclipse 3.6

<http://dist.springsource.org/milestone/GRECLIPSE/e3.6/>

And a zipped version of the update site is available at:

### **i** Zipped Groovy-Eclipse update site for Eclipse 3.6

<http://dist.springsource.org/milestone/GRECLIPSE/e3.6/archive-2.1.1.xx-20101209-2200-e36-RC1.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

This new and noteworthy is for the release candidate and does not contain images. A more complete new and noteworthy will be available for the final release expected in about a week.

#### *Outline*

- [Inferencing inside of closures](#)
- [Quickfixes](#)
- [Inferencing of statically imported fields and methods](#)
- [Content assist for Constructors](#)
- [Improved semantic highlighting](#)
- [Organize imports improvements](#)
- [Create Groovy Script](#)
- [Outline view for binary Groovy types](#)
- [Bookmarks](#)
- [Compatibility](#)
- [Bug fixes](#)
- [What's next?](#)

#### *Inferencing inside of closures*

Type inferencing and content assist inside of closures now uses the type of calling method as the type of `this`. For example:

```
new Foo().run {
    this
}
```

Hovering over `this` in the editor and the bubble will show that `this` is of type `Foo`. See [GRECLIPSE-916](#).

#### *Quickfixes*

We now have quickfixes in the Groovy editor, see [GRECLIPSE-909](#). In this release, we introduce two quickfixes: convert to Groovy file ([GRECLIPSE-910](#)) and add import ([GRECLIPSE-911](#)).

#### *Inferencing of statically imported fields and methods*

Statically imported fields and methods are now available from content assist and in the inferencing engine. See [GRE CLIPSE-698](#) for more information.

#### *Content assist for Constructors*

Content assist now includes appropriate constructors when invoked in front of a new keyword. See [GRECLIPSE-839](#).

#### *Improved semantic highlighting*

Semantic highlighting now highlights field references, static references, and deprecated references. See [GRECLIPSE-869](#) and [GRECLIPSE-708](#).

#### *Organize imports improvements*

Organize imports in Groovy scripts now respects the settings in Preferences -> Java -> Code Style -> Organize Imports. Imports are sorted appropriately and grouped according to preferences. Also, semi-colons are now automatically removed. See [GRECLIPSE-498](#).

#### *Create Groovy Script*

The New Groovy Class wizard now has an option to create a Groovy script instead of a standard class. See [GRECLIPSE-485](#).

#### *Outline view for binary Groovy types*

When binary Groovy types are shown in the outline view and the package explorer all synthetic methods and fields are filtered from the view. See [GRECLIPSE-865](#).

#### *Bookmarks*

It is now possible to add a bookmark in a Groovy file by right-clicking on the editor ruler and selecting 'Add Bookmark...'. See [GRECLIPSE-853](#).

#### *Compatibility*

Groovy-Eclipse 2.1.1.RC1 includes Groovy 1.7.5 and is installable on Eclipse 3.6.0 and 3.6.1. There are no plans to release a 3.5 version of this release.

#### *Bug fixes*

Over [60 issues](#) have been fixed for this release.

#### *What's next?*

The 2.1.1 final release is scheduled for a week from now to coincide with the 2.5.2 release of the [SpringSource Tool Suite](#). Because this version is a release candidate, only critical bug fixes will be made between now and the final release.

We appreciate community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#).

### **Groovy-Eclipse 2.1.1 New and Noteworthy**

In the final release of Groovy-Eclipse 2.1.1, just 6 weeks after our previous release of 2.1.0, there are many exciting new features to look forward to.

You can use the following update site to install this release. To install, copy and paste this link into your Eclipse update manager:

**i Groovy-Eclipse update site for Eclipse 3.6**

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

And a zipped version of the update site is available at:

**i Zipped Groovy-Eclipse update site for Eclipse 3.6**

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.1.1.xx-20101215-2100-e36.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions provided by the update manager.

**Outline**

- [Inferencing inside of closures](#)
- [Quickfixes](#)
- [Inferencing of statically imported fields and methods](#)
- [Content assist for Constructors](#)
- [Improved semantic highlighting](#)
- [Organize imports improvements](#)
- [Create Groovy Script](#)
- [Outline view for binary Groovy types](#)
- [Bookmarks and Tasks](#)
- [Groovy Event Console](#)
- [Compatibility](#)
- [Bug fixes](#)
- [Thanks!](#)
- [What's next?](#)

**Inferencing inside of closures**

Type inferencing and content assist inside of closures now uses the type of calling method as the type of `this`. For example, the following code snippet with use `User` as the type of `this` inside the closure:

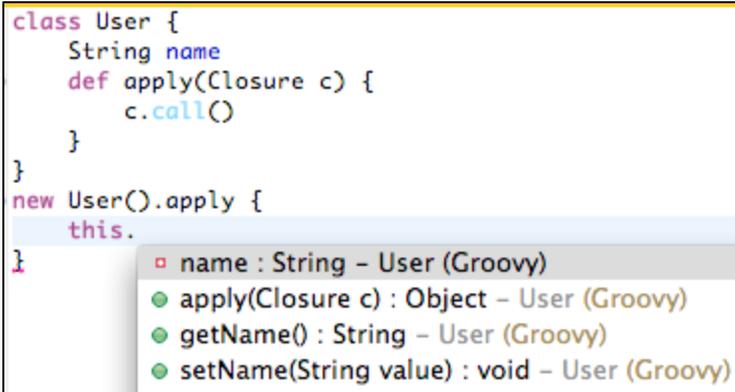
```
class User {
    String name
    def apply(Closure c) {
        c.call()
    }
}
new User().apply {
    this.name
}
```

String User.name

Similarly, content assist will work as expected:

```
class User {
    String name
    def apply(Closure c) {
        c.call()
    }
}
new User().apply {
    this.
}

```



A screenshot of a code editor showing a Groovy class `User` with a `name` property and an `apply` method. Below the code, a tooltip is displayed, listing the class name and several methods: `apply(Closure c) : Object - User (Groovy)`, `getName() : String - User (Groovy)`, and `setName(String value) : void - User (Groovy)`.

For more information, see [GRECLIPSE-916](#).

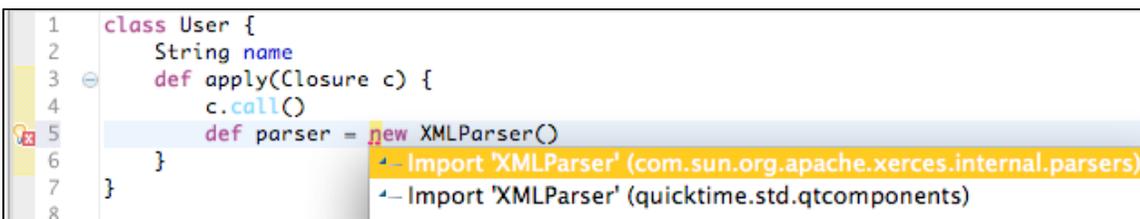
#### Quickfixes

We now have quickfixes in the Groovy editor, see [GRECLIPSE-909](#). In this release, we introduce two quickfixes: add import ([GRECLIPSE-911](#)) and convert to Groovy file ([GRECLIPSE-910](#)).

The quickfixes can be invoked like any other standard Java quickfix, either by clicking on the light bulb in the gutter:

```
1 class User {
2     String name
3     def apply(Closure c) {
4         c.call()
5         def parser = new XMLParser()
6     }
7 }
8

```



A screenshot of a code editor showing a Groovy class `User` with a `parser` property. A light bulb icon is visible in the gutter next to line 5. A tooltip is displayed, showing two import suggestions: `Import 'XMLParser' (com.sun.org.apache.xerces.internal.parsers)` and `Import 'XMLParser' (quicktime.std.qtcomponents)`.

By hovering over the exception:

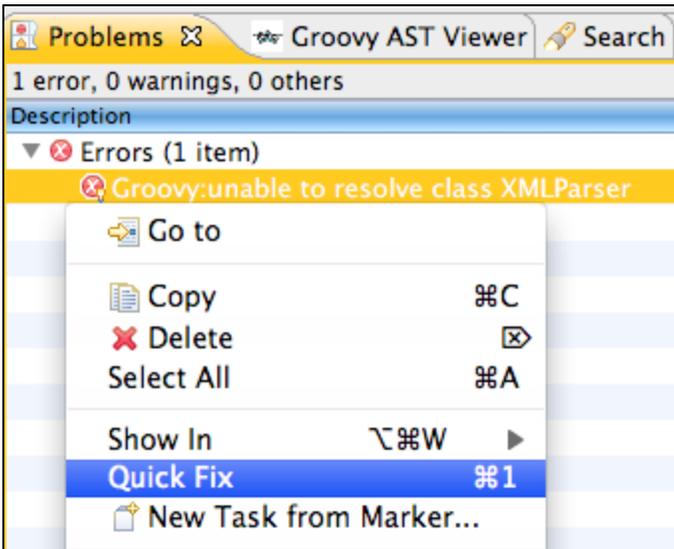
```
1 class User {
2     String name
3     def apply(Closure c) {
4         c.call()
5         def parser = new XMLParser()
6     }
7 }
8

```



A screenshot of a code editor showing a Groovy class `User` with a `parser` property. A light bulb icon is visible in the gutter next to line 5. A tooltip is displayed, showing a Groovy error message: `Groovy:unable to resolve class XMLParser`. Below the error message, it says "2 quick fixes available:" and lists two import suggestions: `Import 'XMLParser' (com.sun.org.apache.xerces.internal.parsers)` and `Import 'XMLParser' (quicktime.std.qtcomponents)`.

By using the keyboard shortcut (CTRL-1 or CMD-1 on macs), or through the problems view:

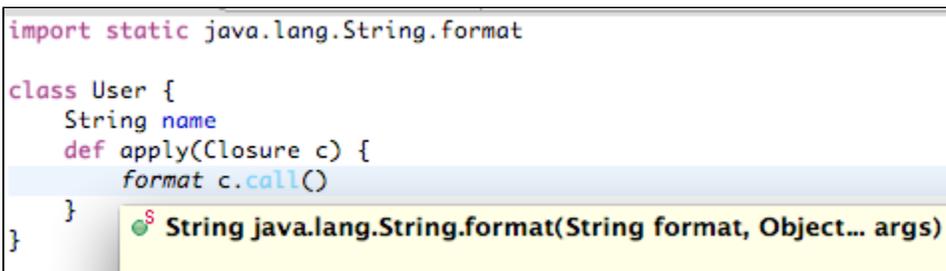


The convert to Groovy file quick fix is available whenever there is an error in your Java code that looks like Groovy syntax (for example, a missing semi-colon):



#### *Inferencing of statically imported fields and methods*

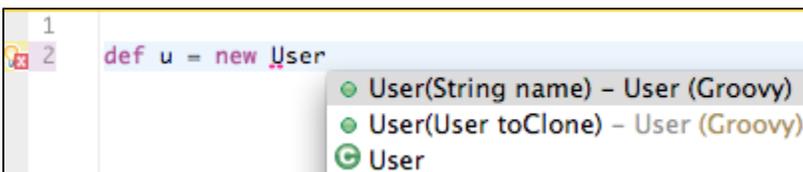
Statically imported fields and methods are now available from content assist and in the inferencing engine.



See [GRECLIPSE-698](#) for more information.

#### *Content assist for Constructors*

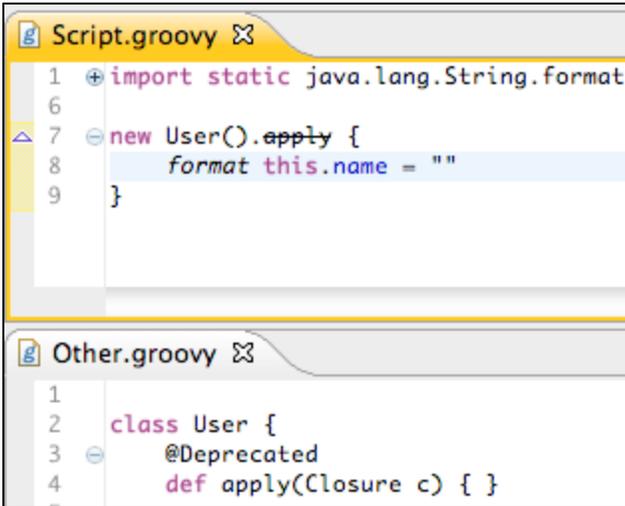
Content assist now includes appropriate constructors when invoked in front of a new keyword.



See [GRECLIPSE-839](#).

#### *Improved semantic highlighting*

Semantic highlighting now highlights field references, static references, and deprecated references as shown in the example below:



The screenshot shows two Groovy files in the Eclipse IDE. The top file, 'Script.groovy', contains the following code:

```
1 import static java.lang.String.format
6
7 new User().apply {
8     format this.name = ""
9 }
```

The bottom file, 'Other.groovy', contains the following code:

```
1
2 class User {
3     @Deprecated
4     def apply(Closure c) { }
```

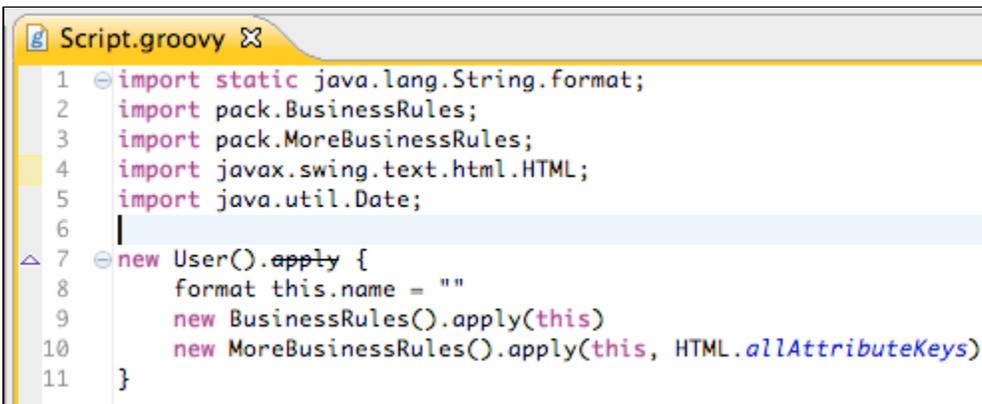
Semantic highlighting is visible: 'format' in line 8 of 'Script.groovy' is highlighted in blue, and '@Deprecated' in line 3 of 'Other.groovy' is highlighted in red.

See [GRECLIPSE-869](#) and [GRECLIPSE-708](#).

#### *Organize imports improvements*

Organize imports in Groovy scripts now respects the settings in Preferences -> Java -> Code Style -> Organize Imports. Imports are sorted appropriately and grouped according to preferences. Also, semi-colons are now automatically removed.

For example, the following simple script is originally:



The screenshot shows a Groovy file named 'Script.groovy' with the following code:

```
1 import static java.lang.String.format;
2 import pack.BusinessRules;
3 import pack.MoreBusinessRules;
4 import javax.swing.text.html.HTML;
5 import java.util.Date;
6
7 new User().apply {
8     format this.name = ""
9     new BusinessRules().apply(this)
10    new MoreBusinessRules().apply(this, HTML.allAttributeKeys)
11 }
```

The code is unformatted, with imports on separate lines and semi-colons at the end of each line.

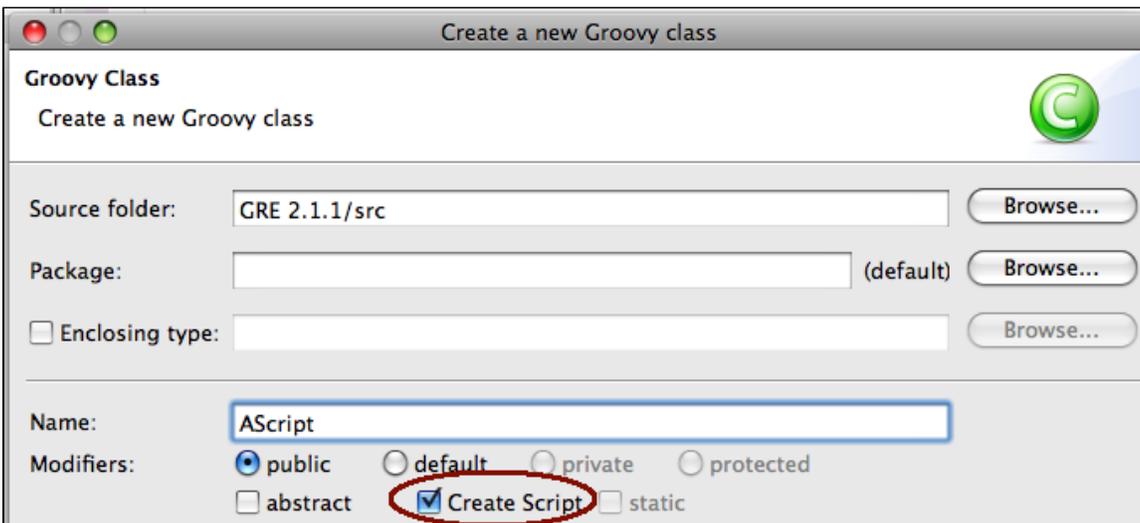
And becomes this after organize imports is invoked:

```
*Script.groovy ✕
1  import static java.lang.String.format
2
3  import javax.swing.text.html.HTML
4
5  import pack.BusinessRules
6  import pack.MoreBusinessRules
7
8  new User().apply {
9      format this.name = ""
10     new BusinessRules().apply(this)
11     new MoreBusinessRules().apply(this, HTML.allAttributeKeys)
12 }
```

See [GRECLIPSE-498](#).

#### Create Groovy Script

The New Groovy Class wizard now has an option to create a Groovy script instead of a standard class. To create a script, select the "Create Script" check box in the New Groovy class wizard as shown here:



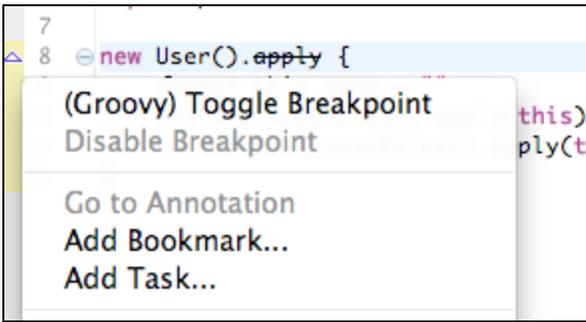
See [GRECLIPSE-485](#).

#### Outline view for binary Groovy types

When binary Groovy types are shown in the outline view and the package explorer all synthetic methods and fields are filtered from the view. See [GRECLIPSE-865](#).

#### Bookmarks and Tasks

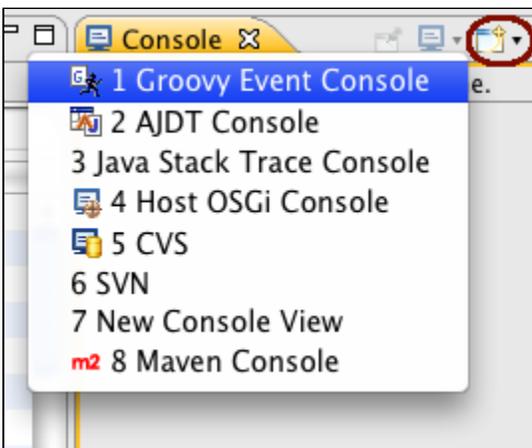
It is now possible to add a bookmark or a task in a Groovy file by right-clicking on the editor ruler and selecting 'Add Bookmark...' or 'Add Task...' as shown here:



See [GRECLIPSE-853](#).

#### *Groovy Event Console*

Groovy-Eclipse now ships with a Groovy event console that prints out information of the internal workings of Groovy-Eclipse. You can access the console from the console view and clicking on the *Open Console* button:



Most of this information is for debugging of Groovy-Eclipse itself and will be used in the future to help diagnose problems on users' machines.

#### *Compatibility*

Groovy-Eclipse 2.1.1.RC1 includes Groovy 1.7.5 and is installable on Eclipse 3.6.0 and 3.6.1. There are no plans to release a 3.5 version of this release.

#### *Bug fixes*

Over [65 issues](#) have been addressed for this release.

#### *Thanks!*

The Groovy-Eclipse team would like to thank Nieraj Singh for his excellent work on the new Groovy-Eclipse Quick Fix feature. We'd also like to thank the community for continuing to provide feedback, raising issues, and for telling us what you need. Keep it up! You are what keeps this project alive.

#### *What's next?*

We are planning the 2.1.2 release for March 2011. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#).

### **Groovy-Eclipse 2.1.2 New and Noteworthy**

You can use the following update site to install this release.

**i Groovy-Eclipse update site for Eclipse 3.6**

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

And a zipped version of the update site is available at:

**i Zipped Groovy-Eclipse update site for Eclipse 3.6**

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.1.2.xx-20110310-1500-e36.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

Snapshot versions targeting Eclipse 3.7 are available. Our 3.7 support is in beta. Although largely working, some functionality may be missing or buggy. You can access the 3.7 update site here:

**i Groovy-Eclipse snapshot update site targeting Eclipse 3.7**

<http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e3.7/>

**Outline**

- [Find occurrences in groovy files \(CTRL-Shift-U\)](#)
- [Mark occurrences](#)
- [Inline renaming](#)
- [Improved syntax highlighting](#)
- [Better type inferencing for closures inside DGM calls](#)
- [Basic support for Groovy++](#)
- [Timers for AST Transform operations now shown in Groovy Event Console](#)
- [Now includes Groovy 1.7.8](#)
- [Better ordering for Quick fixes and content assist](#)
- [Type inferencing for getAt](#)
- [Better unicode support](#)
- [Performance improvements](#)
- [Optional non-jar locking behaviour on Windows](#)
- [Headless Groovy-Eclipse feature](#)
- [Compatibility](#)
- [Bug fixes](#)
- [What's next?](#)

**Find occurrences in groovy files (CTRL-Shift-U)**

Typing CTRL-Shift-U (or CMD-Shift-U on Macs) in a Groovy editor when selecting an identifier will search for all occurrences of that identifier:

```
1 class Result {
2     def val
3     def doOperation = { val *= it }
4 }
5 def val = 9
6 def res = new Result()
7 val = res.doOperation(val)
8 print res.val
9
```

Search 🔍

'val' - 3 occurrences in 'Operation.groovy'

- 1: def val
- 2: def doOperation = { val \*= it }
- 7: print res.val

Above, notice how only references to the `val` property are highlighted. Other references with the same name are not highlighted.

#### Mark occurrences

If the feature is enabled in your workspace for Java Editors, then when selecting an identifier all references to that identifier will be marked (no need to explicitly press CTRL+Shift+U, but the results will not appear in the search view). The coloring is slightly different from find occurrences, as you can see in the following screenshot:

```
1 class Result {
2     def val
3     def doOperation = { val *= it }
4 }
5 def val = 9
6 def res = new Result()
7 val = res.doOperation(val)
8 print res.val
9
```

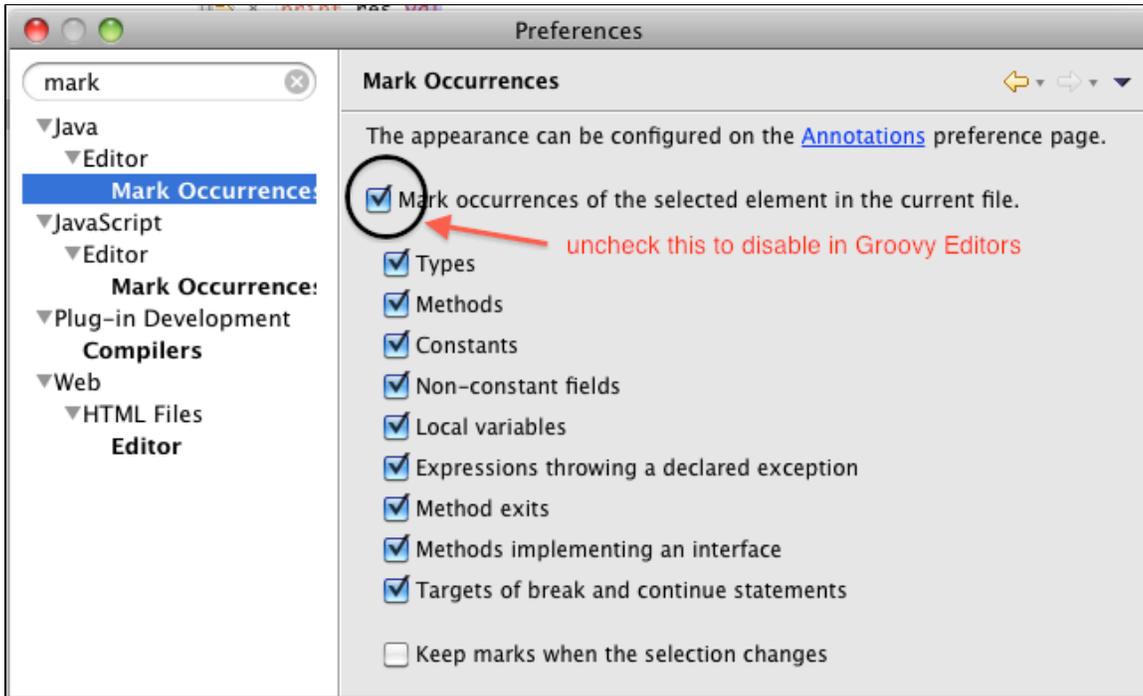
Search 🔍

'val' - 3 occurrences in 'Operation.groovy'

- 1: def val
- 2: def doOperation = { val \*= it }
- 7: print res.val

In the screenshot above, a reference to the `doOperation` closure field currently has the caret and the declaration plus all references are selected. This does not affect the previous search results as you can see.

This behavior is similar to that of the Java editor. In fact, mark occurrences in Groovy editors can be configured in the Java Editor mark occurrences preference page:



Note that, the Groovy editor does not support all the variants of mark occurrences that the Java editor does. It is either enabled completely or disabled completely, which is toggled by selecting *Mark occurrences of the selected element in the current file*.

#### Inline renaming

The Groovy editor now supports inline renaming, just like the Java editor. If the preference is set, all calls to *Rename Refactor* inside the editor will rename the variables inline:

```
class Result {
    def val
    def doOperation = { val *= it }
}
def val = 9
def res = new Result()
val = res.doOperation(val)
print res.val
```

Enter new name, press ↵ to refactor ▾

#### Improved syntax highlighting

Deprecated declarations and references are now highlighted with a strikethrough:

```
class Result {
    @Deprecated
    def val
    def doOperation = { val *= it }
}
def val = 9
def res = new Result()
val = res.doOperation(val)
print res.val
```

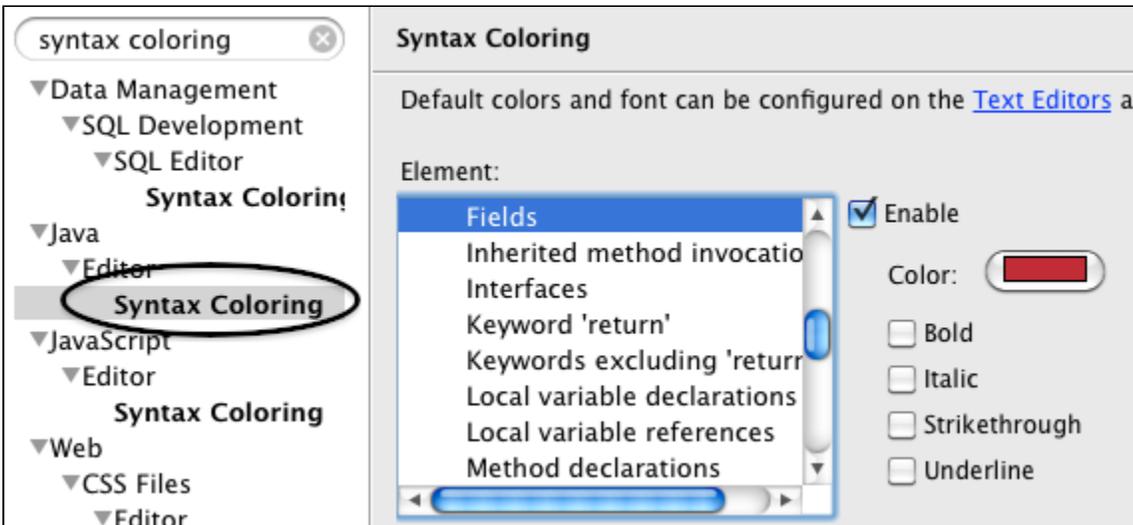
Field and method references are highlighted too, if the preference is set:

```

class Result {
    def val
    def doOperation () { val *= it }
}
def val = 9
def res = new Result()
val = res.doOperation(val)
print res.val

```

Method, field, and deprecation highlighting preferences are borrowed from Java and so they can be configured on the Java syntax highlighting page:



Additionally, we have reworked the Groovy Editor preferences page so that there are more syntax highlighting options. Now, it is possible to highlight the `return` keyword separately as well as operators:

```

{ return val *= it }

```

Yes. Your Groovy editor can now be made to be very pretty. Explore the Preferences -> Groovy -> Editor preferences page for a full list of what can be changed and more options for customization.

**Better type inferencing for closures inside DGM calls**

Standard Default Groovy Method calls like `each`, `collect`, `find`, etc are recognized by the inferencing engine and their associated closure parameter type is inferred from the type of the collection being iterated over (if it is known). For example, in the following screenshot the collection is of type `List<String>`, and so we can infer that the type of `it` is `String`:

```

def vals = [ 'a', 'b', 'c' ]
vals.each { print it.substring(0) }

```

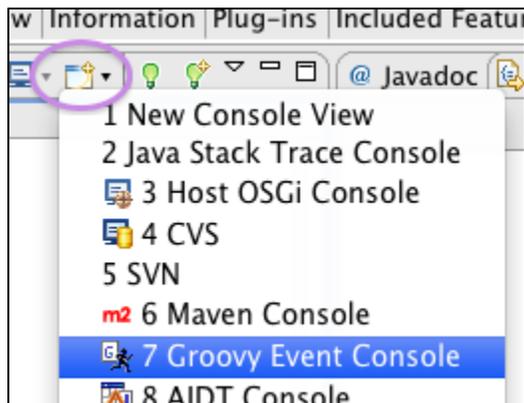
String it - Operation.run()

**Basic support for Groovy++**

This version of Groovy-Eclipse allows usage of the Groovy++ Ast transforms. Simply grab a version of groovy++ from [the Groovy++ project page](#), for example: [groovypp-0.4.170.jar](#) and place the jar on your project classpath. With that in place you can start using static resolution by marking your code with `@Typed`. For examples see the [Groovy++ documentation](#).

**Timers for AST Transform operations now shown in Groovy Event Console**

The *Groovy Event Console* is available from the console view. To activate it, click on the *Open Console* button:



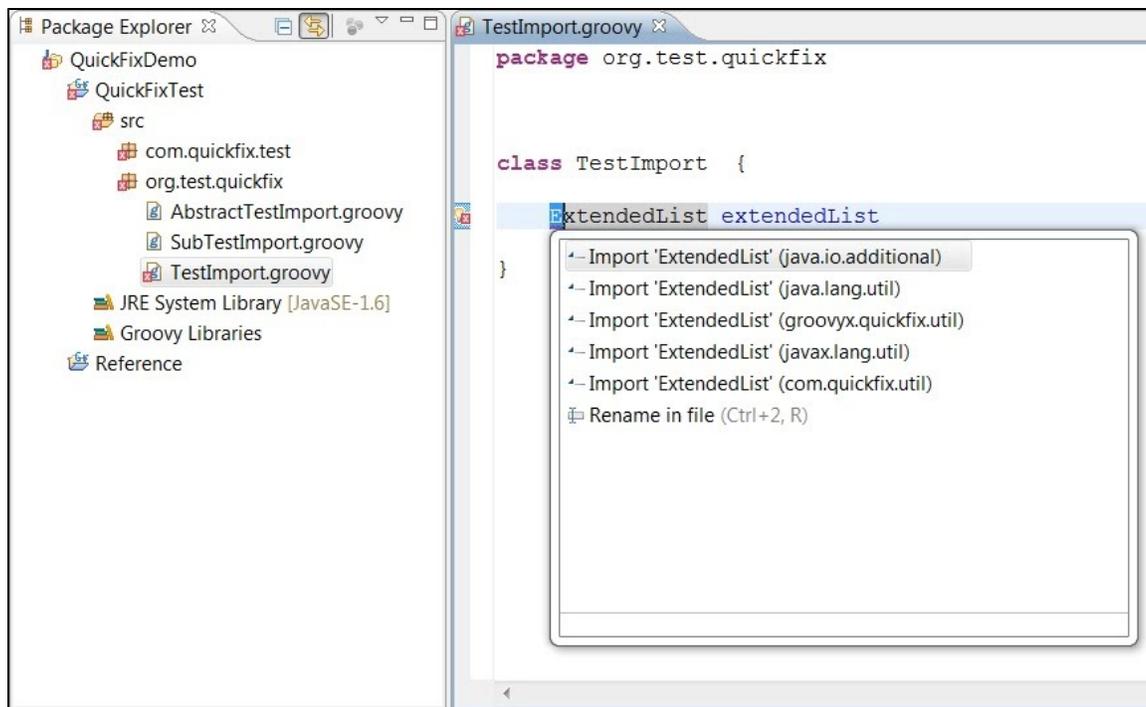
Now, interesting Groovy-Eclipse events will be logged to the console view, including information about AST Transforms. Global AST transforms will only be shown if they take more than 0ms to run, to try and reduce unnecessary entries in the console. **Please note:** the event console is meant for debugging and diagnostics. It may degrade the performance of Groovy-Eclipse if it is open.

**Now includes Groovy 1.7.8**

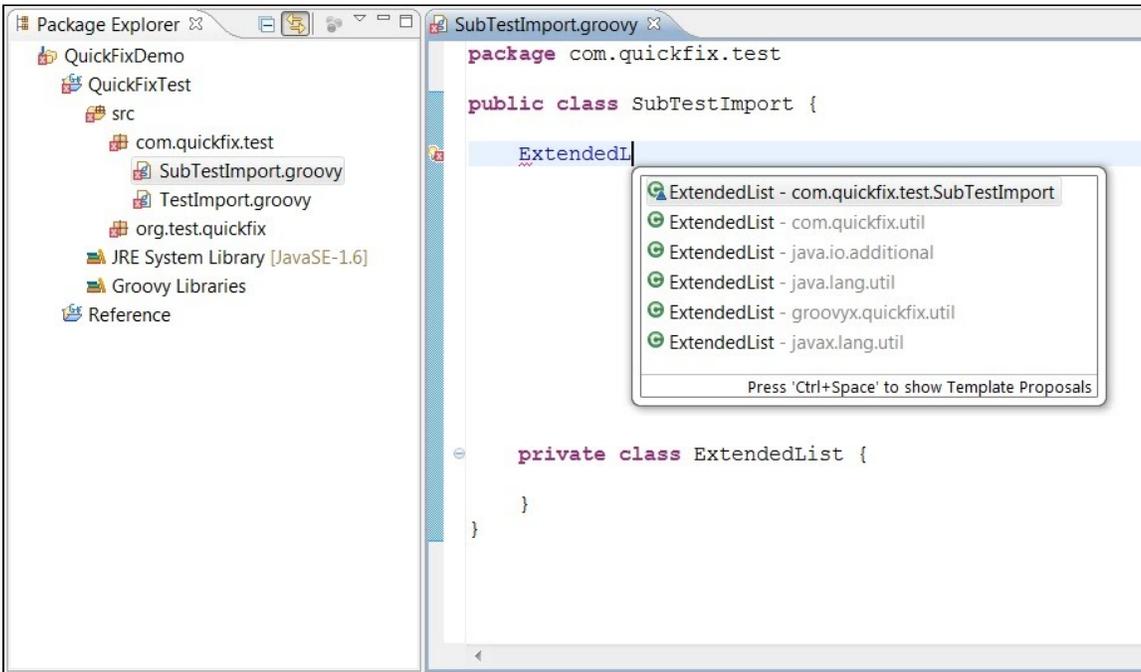
The Groovy compiler for Groovy-Eclipse is 1.7.8. We will be releasing 1.8 support shortly.

**Better ordering for Quick fixes and content assist**

Suggested types to import in both quick fix and content assist are now ordered based on various criteria so that more common types will be found towards the top of the list. Types imported from `java.*`, `groovy.*`, `groovyx.*`, and `javax.*` have higher priority, in that order, than types from other packages as seen in this screen shot where the type `ExtendedList` is being imported:



However, if importing types from similarly named packages as well as types declared in the same top level type like a private nested type, the private nested type has higher precedence, followed by public types from similarly named packages, followed by types from `java.*`, `groovy.*`, `groovyx.*`, and `javax.*`, and finally types from other packages. The following screen shows the private type `ExtendedList` appearing first in the content assist suggestion, followed by a type from a similarly named package, followed by the remaining suggestions:



#### **Type inferencing for `getAt`**

Custom `getAt(String)` methods will now be used to feed the inferencing engine when using `[]`. For example:

```
class MyTree {
    String name
    MyTree[] branches
    MyTree getAt(String branchName) {
        branches.find { it.name == branchName }
    }
    static MyTree loadTree() {
        // do something interesting
    }
}

def tree = MyTree.loadTree()
tree["is Groovy"].name
```

String MyTree.name

In the preceding code, notice how the reference to the `name` field is correctly resolved to be coming from the `MyTree` class. The inferencing engine realizes that the use of the `[]` corresponds to an invocation of the `getAt` method.

#### **Better unicode support**

Unicode escape sequences (e.g., `\uE096`) now have correct offsets and lengths in the parse tree, so you can use unicode escape sequences freely in your Groovy code and source locations in your file will not be corrupted.

### *Performance improvements*

We have made several performance enhancements for the 2.1.2 release, below are the two most notable ones. They primarily target content assist.

## **Code assist performance has been improved**

Type resolution is now lazier. During code assist we don't actively chase down (resolve) property accessor type references unless necessary.

## **Speedier type inferencing**

Where appropriate, the inferencing engine will not visit the entire file to determine the type of a single expression. Rather, it will prune the branches of the AST and only visit the method or field declaration that contains the expression to be inferred. This has a noticeable affect on content assist invocation speeds.

### *Optional non-jar locking behaviour on Windows*

On windows there is now an option to try and avoid Groovy-Eclipse locking jars and preventing them from being updated (or removed, in the case of attempting to uninstall a plugin in a grails project). Specify the system property `-Dgreclipse.nonlocking=true` when starting Eclipse/STS to activate the use of non locking classloaders. If no problems come up over the next few releases this may be made the default mode of operation. Locks should also be released (regardless of that system property setting) if the offending project is closed.

### *Headless Groovy-Eclipse feature*

Those who wish to install Groovy-Eclipse without any UI components can do so from the update site. The feature id is `org.codehaus.groovy.headless.feature`. This is primarily useful for those who are performing headless builds using [PDE](#) or [Buckminster](#). Note that this feature is *uncategorized* in the update site, so under normal circumstances you will not see it in Eclipse's Update Manager. However, if you *uncheck* "Group items by category", you will see the feature.

For more information, see [GRECLIPSE-716](#).

### *Compatibility*

Groovy-Eclipse 2.1.2 includes Groovy 1.7.8 and is installable on STS 2.6.0, Eclipse 3.6.0, 3.6.1, or 3.6.2. Snapshot builds targeting Eclipse 3.7 are available as well. See above for the update site.

### *Bug fixes*

Over [50 issues](#) have been fixed for this release.

### *What's next?*

The 2.2.0 final release is scheduled for the end of June, around the time that Eclipse 3.7 will be released. We will consider a 2.1.3 release that includes Groovy 1.8 support.

We appreciate community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, but we can only do so when we hear from the community.

## **DSL Descriptors for Groovy-Eclipse**

This document describes Groovy-Eclipse's DSL descriptors. DSL descriptors (or DSLDs) are groovy files that describe domain-specific extensions to Groovy-Eclipse's inferencing engine and content assist. They allow end-users to script extensibility to Groovy-Eclipse's editing support. This document is up to date for Groovy-Eclipse 2.5.2.

- [The Problem](#)
- [New Features in Groovy-Eclipse 2.5.2](#)
  - [log expressions](#)
  - [supportsVersion and assertVersion](#)
  - [delegatesTo changes](#)
  - [Use contribute instead of accept](#)
  - [namedParams and optionalParams](#)
  - [New pointcuts](#)
  - [isDeprecated](#)
- [The infrastructure](#)
- [An Overview](#)
  - [Under the hood](#)
  - [More formally](#)
- [A deep dive](#)
  - [Pointcuts](#)
    - [Semantic pointcuts](#)
    - [Filtering pointcuts](#)
    - [Lexical pointcuts](#)
    - [Structural pointcuts](#)
    - [Combinatorial pointcuts](#)
    - [Binding pointcuts](#)
    - [User-defined pointcuts](#)
    - [Other pointcuts](#)
    - [Assigning pointcuts to variables](#)
  - [Contribution Blocks](#)
  - [Binding](#)
  - [SupportsVersion and AssertVersion](#)
  - [Register pointcut](#)
  - [Wormhole](#)
- [Some larger examples](#)

### ***The Problem***

The Groovy language is an excellent platform for the easy creation of domain specific languages (DSLs). However, these DSLs are not directly supported by the Groovy editor. When DSLs are used heavily, standard IDE features like content assist, search, hovers, and navigation lose their value. For a while now, it has been possible to write an Eclipse plugin to extend Groovy-Eclipse, which requires specific knowledge of the Eclipse APIs. This is no longer necessary. Creating a DSL descriptor for a DSL is a way to make DSLs become first class citizens of Groovy-Eclipse.

As an example, Joachim Baumann describes how to implement [a simple DSL for working with distances](#). Using this DSL, you can write things like this to calculate the total distance travelled:

```
3.m + 2.yd + 2.mi - 1.km
```

This is a simple and expressive DSL, but when you type this into a Groovy Editor in Groovy-Eclipse:

```
3.m + 2.yd + 2.ml - 1.km
```

You see underlines and no hovers. Using a DSLD, it is possible to teach the editor some of the semantics behind these custom DSLs. It is also possible to provide documentation hovers:

```
3.m + 2.yd + 2.mi - 1.km
```

**Distance Number.m**

A Meter from <http://joesgroovyblog.blogspot.com/2007/09/and-miles-to-go-before-i-sleep.html>

This document will describe how to extend Groovy-Eclipse through DSLDs.

#### *New Features in Groovy-Eclipse 2.5.2*

In Groovy-Eclipse 2.5.2, we have introduced quite a few new features. A brief description of these new features are in this section, and they are described in more detail below, mixed in with the rest of the documentation

## log expressions

For debugging and logging inside of DSLDs, there is now the `log` method. Like `println`, the `log` method takes a (g)string. Instead of printing to `sysout`, it prints the string to the Groovy console if it is open. `log` is a no-op method if the console is closed.

## supportsVersion and assertVersion

Previously, `supportsVersion` prevented the script from running if the version test had failed. Now, it returns a boolean and is therefore suitable for use inside of `if` statements so that different functionality can be run depending on the current version. `assertVersion` now behaves the way that `supportsVersion` used to. It will prevent the script from being applied.

## delegatesTo changes

The `delegatesTo` operation has become more flexible. It now accepts a set of named arguments, including `type`, `asCategory`, `useNamed`, `except`, and `isDeprecated`.

Here is an example:

```
delegatesTo type: "com.foo.MyCategory",
asCategory: true, except: ["ignoredMethodA",
"ignoredMethodB"]
```

This snippet will add all methods from `MyCategory` to the current type as *category* methods. In other words, `MyCategory` is treated like a category. Only static methods whose first argument type matches the current type will be included in the delegation. And the `except` clause will exclude all methods named in the list. This is useful for when these particular methods are handle by some other means.

The old style of calling `delegatesTo` with a single type name is still applicable.

## Use `contribute` instead of `accept`

The use of `accept` has been deprecated and is replaced by `contribute`, with a slightly different syntax. Here are some examples:

New:

```
contribute(currentType("com.foo.Sumthin")) {  
    // and contributions here.  
}
```

Old:

```
currentType("com.foo.Sumthin").accept {  
    // and contributions here.  
}
```

The `accept` style of connecting a pointcut to a contribution group is still available, but may go away in future versions.

## namedParams and optionalParams

A `method` contribution now recognizes three kinds of parameters: regular, named and optional. Regular and named parameters appear in content assist, and optional does not. Named and optional parameters are inserted with the argument name prefixed.

Here is an example:

```
method name:"meth", params: [first:String,  
second: "foo.MyObj",  
  
namedParams:[third:Integer, fourth:Long],  
  
optionalParams:[fifth:"foo.Optional",  
sixth:"foo.OtherOptional"]
```

Note that this new feature deprecates the `useNamedArgs` parameter.

## New pointcuts

Quite a few new pointcuts have been introduced. Some are fairly restrictive in the locations they can be used.

- **hasArgument** : available inside of `enclosingMethod` and `enclosingCall` to match on declared arguments. Useful to combine with the `type`, `name`, and `value` pointcuts.
- **hasAttribute** : available inside of `annotatedBy` to match on attributes. Useful to combine with the `type`, `name`, and `value` pointcuts.
- **type** : available to match on the declared type of method arguments, annotation parameters, field types, and method (return) types.
- **value** : available to match on the value of an annotation
- **isThisType** : this pointcut supersedes the `currentTypeIsEnclosingType` pointcut, which is now deprecated.

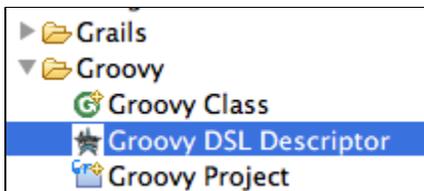
## isDeprecated

It is now possible to mark contributions as deprecated. In the UI, these contributions will be displayed with a line running through them (just like other kinds of deprecated references).

There is a new `isDeprecated` argument available to `method`, `property`, and `delegatesTo` contributions.

### *The infrastructure*

To create a DSLD file inside of Eclipse, go to *File -> New -> Groovy DSL Descriptor*.

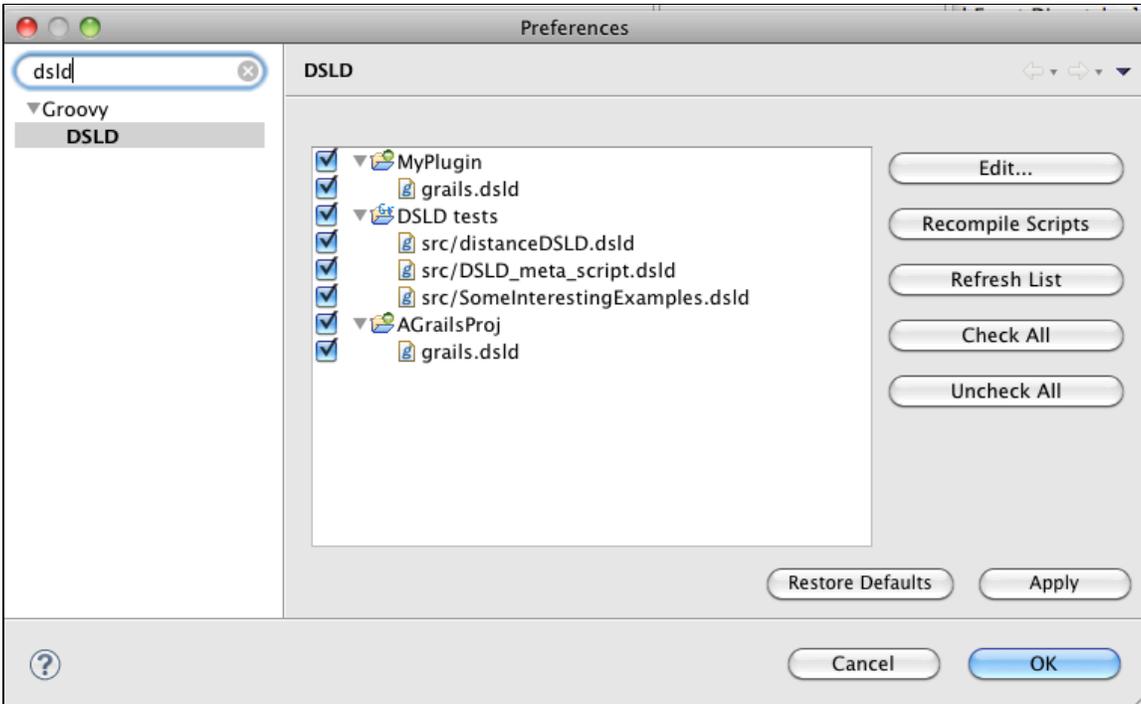


Choose a location if a suitable one has not been selected by default. In order to get content assist and hover support for your DSLD, it is recommended to choose a location inside of a source folder, but this is not required.

Groovy-Eclipse's DSL support will process all DSLD files that are in the project as well as all DSLD files that are on

the project's classpath in a package named DSLD (this could be in a jar file an external class folder, or in coming from another project). Note that each project has its own set of DSLD files and they are not shared unless there is an explicit dependency and DSLD files are placed in the `dsld` package. In the future, we will have a single location where global scripts can reside, but this is not yet implemented.

To see what scripts are currently available for each project, you can go to Preferences -> Groovy -> DSLD. Here, you can see a list of all projects and their DSLDs:



Each file can be disabled by deselecting it from the tree viewer. Also, the set of DSLD files can be refreshed (i.e., the current set of files are dropped and the entire workspace is searched again for DSLD files) and all scripts can be recompiled from this page using the buttons on the right. In the normal workflow, these last two options should not be necessary to use, but if you are seeing problems with your DSLDs, then you may want to try use them.

For development work on DSLDs themselves, it is strongly recommended that you place them in a source folder. This will give you editing features like hover support and content assist, and additionally, many syntax errors will appear in the Problems view. If you want to consume a DSLD in your project, then it is not necessary to place them in a source folder.

Lastly, when doing DSLD work, it is recommended that you open up both the [Groovy Event Trace Console](#) and the Eclipse Error Log. If there are syntax errors or other kinds of problems with your script, they will be printed to the Groovy Event Trace Console. If there are problems with the DSL infrastructure itself, there will be entries in the error log (these exceptions are likely bugs in Groovy-Eclipse and should be reported to the mailing list or in the issue tracker).

✓ **Reminder**

When doing DSLD work, it is recommended that you place your DSLDs in a source folder. Also, you will benefit by opening up both the [Groovy Event Trace console](#) and the **Eclipse Error Log view**.

However, when consuming existing DSLDs, they do not need to be in a source folder. They can be in any folder in the project. Alternatively, DSLDs can live in jars or class folders in the `/dsl` package. If this jar or class folder is on the classpath project, then these DSLDs will be used by the project (but only if they are in the `/dsl` package).

*An Overview*

The DSLD language is an [aspect-oriented domain specific language](#). The main components of a DSLD script are:

**pointcuts**

A query that describes a set of Groovy expressions in a program, for example the following is a simple pointcut that "matches" for all expressions whose the type is a subtype of Number.

```
currentType(subType(Number))
```

## contribution blocks

A code block that describes the extra properties and methods available when a pointcut matches. Contribution blocks are analogous to [advice in AspectJ](#). A pointcut is ignored unless it is associated with a contribution block (and vice versa). Here is a simple contribution block:

```
{
    property name:"cm",
    type:"Distance", doc:
        "" "A cm from
        <a
href=\"http://joesgroovyblog.blogspot.com/2007/09/and-miles-to-go-before-i-sleep.html\">
http://joesgroovyblog.blogspot.com/2007/09/and-miles-to-go-before-i-sleep.html</a>""
}
```

This block adds a new property called `cm` with a type of `Distance`. The javadoc is specified in the `GString` and can use standard HTML tags. However, without being associated with a pointcut, this contribution block will be used and will not add the method to any type.

To associate a pointcut with a contribution block, you call the `contribute` method. For example, this adds the `cm` property to all `Numbers`:

```
contribute(currentType(subType(Number))) {
    property name:"cm", type:"Distance", doc:
        "" "A cm from
        <a
href=\"http://joesgroovyblog.blogspot.com/2007/09/and-miles-to-go-before-i-sleep.html\">
http://joesgroovyblog.blogspot.com/2007/09/and-miles-to-go-before-i-sleep.html</a>""
}
```

The `contribute` method takes a pointcut and a closure. Inside that closure, you can specify a set of methods or properties that should be added to the type of the expression being analyzed.

When this script is copied into a Groovy project, `cm` will be included in content assist of all `Numbers` and their subclasses. Also, references to `cm` on `Numbers` will not be underlined as we showed in the example above. Finally, hovering over these references will display the doc defined in the contribution block.

## Under the hood

The pointcuts and contribution blocks are used by Groovy-Eclipse's [inferencing engine](#). The inferencing engine walks the AST and delegates the calculation of expression types to *type lookup* objects. The type lookups have some capacity for sharing state. DSLD is implemented as a type lookup.

There are two phases for DSLD files.

1. *The initial compile and processing*: at startup or whenever a DSLD file is added or changed, the file is (re-)compiled and the resulting script is executed. Calls to `contribute` will store the pointcut and its associated contribution block in a table per project.
2. *Invocation by the inferencing engine*: for each expression in the file, each pointcut stored by the project is evaluated. If any pointcut evaluates to true, then that pointcut's contribution block(s) is evaluated and all contributions are added to the current type being evaluated.

## More formally

We previously showed the core of the DSLD language, which we can now describe slightly more formally in terms of a join point model. As defined by Masuhara, et al, a [Join Point model](#) requires 3 things:

1. A set of reference points that can be referred to; these are the *join points*.
2. A means of identifying (a set of) join points; these are the pointcuts.
3. A way to specify semantics at a particular set of join points; such as through advice, intertype declarations, or contribution blocks.

The DSLD language uses a syntactic join point model, where the join points are expression nodes in a Groovy abstract syntax tree (AST). DSLD affects the edit-time semantics of a program by contributing properties and methods to types in the IDE. This is different from most other AOP languages that affect a program's the semantics at runtime or compile time.

The individual components of a join point model are matched to concepts in the DSLD language as follows:

1. The join points are expression AST nodes in a Groovy file
2. DSLD defines a pointcut language to specify a sub-set of expression node join points
3. And Contribution blocks augment the semantics of Groovy-Eclipse's [inferencing engine](#) to be enhanced with new type suggestions

### *A deep dive*

In this section, we go into more detail on the DSLD language and document all of its features.

## Pointcuts

There are a fixed set of standard pointcuts which can be composed using '&' or '!' and negated using '!~'. All pointcuts define their own javadoc and will be displayed when hovered over (but only if the DSLD meta script is available in the current project). Pointcuts are generally self documenting using content assist and hovers inside of the DSLD file, but here we describe the more important ones.

The parts of a pointcut are as follows:

- *Arguments*: in the DSLD script the text inside of a pointcut's '(' )' are the arguments. Most pointcuts take one or zero arguments.
- *Evaluation arguments*: during evaluation, a pointcut is passed a value to match on. This value is called the evaluation argument. For example, `currentType(annotatedBy(Delegate))`, `currentType` is the top-level pointcut and is always passed in the current type as a `ClassNode` object. `annotatedBy` is passed in what is matched by its surrounding pointcut, in this case it is `currentType`.
- *Return values*: the value returned when a match happens. This is typically, a collection of AST nodes. These return values can be bound to names that are available in a contribution block.

When a match is found, all pointcuts return the objects that are matched in a `java.util.Collection` object. For consistency, a collection is returned even when a single object is matched.

## Semantic pointcuts

These are pointcuts that are directly dependent on the inferencing engine. All semantic pointcuts are top level pointcuts only and expect the current type as a class node as an evaluation argument.

### **currentType**

Matches if the current type matches the pointcut argument. The type erasure is used, so the match will be the same if either `List<String>` or `List<Class>` are passed in.

#### *Pointcut Arguments*

Expects a string, `Class`, or `ClassNode` corresponding to the type to match against. Alternatively, a filtering pointcut can be used.

#### *Evaluation arguments*

Always a top-level pointcut.

#### *Return values*

The current type as a singleton `ClassNode` collection.

#### *Example*

```
// matches always  
currentType()
```

```
// matches if the current type is a GString  
currentType(GString)
```

```
// matches if the current type is a sub type  
of java.lang.Number  
currentType ( subType ( Number ) )
```

### **isThisType**

Matches when the current type is the same as the enclosing type. In other words, the current type is *this*.

*Pointcut Arguments*

None

*Evaluation arguments*

Always a top-level pointcut.

*Return values*

The current type as a singleton `ClassNode` collection.

*Example*

```
isThisType ( )
```

### **currentTypelsEnclosingType**

(Deprecated, use `isThisType` instead)

Matches when the current type is the same as the enclosing type.

*Pointcut Arguments*

None

*Evaluation arguments*

Always a top-level pointcut.

*Return values*

The current type as a singleton `ClassNode` collection.

*Example*

```
currentTypeIsEnclosingType()
```

In the following code, this pointcut will match on `this` and `x()`, but not `substring`:

```
class Foo {  
    def x() { "1" }  
    def y() {  
        this.x().substring(0)  
    }  
}
```

### **sourceFolderOfCurrentType**

Matches when the source folder that the current type is declared in the source folder of the argument. This pointcut does not match for binary types.

#### *Pointcut Arguments*

A string specifying the source folder to match on.

#### *Evaluation arguments*

Always a top-level pointcut.

#### *Return values*

The source folder as a singleton string.

#### *Example*

Matches when the current type is defined in `src/main/groovy`:

```
sourceFolderOfCurrentType("src/main/groovy")
```

Matches on all source types:

```
sourceFolderOfCurrentType()
```

# Filtering pointcuts

These are pointcuts that are typically used as arguments to other pointcuts and can further filter a result to refine what exactly matches. They can be used as a top-level pointcut, but in this case they are implicitly passed in the current type.

## field/method/property

Matches on a set of fields, methods, or properties.

### *Pointcut Arguments*

A string corresponding to the field/method/property name, or another filtering pointcut to further refine the match.

### *Evaluation arguments*

These pointcuts expect either a `ClassNode` or a `Collection` of class members. If passed in a `ClassNode`, then all of the fields/methods/properties will be extracted to match against.

### *Return values*

Returns a collection of fields/methods/properties that have the characteristics specified by pointcut argument.

### *Example*

Matches if the current type has any fields:

```
currentType(fields())
```

Matches if any method in the current type is named "myMethod". Returns all matched methods as a collection:

```
currentType(methods("myMethod"))
```

Matches if the current type is a sub type of a class that defines a method "myMethod":

```
currentType(subType(methods("myMethod")))
```

Note that `currentType` is optional and if omitted, then the current type is implicitly passed in. So, the following are identical pointcuts to the previous

```
fields()  
methods("myMethod")  
subType(methods("myMethod"))
```

### **subType**

Matches if the type passed in as an evaluation argument is a sub type of the pointcut argument.

#### *Pointcut Arguments*

A string, Class, or ClassNode corresponding to the type that the current type will be checked against.

#### *Evaluation arguments*

A ClassNode, collection of ClassNodes to match against. Alternatively, a MethodNode, FieldNode, PropertyNode, or collection of them can be matched against. In this case, the declaring type of the declarations will be used.

#### *Return values*

A collection of super-types that match the conditions described in the pointcut argument

#### *Example*

Matches when the current type is a sub type of something annotated by @Delegate:

```
currentType(subType(annotatedBy(Delegate)))  
// or  
subType(annotatedBy(Delegate))
```

Matches when the enclosing type is a subtype of TestCase:

```
enclosingClass(subType("junit.framework.TestCase"))
```

### **isStatic/isPublic/isPrivate/isSynchronized/isStatic/isFinal**

Matches when one or more of the evaluation arguments is declared with the given modifier.

#### *Pointcut Arguments*

None.

### *Evaluation arguments*

Expecting a collection of AnnotatedNodes.

### *Return values*

A subset of the evaluation arguments that all have the given modifier.

### *Example*

Matches if the current type has any static fields:

```
fields(isStatic())
```

### **annotatedBy**

Matches when the evaluation arguments correspond to the pointcut argument.

### *Pointcut Arguments*

A string, Class, or ClassNode corresponding to the annotation to match. Alternatively, another pointcut can be used to further constrain the match.

### *Evaluation arguments*

A collection of AnnotatedNodes.

### *Return values*

A collection of all AnnotationNodes matched inside of the evaluation arguments.

### *Example*

Matches when the current type is annotated by @Deprecated:

```
annotatedBy("java.lang.Deprecated")
```

Matches when the current type has any field with any annotation

```
fields(annotatedBy())
```

Matches when the current type is annotated by an annotation with the @Retention meta-annotation:

```
annotatedBy(annotatedBy("java.lang.annotation.Retention"))
```

### **hasAttribute**

Matches when an annotation passed in has an attribute with the appropriate name, type, or value.

#### *Pointcut Arguments*

A string corresponding to the name of the annotation to match, or a combination of name, type, and value pointcuts to match against.

#### *Evaluation arguments*

A collection of AnnotationNodes.

#### *Return values*

The value or values of the attribute(s) that are matched as a collection. Values that are constants will be reified so that the constants will be available to the DSLD as primitives and strings.

#### *Example*

Matches if a field in the current type is annotated by an annotation with an attribute called value.

```
currentType(field(annotatedBy(hasAttribute("value"))))
```

Matches if a field in the current type is annotated by an annotation with an attribute called value and this attribute's value is true.

```
currentType(field(annotatedBy(hasAttribute(name("value") & value(true)))))
```

Matches if a field in the current type is annotated by an annotation with an attribute called value and this attribute's type is java.lang.Boolean. Additionally, the value of the attribute is bound to val.

```
currentType(field(annotatedBy(hasAttribute(name("value") & type(Boolean) & bind(val : value())))))
```

### type

Matches on the type of a field, argument, parameter, annotation attribute, or method return type. See `hasAttribute` and `hasArgument` for examples.

### declaringType

Matches on the declaring type of a field, method, or property. See `enclosingCall` and `hasArgument` for an example.

### value

Matches on the value of an argument or annotation attribute. See `hasAttribute` and `hasArgument` for examples. If the matched value is bound to a name, then the value is *reified*. This means that the value can be used inside of the DSLD as a primitive or a string. For example:

```
contribute(enclosingCall(hasArgument(name("f  
lar") & bind(vals : value())))) {  
  vals.each { val ->  
    if (val > 9) {  
      log "val is greater than 9"  
    } else {  
      log "val is less than 9"  
    }  
  }  
}
```

In the example above, the bound values can be directly compared to primitives or strings in the DSLD.

### name

Matches on the fully qualified name of an element.

#### *Pointcut Arguments*

A string corresponding to the name to match

### *Evaluation arguments*

A collection of any objects can be passed in. The pointcut argument will be matched against the name of the AST node if a method, field, property, or class declaration. Else, `toString()` will be called on the evaluation argument.

### *Return values*

The name or names that were matched as a collection.

### *Example*

Matches if the name of the current type is "p.Bar" and it has a method named "foo":

```
currentType(name("p.Bar") & methods("foo"))
```

## Lexical pointcuts

These are pointcuts that depend on the lexical structure of the surrounding AST.

### **enclosingScript/enclosingClass**

Matches when the enclosing type is a script/class.

### *Pointcut Arguments*

A string, Class, or ClassNode corresponding to the type name to match on. Alternatively, a pointcut argument can be used.

### *Evaluation arguments*

These pointcuts are top-level only.

### *Return values*

The type that was matched.

### *Example*

Matches when inferring inside of a script:

```
enclosingScript()
```

Matches when inferring inside of a script named "pack.MyScript":

```
enclosingScript("pack.MyScript")
```

Matches when inferring inside of a class that has a @Singleton annotation

```
enclosingClass(annotatedBy(Singleton))
```

### **enclosingField/enclosingMethod**

Matches when enclosed by a field or method of the given characteristics. A code block is *enclosed* by a field if it is inside of a field initializer. For example, the print statement is *enclosed* by the x field:

```
class F {  
    def x = { print "I am enclosed!" }  
}
```

### *Pointcut Arguments*

A string, Class, or ClassNode corresponding to the type name to match on. Alternatively, a pointcut argument can be used.

### *Evaluation arguments*

These pointcuts are top-level pointcuts only.

### *Return values*

A singleton collection of the enclosing method/field if the current location matches.

### *Example*

Matches when the enclosing method or field has the given annotation:

```
enclosingMethod(annotatedBy("com.foo.MyAnnotation")) |  
enclosingField(annotatedBy("com.foo.MyAnnotation"))
```

### **enclosingCall**

Matches on the enclosing method call. The enclosing method call is the method call of which the current expression is an argument. There are multiple enclosing calls if the current expression is nested inside of multiple method calls (e.g., foo(bar(arg))).

### *Pointcut Arguments*

Inner pointcuts are required. The inner pointcuts accepted are: `name` to match on the method name, `type` to match on the return type, `declaringType` to match on the method's declaring type, and `hasArgument` to match on the method call's arguments.

#### *Evaluation arguments*

This pointcut is top-level only.

#### *Return values*

The method call expression as a Groovy AST node.

#### *Example*

Matches when the enclosing method call is named "myMethod" and the declaring type is Bar and the call has a named argument of either "first" or "second":

```
enclosingCall (name ("myMethod") &  
declaringType ("com.foo.Bar") &  
(hasArgument ("first") |  
hasArgument ("second")))
```

#### **hasArgument**

Matches on an argument to the enclosing method call or the enclosing method declaration. Note that for method calls, this pointcut is only applicable on named arguments. Also, when this pointcut is used in a method declaration, the `value` inner pointcut cannot be used.

#### *Pointcut Arguments*

If a string is passed as the pointcut argument, then the match is against the name of the parameter. Alternatively, any of `name`, `value`, and `type` can be used as enclosed pointcuts.

#### *Evaluation arguments*

This pointcut must be used inside of `enclosingCall` or `enclosingMethod`.

#### *Return values*

The *value* of a named argument if matching on a named argument in a method call, otherwise the name of the parameter if matching on a method declaration.

#### *Example*

Matches when the enclosing method call is named "myMethod" and the declaring type is Bar and the call has a named argument of first whose type is String. Additionally, binds *val* to a the value of the named argument.

```
enclosingCall(name("myMethod") &  
declaringType("com.foo.Bar") &  
(hasArgument(name("first") & type(String) &  
bind( val : value()))))
```

Matches when the enclosing method declaration is named "myMethod" and the declaring type is Bar and the call has a named argument of first whose type is String. Note that here, the `value` pointcut is not applicable since there is no value to match on.

```
enclosingCall(name("myMethod") &  
declaringType("com.foo.Bar") &  
(hasArgument(name("first") & type(String))))
```

#### **enclosingCallName/enclosingCallDeclaringType**

Matches when the enclosing method call has the given name or the given type. The enclosing method call is the method call of which the current expression is an argument. For example:

```
myVariable.someMethod(bar, baz) {  
    it.otherMethod(inside)  
}
```

In the code above, `bar` and `baz` are enclosed by the `someMethod` method call. Additionally, all expressions in the closure are enclosed. The reference to `inside` is enclosed by two method calls: `someMethod` and `otherMethod`.

#### *Pointcut Arguments*

`enclosingCallName` takes a string corresponding to the name of the method call. `enclosingCallDeclaringType` takes a string, `Class`, `ClassNode`, or other pointcut to describe the type to match on.

#### *Evaluation arguments*

This pointcut is top-level only.

#### *Return values*

Returns all enclosing method name/declaring types as an ordered set. The ordering is from the lexically closest call to the furthest. In the example above, the ordering would be `otherMethod` and then `someMethod`.

### *Example*

Matches when enclosed by a method call `foo` declared by `com.bar.MyClass`:

```
enclosingCallName("foo") &  
enclosingCallDeclaringType("com.bar.MyClass"  
)
```

### **enclosingClosure**

Matches when enclosed by a closure.

#### *Pointcut Arguments*

No arguments.

#### *Evaluation arguments*

This pointcut is top-level only.

#### *Return values*

An ordered collection of `ClosureExpressions` corresponding to the lexically closest closure to the furthest. For example:

```
def constraints = {  
  tryThis {  
    print "Me"  
  }  
}
```

In the previous code snippet, when inferring on the `print` method call, `enclosingClosure` would return two closure expressions: first the closure attached to `tryThis` and then the closure attached to `constraints`.

### *Example*

Here is a pointcut describing when inside of a `SwingBuilder` closure:

```
enclosingClosure() &  
enclosingCallDeclaringType("groovy.swing.SwingBuilder") & enclosingCallName("edt")
```

## Structural pointcuts

These are pointcuts that depend on the structure and naming conventions of the file system.

### **nature**

Matches on the Eclipse project nature of the current project. For

#### *Pointcut Arguments*

A string corresponding to the desired project nature. Examples are:

- Grails: `com.springsource.sts.grails.core.nature`
- GWT nature: `com.google.gwt.eclipse.core.gwtNature`

#### *Evaluation arguments*

This pointcut is top level only.

#### *Return values*

Returns the name of the nature as a singleton collection.

#### *Example*

Matches domain classes in grails projects:

```
nature("com.springsource.sts.grails.core.nature") &  
sourceFolderOfCurrentType("grails-app/domain")
```

### **fileName/fileExtension**

Matches on the full file name or on the file extension of the current file being inferred.

#### *Pointcut Arguments*

A string corresponding to the file name or extension (excluding the '.') to match.

### *Evaluation arguments*

This pointcut is top level only.

### *Return values*

Returns the name of the file that was matched.

### *Example*

Matches on gradle files:

```
fileExtension("gradle")
```

Matches on MyScript.groovy:

```
fileName("MyScript.groovy")
```

### **sourceFolderOfCurrentFile**

Matches on the source folder of the file currently being inferred. Note that this is different from `sourceFolderOfCurrentType` that will match on the current type. `sourceFolderOfCurrentFile` will return the same value for the entire file, whereas `sourceFolderOfCurrentType` will have a different value depending on the type of the current expression.

### *Pointcut Arguments*

A string corresponding to the source folder to match on.

### *Evaluation arguments*

This pointcut is top level only.

### *Return values*

The matches source folder as a singleton collection.

### *Example*

Matches when in the `src` folder:

```
sourceFolderOfCurrentFile("src")
```

## **Combinatorial pointcuts**

No need to explicitly invoke these pointcuts. They are included here for completeness and are implicitly created when using `&`, `|`, and `~` (meaning the `and`, `or`, and `not` pointcuts respectively).

### **and/or/not**

#### *Pointcut Arguments*

These are the only pointcuts that can take more than one argument. The arguments must be other pointcuts.

#### *Evaluation arguments*

These pointcuts can take any evaluation arguments. They are simply passed on to their contained pointcuts.

#### *Return values*

For `or` and `and`, a collection of all elements matched by the containing pointcuts. For `not`, this will return a singleton collection with an `Object` value if the containing pointcut does not match.

#### *Example*

It is generally good to use parens around the pointcut argument of `not`. Matches when the current type has no fields:

```
~(fields())  
// equivalent to:  
not(fields())
```

Matches when inside a deprecated class, and there is a field or method in the class named "bar":

```
enclosingClass(annotatedBy(Deprecated)) &  
(fields("bar") | methods("bar"))  
// equivalent to:  
and(enclosingClass(annotatedBy(Deprecated)),  
or(fields("bar"), methods("bar")))
```

### ✓ Things to be careful about

Due to Groovy's operator precedence rules, parens must be used around not ('~') or else the ~ will apply to the pointcut name (without the parens), instead of the pointcut expression (with the parens). You should do this:

```
(~ isPublic() ) | name("java.lang.String")
```

instead of this:

```
~ isPublic() | name("java.lang.String")
```

## Binding pointcuts

There is only one binding pointcut, called `bind`. This pointcut explicitly binds the return value to a name to make it available in a contribution block.

The explicit use of the `bind` pointcut can be omitted when it is surrounded by another pointcut. There is a bigger discussion of this pointcut below.

### **bind**

#### *Pointcut Arguments*

A single pointcut

#### *Evaluation arguments*

Any evaluation arguments. They are passed to the contained pointcut.

#### *Return values*

Returns the value returned by the contained pointcut.

#### *Example*

Binds a collection of deprecated annotations:

```
fields(bind(dep : annotatedBy(Deprecated)))
```

```
// this fully is equivalent to:
```

```
fields(dep : annotatedBy(Deprecated))
```

Binds to deprecated fields in the current type. In this case, bind is necessary:

```
bind(dep : fields(annotatedBy(Deprecated)))
```

## User-defined pointcuts

It is possible to register your own pointcuts that will be available in the current DSLD script only. See the `registerPointcut` section below.

## Other pointcuts

The DSLD language is a work in progress. Here are some pointcuts that are not yet implemented, but may be based on the needs of the community: `regex`, `instanceof`, `superType`, `enclosingType`, `enclosingEnum`, and `others`.

### Some subtleties about `java.lang.Class` references

Even though the DSLD script is being edited in the context of your project, the script is actually loaded by Groovy-Eclipse. And so, the runtime classpath of the script corresponds to Groovy-Eclipse's classpath, rather than the classpath of your project.

Consequently, you cannot reference class objects for types defined in your project. However, you can reference class objects that are available to Groovy-Eclipse. This might be confusing since the compiler will not show compile errors when types defined in your project are referenced as class objects, but it will show compile errors when Groovy-Eclipse types are referenced. This is because the Groovy-Eclipse compiler works off of the project's classpath. It is not yet aware that DSLD files will be run with a different classpath.

More specifically:

- Instead of referencing the class `MyLocalType` directly, you can reference it as a String "`com.mycompany.MyLocalType`"
- Standard JDK, GDK, and all types defined in `groovy-all` are available directly in your DSLD and will show compile errors.
- It is possible to reference types in packages beginning with `org.eclipse.jdt`. **and org.codehaus.groovy.eclipse**. if all references are fully qualified. However, this is not recommended unless you really know what you are doing.

## Assigning pointcuts to variables

Sometimes it is useful to assign pointcuts to variables. For example, here is how we might describe a Grails domain class and a controller class:

```
def grailsArtifact = { String folder ->
    sourceFolderOfCurrentType("grails-app/" +
    folder) &

    nature("com.springsource.sts.grails.core.nature") & (~enclosingScript())
}
def domainClass = grailsArtifact("domain")
def controllerClass =
grailsArtifact("controllers")
```

Notice how it is possible to use a closure so that pointcut components can be shared and parameterized.

We can use the `domainClass` pointcut above as a component in a larger pointcut that describes where the Grails constraints DSL is applicable:

```
contribute(domainClass &
    enclosingField(name("constraints") &
    isStatic()) &
    inClosure() &
    currentTypeIsEnclosingType() &
    (bind(props : properties())) { ... }
```

Let's break this down a bit:

1. The first thing to notice is that the `domainClass` reference doesn't require and parens. This is because parens have already been used when the pointcut was first declared.
2. Next, notice the `enclosingField` pointcut. This component matches when the enclosing field name is `constraints` and the field is static.
3. Usually, the `name()` pointcut is implicit and optional, but since `&` requires pointcuts on either side, we need to wrap the `"constraints"` string inside of a pointcut.
4. Next, the expression must be inside of a closure

5. The `currentTypeIsEnclosingType()` pointcut means that the type of the current expression must be the type of the enclosing class. Thus, references to `this` will match, but references to other types (such as `new String()` will not).

## Contribution Blocks

Now that we have described the pointcut language, we can delve into what happens in contribution blocks.

You have already been introduced to the following form, which adds a property to the type of the expression matched in the contributing pointcut:

```
contribute(...) {  
    property name : "cm", type: Number, doc :  
    "..."  
}
```

The full form of `property` is:

```
property name : "cm", type: "Distance",  
declaringType: Number, isStatic : false,  
    doc : "<b>Enter javadoc here</b> html is  
supported",  
    provider : "A readable name for your DSL  
(no html)"
```

A few notes:

- `name` is the only required argument
- `type` defaults to `java.lang.Object` and the argument can be of type `java.lang.String`, `java.lang.Class`, or `org.codehaus.groovy.ast.ClassNode`.
- `declaringType` defaults to the `currentType` and can accept a `String`, `Class`, or `ClassNode`
- `doc` is the javadoc that will show up in hovers and accepts html syntax
- `provider` is a human readable name for the current DSL and appears in content assist to give hints as to how the given completion proposal was calculated

The following methods are available (for completeness, we include `property` again):

## property

declares a new property. The full form is like this. Name is the only required field:

```
property name : "nameOfTask",
type: String, declaringType:
"java.lang.String", isStatic :
false,
    isDeprecated: false,
provider : "A readable name for
your DSL",
    doc : "<b>Enter javadoc
here</b> html is supported"
```

## method

declares a new method. The full form is like this. Name is the only required field:

```
method name : "nameOfTask",
type: String, declaringType:
"java.lang.String",
  params : [ arg1 : String,
arg2 : Class],
  namedParams : [arg3 : Long,
arg4 : Short],
  optionalParams: [arg5:
Byte], isStatic : false,
  isDeprecated: false, // if
true, then any uses of this
method will have a line through
it
  noParens: false, // if true
content assist will assume this
is a groovy command chain
expression and avoid using
parens
  doc : "<b>Enter javadoc
here</b> html and javadoc tags
supported",
  provider : "A readable name
for your DSL"
```

A note on parameter kinds:

- `params` : regular parameters are added via content assist on the method
- `namedParams` : named parameters are added via content assist on the method prefixed by a name. And if content assist is invoked after a paren or a comma (with no text prefix), unused named parameters will be suggested.
- `optionalParams` : optional parameters are only available when performing completion after a paren or comma (with no text prefix). They are not included during normal method content assist.

Parameter names may include whitespace and special characters as long as the parameter is quoted. For example, it is possible to simulate varargs like this:

```
method name : "meth", params :
["... values", String]
```

<b>delegatesTo</b>	<p>adds all of the public methods in the delegated type to the current type. For example:</p> <div style="border: 1px dashed blue; padding: 10px; margin: 10px 0;"> <pre>delegatesTo List</pre> </div> <p>will add all public methods of list to the current type for content assist, underlining, hovers, and navigation.</p>
<b>delegatesTo</b>	<p>(alternative form with named arguments) adds all of the public methods in the delegated type to the current type. There is a possibility to parameterize using named arguments:</p> <ul style="list-style-type: none"> <li>• <code>type</code> : (required) the type to delegate to.</li> <li>• <code>asCategory</code> : (optional) if true, treat the delegated type as a category and only include static methods where the first parameter matches the type of the current type.</li> <li>• <code>useNamed</code> : (optional) if true, content assist will be applied using named arguments.</li> <li>• <code>except</code> : (optional) a list of method names to exclude from the delegation (useful if these methods are already being added through some other contribution).</li> <li>• <code>isDeprecated</code> : (optional) if true then this contribution is deprecated and all references to delegated methods will appear in the UI with a line through them.</li> </ul>

For example:

```
delegatesTo type:List, asCategory:false,
useNamed:false, except:["addAll",
"removeAll"], isDeprecated:true
```

<b>delegatesToUseNamedArgs</b>	<p>(<i>deprecated</i>, delegatesTo type:"com.Foo", useNamed:true instead) similar to delegatesTo, but uses named arguments when applying content assist proposals</p>
--------------------------------	---

And the following properties

<b>provider</b>	sets the provider for the entire contribution block. Eg, <pre>provider = "My Groovy DSL"</pre> will ensure that "My Groovy DSL" appears in content assist next to all contributions added by this block.
<b>currentNode</b>	Accesses the current Groovy AST node (an expression node)
<b>enclosingNode</b>	Accesses the Groovy AST node (an expression node) that encloses the <code>currentNode</code> . This is the parent node of the <code>currentNode</code> in the AST.
<b>wormhole</b>	A means to pass state between contribution blocks. An example is given below.

This list of available properties and methods will likely change as we continue to work on DSLD.

## Binding

Sometimes, the items matched in the contributing pointcut are required in the contribution block. You can use named arguments for pointcuts to bind a name to the object that the pointcut matched. This binding is available inside of the contribution block.

For example, here is the DSL for the `@Delegate` AST transform:

```
contribute(currentType(fields :  
findField(annotatedBy(Delegate)))) {  
    fields.each { FieldNode field ->  
        delegatesTo type:field.declaringType  
    }  
}
```

The `fields` argument is bound to all of the fields in the current type with the `Delegate` annotation. If there is only one field, then the result is a single Groovy `FieldNode`, if there are multiple matches, then `fields` is bound to an object of type `List<FieldNode>`. As a general rule, bound variables are never null inside of a contribution block. If they were null, then that would imply that a match had not occurred.

Sometimes, it might be necessary to bind on the outermost pointcut component. In this case, you can use the `bind()` pointcut. For example, the following is the syntax for the `@Singleton` AST transform

```
contribute( bind( type  
:currentType(annotatedBy(Singleton))) {  
    method name:"instance", type:type,  
    isStatic:true, declaringType:type,  
        doc:"Get the singleton instance of  
this Class"  
    }  
}
```

## SupportsVersion and AssertVersion

To ensure that a script only runs when particular features are installed, you can use the `assertVersion` top-level method. This method call should go at the top of a script to prevent any other part of the script from executing if the script is not supported.

The syntax looks like this:

```
assertVersion(component1:"x.y.z",  
component2:"a.b.c")
```

This means that the script is only active if all components are active with a version greater than or equal to the ones supplied. If anything does not match, then the entire script is disabled.

Currently, only 'groovy', 'groovyEclipse', and 'grailsTooling' are supported, but we may add other component kinds later. A real example is here:

```
assertVersion(groovy:"1.7.8",groovyEclipse:"  
2.1.3")
```

Alternatively, you can use the `supportsVersion` top-level method for a similar purpose. This method returns a boolean (true iff the current version matches the version requirements in the method arguments). This allows you to execute different contribution blocks depending on version constraints. For example:

```
// Power asserts have changed packages
between 1.7 and 1.8
if (supportsVersion(groovy:"1.8.0")) {

    contribute(currentType("org.codehaus.groovy.
runtime.powerassert.PowerAssertionError")) {
        // contribute methods and properties
    }
} else {

    contribute(currentType("org.codehaus.groovy.
transform.powerassert.PowerAssertionError"))
{
    // contribute methods and properties
}
}
```

## Register pointcut

It is possible to create and register a locally defined pointcut. You can do so by invoking the `registerPointcut` closure at the top level of a DSLD file. `registerPointcut` takes two arguments: the name of the pointcut to register and a closure that serves as the test to see whether or not the pointcut matches.

For example, the following defines `twoArgs`, a pointcut that matches when the evaluation argument is a `MethodNode` with two arguments with the proper names:

```

import org.codehaus.groovy.ast.MethodNode
import org.codehaus.groovy.ast.Parameter;

registerPointcut("twoArgs", {
    if (it instanceof MethodNode) {
        Parameter[] params = it.parameters
        if (params?.length >= 2 && params[0].name
            == "firstName" && params[1].name ==
            "lastName") {
            return params.collectEntries{ [it.name,
it.type] }
        }
    }
})

```

This pointcut can be used as follows:

```

contribute( (enclosingMethod(twoArgs() &
name("processName")) &
currentTypeIsEnclosingType())) {
    method name: "checkDatabaseForNames"
}

```

This will match on the following code:

```

def processName(String firstName, String
lastName, String middleName) {
    //...in here
}

```

Notice that when there is a match, the return value is a collection of all of the matched parameter types. This return value can be bound to a name and made available in a contribution block, like so:

```
contribute((enclosingMethod(bind(paramNameType  
peCollection : twoArgs()) &  
name("processName")) &  
currentTypeIsEnclosingType())) {  
  for (paramNameTypes in  
paramNameTypeCollection) {  
    method name: "checkDatabaseForNames",  
params: paramNameTypes  
  }  
}
```

Now, the generated method will have the same parameter names and types as the method declaration that encloses it. We need to iterate through `paramNameTypeCollection` because all named bindings are collections. Note that the default groovy method `collectEntries` is only available in Groovy 1.8 or greater.

There is more that you can do with `registerPointcut`. The pointcut closure recognizes all named arguments that are passed in to it. For example:

```

registerPointcut("alsoTwoArgs", {
    def result = null
    if (!methods || !current) {
        return null
    }
    // since current and methods are passed in
    as bindings, they are collections and need
    an iterator
    ClassNode type =
current?.iterator()?.next()
    if (type) {
        for (elt in methods) {
            if (elt instanceof MethodNode) {
                Parameter[] ps = elt.parameters
                if (ps?.length >= 2 && ps[0].name ==
"firstName" && ps[1].name == "lastName") {
                    return "findProcessorForClass" +
type.getNameWithoutPackage()
                }
            }
        }
    }
})

```

```

contribute(bind(names : alsoTwoArgs(methods
: enclosingMethod("processName"), current :
currentTypeIsEnclosingType())) {
    for (name in names) {
        property name:name, type:GroovyObject
    }
}

```

This matches the following code:

```
class FullName {
    def processName(firstName, lastName) {
        this.findProcessorForClassFullName()
    }
}
```

## Wormhole

It is possible to pass state between contribution blocks using the wormhole. More about this later...

### The dark side of DSLDs

Since DSLDs are running in the same process as your current Eclipse is, there is some danger. Unless you really know what you are doing, some things that you should **not** try:

- Add `System.exit()` to any of your scripts (take a wild guess as to what will happen...)
- Make any changes to global state
- Access `org.eclipse.core.internal.runtimeAdapterManager` or any other Eclipse singleton.

I think you get the point. We give you the power and you must decide what to do with it. Just like many parts of the Groovy eco-system, the DSLD language gives you all the rope you need to hang yourself.

In the future, it is likely that DSLDs will be executed in the context of a security manager to prevent these kinds of problems.

### *Some larger examples*

Some larger examples can be found at [DSL D examples](#).

### *DSL D examples*

Here are some larger examples of DSLDs.

All scripts here are available under an EPL license. You are free to redistribute any of the scripts below, but please include the following header in the file:

```

/*****
*****
* Copyright (c) 2011 Codehaus.org,
SpringSource, and others.
* All rights reserved. This program and the
accompanying materials
* are made available under the terms of the
Eclipse Public License v1.0
* which accompanies this distribution, and
is available at
* http://www.eclipse.org/legal/epl-v10.html
*
* Contributors:
* Andrew Eisenberg - Initial
implementation
*****
*****/

```

Feel free to add your own scripts to this page.

- [Delegate](#)
- [Singleton](#)
- [Mixin](#)
- [Newify](#)
- [SwingBuilder](#)
- [DSL D Meta Script](#)

## Delegate

This DSL D handles the @Delegat AST transform through a call to delegatesTo.

```
currentType(fields :
fields(annotatedBy(Delegate))).accept {
  provider = "Delegate AST Transform"
  // fields is a collection of FieldNodes
  for (field in fields) {
    delegatesTo(field.type)
  }
}
```

Note that we could have used `bind` instead of `currentType` and had the same behavior.

## Singleton

This shows how `@Singleton` is implemented. We need to explicitly use a `bind` pointcut here since we want to bind to the outermost pointcut.

```
// Don't need to bind here since currentType
is always available in a contribution block
currentType(annotatedBy(Singleton)).accept {
  provider = "Singleton AST Transform"
  property name:"instance", type:currentType,
  isStatic:true, declaringType:currentType,
  doc:"Get the singleton instance of this
Class"
  method name:"getInstance",
  type:currentType, isStatic:true,
  declaringType:currentType, doc:"Get the
singleton instance of this Class"
}
```

## Mixin

The code for @Mixin looks like this:

```
import
org.codehaus.groovy.ast.AnnotationNode
import
org.codehaus.groovy.ast.expr.ClassExpression
import
org.codehaus.groovy.ast.expr.Expression
import
org.codehaus.groovy.ast.expr.ListExpression

currentType(mixins:
annotatedBy(Mixin)).accept {
    provider = "Mixin AST Transform"
    for (AnnotationNode mixin in mixins) {
        if (mixin instanceof AnnotationNode) {
            Expression expr =
mixin.getMember("value")
            if (expr instanceof ClassExpression) {
                delegatesTo expr
            } else if (expr instanceof
ListExpression) {
                for (Expression ex in expr?.expressions)
                {
                    delegatesTo ex
                }
            }
        }
    }
}
```

# Newify

Here is the @Newify DSLD. This matches whenever an enclosing field, method, or class is annotated by groovy.1 ang.Newify. We add both the Python-style constructors and the Ruby style constructors.

```
import
org.codehaus.groovy.ast.AnnotationNode
import org.codehaus.groovy.ast.ClassNode;
import
org.codehaus.groovy.ast.ConstructorNode;
import
org.codehaus.groovy.ast.expr.Expression;
import
org.codehaus.groovy.ast.expr.ListExpression;

(enclosingMethod(newify:
annotatedBy(Newify)) |
enclosingClass(newify: annotatedBy(Newify))
| enclosingField(newify:
annotatedBy(Newify))).accept {
    provider = "Newify AST Transform"
    for (AnnotationNode a : newify) {
        def classes = findClasses(a)
        println "CurrentType: " + currentType
        // when there are no classes in the list,
then applies to all classes
        if (classes.isEmpty() ||
classes.contains(currentType)) {
            // no arg constructor
            // Python style
            method name: "new", type: currentType,
```

```

declaringType: currentType, isStatic: true
  // Ruby style
  method name:
currentType.nameWithoutPackage, type:
currentType, isStatic: true

  for (cons in
(currentType.declaredConstructors?: [])) {
    def extractedParams =
extractParams(cons)
    // Python style
    method name: "new", type: currentType,
declaringType: currentType, params:
extractedParams, isStatic: true
    // Ruby style
    method name:
currentType.nameWithoutPackage, type:
currentType, params: extractedParams,
isStatic: true
  }

}
}
}

// helpers
// convert to a list of class nodes
List<ClassNode> findClasses(AnnotationNode
a) {
  def value = a.getMember("value")
  if (value instanceof ListExpression) {

```

```
    return value.expressions.collect{ it.type
}
} else if (value instanceof Expression) {
return [value.type]
} else {
return []
}
}
```

```
// Convert Parameters to a map if String ->
ClassNode
```

```
def extractParams(ConstructorNode cons) {
def params = [ : ]

for (param in (cons.parameters ?: [] )) {
params."${param.name}" = param.type
}
```

```
}  
  params  
}
```

## SwingBuilder

Although this next example is quite long, it is fairly straight forward. We are just adding lots and lots and lots of methods to the `SwingBuilder` class. This script is only partially complete since we don't include documentation for all contributed methods. It might be possible to use `BeanInfo` to generate this script automatically.

```
import groovy.model.DefaultTableColumn  
import groovy.swing.SwingBuilder  
import groovy.swing.impl.*  
import javax.swing.*  
import javax.swing.border.CompoundBorder;
```

```
import org.codehaus.groovy.binding.*  
import java.awt.*
```

```
currentType(SwingBuilder).accept {  
  provider = "SwingBuilder DSL"
```

```
  method name: "action", type: Action,  
  useNamedArgs: true, params: [name: String,  
  shortDescription: String, smallIcon: String,  
  accelerator:String, closure: Closure], doc:  
  ""
```

```
  Actions are the lifeblood of a full fledged  
  swing application. Other sources can expound  
  on their usefulness.
```

```
  <br/> see <a  
  href="http://groovy.codehaus.org/SwingBuilde
```

```
r.action">http://groovy.codehaus.org/SwingBuilder.action</a>
```

```
"""
```

```
method name: "frame", type: JFrame,  
useNamedArgs: true, params:  
[contentPane:Container,  
defaultCloseOperation:int, title:String,  
glassPane:Component,  
layeredPane:JLayeredPane,  
contentPane:Container],
```

```
doc: """
```

```
Generates a JFrame.
```

```
<br/> see <a
```

```
href="http://groovy.codehaus.org/SwingBuilder.frame">http://groovy.codehaus.org/SwingBuilder.frame</a>
```

```
"""
```

```
// and so on...
```

```
method name: "actions", type: List,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
```

```
method name: "bean", useNamedArgs: true,  
doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
```

```
method name: "bind", type: FullBinding,
```

useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "borderLayout", type: BorderLayout, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "boundedRangeModel", type: DefaultBoundedRangeModel, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "box", type: Box, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "boxLayout", type: BoxLayout, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "button", type: JButton, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "buttonGroup", type: ButtonGroup, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "cardLayout", type: CardLayout, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "checkBox", type: JCheckBox, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "checkBoxMenuItem", type: JCheckBoxMenuItem, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "closureColumn", type: DefaultTableColumn, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "compoundBorder", type: CompoundBorder, useNamedArgs: true, doc: "See <a

`href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>`

`method name: "colorChooser", type: JColorChooser, useNamedArgs: true, doc: "See <a`

`href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>`

`method name: "comboBox", type: JComboBox, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>`

`method name: "container", type: Container, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>`

`method name: "desktopPane", type: JDesktopPane, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>`

`method name: "dialog", type: JDialog, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>`

`method name: "editorPane", type:`

```
JEditorPane, useNamedArgs: true, doc: "See
<a
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
  method name: "emptyBorder", type:
border.EmptyBorder, useNamedArgs: true, doc:
"See <a
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
  method name: "etchedBorder", type:
border.EtchedBorder, useNamedArgs: true,
doc: "See <a
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
  method name: "fileChooser", type:
JFileChooser, useNamedArgs: true, doc: "See
<a
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
  method name: "flowLayout", type:
FlowLayout, useNamedArgs: true, doc: "See <a
href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"
  method name: "formattedTextField", type:
JFormattedTextField, useNamedArgs: true,
doc: "See <a
```

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "gbc", type: GridBagConstraints, useNamedArgs: true, doc: "See <a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "gridBagConstraints", type: GridBagConstraints, useNamedArgs: true, doc: "See <a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "glue", type: Component, useNamedArgs: true, doc: "See <a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "gridBagConstraints", type: GridBagConstraints, useNamedArgs: true, doc: "See <a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "gridBagLayout", type: GridBagLayout, useNamedArgs: true, doc: "See <a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder

er</a>"

method name: "gridLayout", type: GridLayout, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "hbox", type: Box, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "hglue", type: Component, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "hstrut", type: Component, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "internalFrame", type: JInternalFrame, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "label", type: JLabel, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

er</a>"

method name: "layeredPane", type: JLayeredPane, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "list", type: JList, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "loweredBevelBorder", type: border.BevelBorder, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "loweredEtchedBorder", type: border.EtchedBorder, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "map", type: Map, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "matteBorder", type: border.MatteBorder, useNamedArgs: true, doc:

"See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "menu", type: JMenu, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "menuBar", type: JMenuBar, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "menuItem", type: JMenuItem, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "optionPane", type: JOptionPane, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "overlayLayout", type: OverlayLayout, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"

method name: "panel", type: JPanel,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "passwordField", type:  
JPasswordField, useNamedArgs: true, doc:  
"See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "popupMenu", type:  
JPopupMenu, useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "progressBar", type:  
JProgressBar, useNamedArgs: true, doc: "See  
<a  
href=\"http://groovy.codehaus.org/Swing+Build  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "propertyColumn", type:  
table.TableColumn, useNamedArgs: true, doc:  
"See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "radioButton", type:  
JRadioButton, useNamedArgs: true, doc: "See  
<a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "radioButtonItem", type: JRadioButtonMenuItem, useNamedArgs: true, doc: "See <a

href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "rigidArea", type: Component, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "scrollBar", type: JScrollBar, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "scrollPane", type: JScrollPane, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "separator", type: JSeparator, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>

method name: "slider", type: JSlider,

`useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"`

`method name: "spinner", type: JSpinner, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"`

`method name: "spinnerDateModel", type: SpinnerDateModel, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"`

`method name: "spinnerListModel", type: SpinnerListModel, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"`

`method name: "spinnerNumberModel", type: SpinnerNumberModel, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"`

`method name: "splitPane", type: JSplitPane, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Build`

er</a>"

method name: "springLayout", type:  
SpringLayout, useNamedArgs: true, doc: "See  
<a  
href=\"http://groovy.codehaus.org/Swing+Buil  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "tabbedPane", type:  
JTabbedPane, useNamedArgs: true, doc: "See  
<a  
href=\"http://groovy.codehaus.org/Swing+Buil  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "table", type: JTable,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Buil  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "tableColumn", type:  
table.TableColumn, useNamedArgs: true, doc:  
"See <a  
href=\"http://groovy.codehaus.org/Swing+Buil  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "tableLayout", type:  
TableLayoutRow, useNamedArgs: true, doc:  
"See <a  
href=\"http://groovy.codehaus.org/Swing+Buil  
der\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "tableModel", type:

```
table.TableModel, useNamedArgs: true, doc:
"See <a
href=\"http://groovy.codehaus.org/Swing+Buil
der\">http://groovy.codehaus.org/Swing+Build
er</a>"
  method name: "td", type: TableLayoutCell,
useNamedArgs: true, doc: "See <a
href=\"http://groovy.codehaus.org/Swing+Buil
der\">http://groovy.codehaus.org/Swing+Build
er</a>"
  method name: "textArea", type: JTextArea,
useNamedArgs: true, doc: "See <a
href=\"http://groovy.codehaus.org/Swing+Buil
der\">http://groovy.codehaus.org/Swing+Build
er</a>"
  method name: "textField", type: JTextField,
useNamedArgs: true, doc: "See <a
href=\"http://groovy.codehaus.org/Swing+Buil
der\">http://groovy.codehaus.org/Swing+Build
er</a>"
  method name: "textPane", type: JTextPane,
useNamedArgs: true, doc: "See <a
href=\"http://groovy.codehaus.org/Swing+Buil
der\">http://groovy.codehaus.org/Swing+Build
er</a>"
  method name: "toggleButton", type:
JToggleButton, useNamedArgs: true, doc: "See
<a
href=\"http://groovy.codehaus.org/Swing+Buil
der\">http://groovy.codehaus.org/Swing+Build
er</a>"
```

method name: "toolbar", type: JToolBar,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
er\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "tr", type: TableLayoutRow,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
er\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "tree", type: JTree,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
er\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "vbox", type: Box,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
er\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "vglue", type: Component,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
er\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "viewport", type: JViewport,  
useNamedArgs: true, doc: "See <a  
href=\"http://groovy.codehaus.org/Swing+Build  
er\">http://groovy.codehaus.org/Swing+Build  
er</a>"

method name: "vstrut", type: Component,

**useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"**

**method name: "widget", type: Component, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Builder\">http://groovy.codehaus.org/Swing+Builder</a>"**

**method name: "window", type: JWindow, useNamedArgs: true, doc: "See <a href=\"http://groovy.codehaus.org/Swing+Buil**

```
der\">http://groovy.codehaus.org/Swing+Builder</a>"
}
```

## DSL Meta Script

Finally, here is the script that defines editing support for DSLs themselves. A few comments:

- Notice the use of `wormhole` to store the declaration of binding names so that they can appear in content assist later.
- We are accessing the `org.codehaus.groovy.eclipse.dsl.script.PointcutFactory`, which is a class defined inside of Groovy-Eclipse where all of the pointcuts are registered. Doing this ensures that the list of pointcuts available for content assist is always synchronized with what is available in Groovy-Eclipse.
- There is a mock interface `IPointcut` that is defined. It is based on an internal interface of the same name, but in a different package. `IPointcut` only exists in the script so that the `accept` method will not be underlined.
- This script is long and complicated, but it shows many of the features available in the DSL language.

```
import org.codehaus.groovy.ast.ASTNode
import org.codehaus.groovy.ast.ClassNode
import
org.codehaus.groovy.ast.expr.MapEntryExpression

/**
 * This is the meta DSL descriptor for
 *.dsl files.
 */

// first thing to do is to check that
versions are correct
// for this script to be evaluated any
further, all conditions must be met
// also supports grailsTooling and sts,
which are synonyms and correspond to the STS
version eg- 2.6.0
```

```
// we can add more version checking on
request
supportsVersion(groovy:"1.7.8",groovyEclipse
:"2.1.3")
```

```
interface IPointcut {
    /**
     * Accepts a contribution group to this
    Pointcut. What this means
     * is that whenever this pointcut evaluates
    to a match, then the
     * contribution group is evaluated with the
    bindings that have
     * been generated from the pointcut match
     *
     * @param c the contribution group to
    accept
     */
    def accept(Closure c);
}
```

```
// You can store shared pointcuts in a
variable
// This particular pointcut matches all join
points that are inside of a
// groovy project and inside a file with a
*.dsld extension and are scripts
def dsldFile =
nature("org.eclipse.jdt.groovy.core.groovyNa
ture") & fileExtension("dsld") & isScript()
```

```

// You can also create a closure around
pointcuts and assign them to
// a variable. Note that when using these
variables, you must include parens.
def insideContribution = {
enclosingCallName("accept") & inClosure() }

// Ensure that the 'accept' method is
available for all closures and variables
that correspond to pointcuts
(dsldFile & ( ~ insideContribution() ) & (
~currentType(subType(Script)) )).accept {
  delegatesTo IPointcut
}

// here we store all bound names inside of
the wormhole so that they can be available
later
(dsldFile &
enclosingCallDeclaringType(subType(Script))
& (~ enclosingCallName("supportsVersion")) &

  ( ~ inClosure() ) & bind(var :
currentIdentifier()))).accept {
  if (enclosingNode instanceof
MapEntryExpression &&
    var.contains(((MapEntryExpression)
enclosingNode).keyExpression)) {
    def bindings = wormhole.bindings
    if (!bindings) {

```

```
    bindings = [ ]
    wormhole.bindings = bindings
  }
  var.each { bindings << it.text }
}

// Define the kinds of pointcuts
// note the different ways of calling the
two composite pointcuts
// Also, be careful to use parens around
negation '~' since operator precedence may
make the '~' apply to the call to 'accept'
(dsldFile & ( ~ insideContribution() ) &
currentType(subType(Script)) &
(~enclosingCallName("registerPointcut")) &
  currentTypeIsEnclosingType()).accept {
  provider = "the meta-DSL D script"
```

```
  // in here, we can list all pointcuts
explicitly, or we can access the internal
PointcutFactory object
  // A little bit naughty, but this is the
easiest way to maintain consistency with all
possible pointcuts
  // ...the PointcutFactory class is a secret
class that is declared by groovy-eclipse
  // Yes, you can use many Eclipse classes
here. The editor won't like them, so they
must be fully qualified
  // and you cannot import them or use them
```

as types for variables

```
Map<String, String> pointcutNames =
org.codehaus.groovy.eclipse.dsl.script.Point
cutFactory.docRegistry
for (pointcutName in
pointcutNames.entrySet()) {
    method name : pointcutName.key, type:
IPointcut, params : [pointcutArg : Object],
doc : pointcutName.value
}
```

```
method name : "supportsVersion", type :
void, params : [versionKindToName :
Map.class ],
doc : '''Specifies that this script is only
active when the specified version
constraints are met.
```

Currently, only 3 constraints are available: groovy, groovyEclipse, sts, and grailsTooling.<br><br>

Use like this:

```
<pre>supportsVersion(groovy:"1.7.10",groovyE
clipse:"2.1.3")</pre>
```

This means that the current script requires **both** Groovy 1.7.10 or later and Groovy-Eclipse 2.1.3 or later'''

```
method name : "registerPointcut", type :
void, params : [name : String, pointcutBody
: Closure],
```

```

    doc : '''
        Registers a custom pointcut.  This pointcut
        is only available from within the current
        script.

        You must specify a name for the pointcut as
        well as a closure that evaluates whether or
        not there is a match'''

}

// Here, specify everything that can be used
// inside of an accept block (also called a
// Contribution Group)
(dsldFile & insideContribution()).accept {
    provider = "the meta-DSL D script"
    property name : "provider", type : String,
    doc : ""Specifies a Provider for
    the current contribution.

        This provider is displayed during
        content assist and in other places to give a
        quick hint as to where this
        contribution element comes from.""
    method name : "property",
        useNamedArgs : true,
        params : [name : String,
            type : Object,
            isStatic : boolean,
            declaringType : Object,
            provider: String,
            doc : String],
    doc : ""
}

```

Specifies a new property contribution.  
<b>name</b> is mandatory, but all other parameters are optional.

"""

```
method name : "method",
  useNamedArgs : true,
  params : [name : String,
            type : Object,
            params : Map,
            useNamedArgs : boolean,
            isStatic : boolean,
            declaringType : Object,
            provider: String,
            doc : String],
  doc : ""
```

Specifies a new method contribution.  
<b>name</b> is mandatory, but all other parameters are optional.

"""

```
method name : "delegatesTo", params : [
type : Object ], doc : ""
```

Specify that the currentType delegates to the given type. The currentType is the type being analyzed.

And the given type is specified as a parameter (either a String, Class, or ClassNode). All fields and

methods of the given type will be available from the currentType.

```
    ""
    method name : "delegatesToUseNamedArgs",
    params : [ type : Object ], doc : ""
        Specify that the currentType delegates to
        the given type. The currentType is the type
        being analyzed.
        And the given type is specified as a
        parameter (either a String, Class, or
        ClassNode). All fields and
        methods of the given type will be
        available from the currentType.<br/><br/>
        Named arguments will be used for all
        methods.
    ""
```

```
    property name : "wormhole", type : Map, doc
    : ""Use the wormhole to stuff in values
    calculated in one contribution group
    to make it available later in another
    contribution group""
```

```
    property name : "currentType", type :
    ClassNode,
        doc : "This is the declaring type of the
        current expression being evaluated."
    property name : "currentNode", type :
    ASTNode, doc : "This is the ASTNode being
    evaluated"
    property name : "enclosingNode", type :
    ASTNode, doc : "This is the ASTNode
    enclosing the ASTNode being evaluated"
```

```
// Now, extract all bindings from the
wormhole and add them as contributions
for (binding in wormhole.bindings) {
    property name : binding, doc : "Binding
created from pointcut"
}
}

// Add the contributions for inside a
"registerPointcut" call.  User-registered
pointcuts are
// not fully fleshed out yet, so best not to
use them yet.
(dsldFile & inClosure() &
enclosingCallName("registerPointcut")).accep
t {
    property name : "currentScope", type :
org.eclipse.jdt.groovy.search.VariableScope
```

```
    property name : "currentType", type :  
    ClassNode  
}
```

### *Support for testing DSLDs*

As of Groovy-Eclipse 2.5.2, there is now command line support for testing DSLD descriptors. Because DSLD support is intertwined with Eclipse API, it is not possible to simply run unit tests against them. Rather, it is necessary to load up the Eclipse workbench and process the DSLD files from within the workbench in order to evaluate them.

## Setting up

You can only evaluate DSLDs against an existing Eclipse project. So, you need to create an eclipse project inside an existing workspace.

1. Start Eclipse
2. Create a groovy project
3. Create some Groovy classes
4. You can add your DSLDs directly to this project, or you can add them later when you perform the type checking.
5. Add some type annotations to assert the static types of certain expressions. Eg-

```
def name = "myName"  
name // TYPE:java.lang.String  
[name] // TYPE:java.util.List<java.lang.String>  
[name:1] // TYPE:java.util.Map<java.lang.String,java.lang.Integer>
```

Shut down Eclipse and you are ready to run static type checking.

## Running static type checking

The command looks like this:

```
eclipse -application  
org.codehaus.groovy.eclipse.staticCheck  
[-data /path/to/eclipse/workspace] \  
    [--help] [-h] [--extra_dslds <FILES>]  
[--assertions_only] [--excludes <PATH>] \  
    [--includes <PATH>] <PROJECT_NAME>
```

Here is a description of the arguments:

<code>--help OR -h</code>	Prints a help message and exits.
<code>--extra_dslds</code>	list of extra dsld files to be included in this check. Use ' ' as a file separator.
<code>--assertions_only</code>	Don't report unknown types. Only look for type assertions
<code>--excludes</code>	Project-relative exclusion filters.
<code>--includes</code>	Project-relative inclusion filters.
<code>&lt;PROJECT_NAME&gt;</code>	Name of a project to type check. Must be already in the workspace.

The exclusion and inclusion filters use the syntax from [ant filesets](#). Individual filters can be concatenated using '|'.  
For example:

```
--includes "*" --excludes  
"src/test/java/**/*Test.groovy|src/test/groovy/**/*Test.groovy"
```

includes all files, but excludes all `*Test.groovy` files in `test/java` and `test/groovy`.

## The results

Let's assume that there is a Groovy project in an Eclipse workspace with a single file:

```
/MyProj/src/MyScript.groovy  
  
happiness // TYPE:java.lang.Integer  
happiness // TYPE:java.lang.String  
somethingUndefined
```

And this DSLD file exists outside of the workspace:

```
/tmp/myapp.dsld
```

```
currentType().accept { property  
name:"happiness", type:Integer }
```

Then executing this command:

```
eclipse -application  
org.codehaus.groovy.eclipse.staticCheck  
--extra_dslds /tmp/myapp.dsld MyProj
```

Will produce the following output:

```
Adding file:/tmp/other.dsld
```

```
Checking: /MyProj/src/MyScript.groovy
```

```
Line 2: Invalid inferred type. happiness  
Expected: java.lang.String Actual:  
java.lang.Integer
```

```
Line 3: unknown type: somethingUndefined
```

```
Removing file:/tmp/other.dsld
```

If you want to suppress all of the `unknown type` warnings, then include the `--assertions_only` parameter on the command line.

You may also see some error messages sent to `syserr`, like this:

**Job found still running after platform shutdown. Jobs should be canceled by the plugin that scheduled them during shutdown: org.eclipse.jdt.internal.ui.InitializeAfterLoadJob**

These messages can be safely ignored. These messages occur since Eclipse is being shut down early enough before all initialization jobs are complete. We'll be fixing this in future versions.

## More to come

This is just a first pass at static checking. We are looking for feedback as to the best way to display failed assertions, or if there is anything else that can be improved. If there is significant interest, we can do the work to make it easy to create JUnit (or Spock!) tests for DSLs, but we need to hear the feedback first.

### A unified approach to DSL support in IDEs

Based on some comments to the blog post on [DSL descriptors in Groovy-Eclipse](#), we should be discussing a more unified approach to DSL support in both Groovy-Eclipse and IntelliJ. Other IDEs, such as NetBeans, Emacs, etc could also benefit from this kind of unified approach.

#### Primary goals of DSL support

#### Secondary goals of DSL support

#### Differences between existing approaches in Groovy-Eclipse and IntelliJ

#### Commonality between existing approaches

### Groovy-Eclipse 2.5.0 New and Noteworthy

We have several major advances included in the 2.5.0 release of Groovy-Eclipse. In addition to including both Groovy 1.8.0 and Groovy 1.7.10, we have introduced support for DSL descriptors that allow first class support for custom DSLs in Groovy-Eclipse.

You can use the following update site to install this release.

#### **i** Groovy-Eclipse update site for Eclipse 3.6 and 3.7

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

And a zipped version of the update site is available at:

## **i** Zipped Groovy-Eclipse update site for Eclipse 3.6 and 3.7

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.5.0.xx-20110511-1600-e36-RELEASE.zip>

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.5.0.xx-20110511-1900-e37-RELEASE.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

### *Outline*

- [DSL support](#)
- [Groovy 1.8 support](#)
- [Parameter guessing proposals](#)
- [Extended outline support for Scripts](#)
- [Distinguish read access from write access](#)
- [Better generics inferencing support](#)
- [Compatibility](#)
- [Bug fixes](#)
- [What's next?](#)

### *DSL support*

The Groovy language is an excellent platform for creating domain specific languages (DSLs). A good DSL can make programs more concise and expressive as well as make programmers more productive. However, until now these DSLs were not directly supported by the Groovy editor. When DSLs are used heavily, standard IDE features like content assist, search, hovers, and navigation lose their value. For a while now, it has been possible to write an Eclipse plugin to extend Groovy-Eclipse, but this is a heavy-weight approach that requires specific knowledge of the Eclipse APIs. This is no longer necessary. Groovy-Eclipse now supports DSL descriptor (DSL D) files that makes your DSL fully understood by Groovy-Eclipse.

For example, it is possible to create a simple DSL D script that adds support for a [distances DSL](#).

With no DSL D in your workspace, the following code snippet will be underlined and have no hover information:

```
3.m + 2.yd + 2.ml - 1.km
```

But add the following code snippet to a DSL D and the hovers will go away, and appropriate documentation will be available through hovers:

```

currentType( subType( Number ) ).accept {
    [ m: "meter", yd: "yard", cm:
    "centimeter", mi: "mile", km:
    "kilometer" ].each {
        // for clarity, assume $url is set
elsewhere
        property name: it.key, type: "Distance",
            doc: " " "A <code>${it.value}</code>
from <a href=\"$url\">$url</a> " " "
    }
}

```

Then the code snippet from above looks like this:



The screenshot shows a code editor with the following text: `3.m + 2.yd + 2.mi - 1.km`. Below the code, there is a tooltip or documentation window. The first line of the tooltip is **Distance Number.m**. The second line is **A Meter** from <http://joesgroovyblog.blogspot.com/2007/09/and-miles-to-go-before-i-sleep.html>.

For an overview of the feature, see the [blog post on DSL Support](#). For a complete description on how to create and use DSLDs, see the documentation for [DSL Descriptors for Groovy-Eclipse](#).

#### *Groovy 1.8 support*

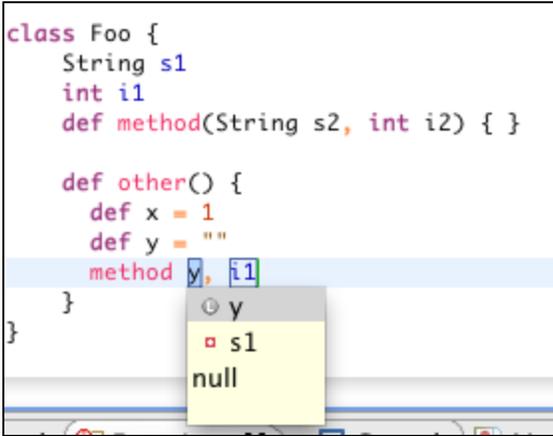
Groovy 1.8.0 final is now available in Groovy-Eclipse as an optional add-on. Groovy 1.8 will not be installed by default with the rest of Groovy-Eclipse. Instead, it is available as a separate feature that can be installed from the Groovy-Eclipse update site. For information on how to do this, see [Compiler Switching within Groovy-Eclipse](#).

#### *Parameter guessing proposals*

Groovy-Eclipse will now suggest some relevant content assist proposals when filling in the arguments for methods. The relevant proposals will be based on the inferred type of local variables and fields that are in the scope of the newly inserted method proposal. For example:

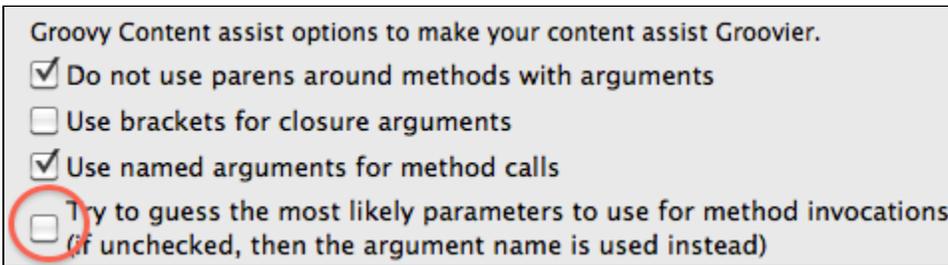
```
class Foo {
  String s1
  int i1
  def method(String s2, int i2) { }

  def other() {
    def x = 1
    def y = ""
    method y, i1
  }
}
```



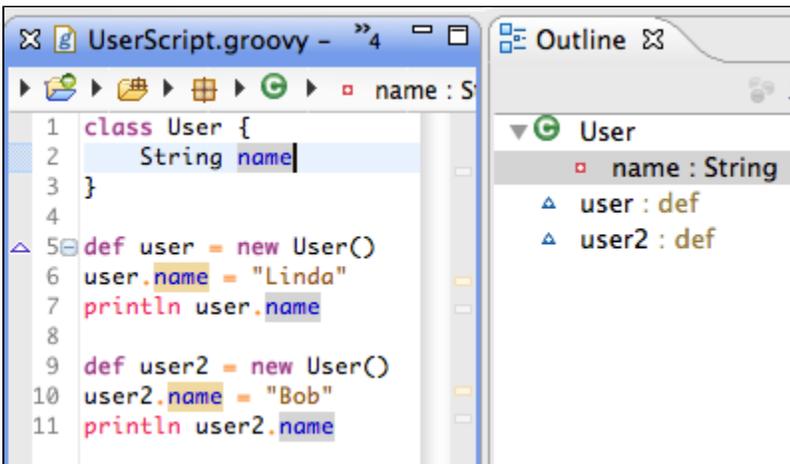
In the screenshot above, the method declaration `method(String, int)` is added through content assist. For the first argument, it is suggested that either the variable `y` or the field `s1` be inserted. This is because the types of these two instances are inferred to be compatible with the type of the first parameter to `method` (i.e., they are `String`}). Similar (but not shown), the second argument will show `{x` and `i1`. This functionality is similar to the parameter guessing proposals available in the Java editor.

You can disable this functionality and instead use the old kind of content assist that simply adds the parameter name to the replacement string. This option is available from Preferences -> Groovy:



#### Extended outline support for Scripts

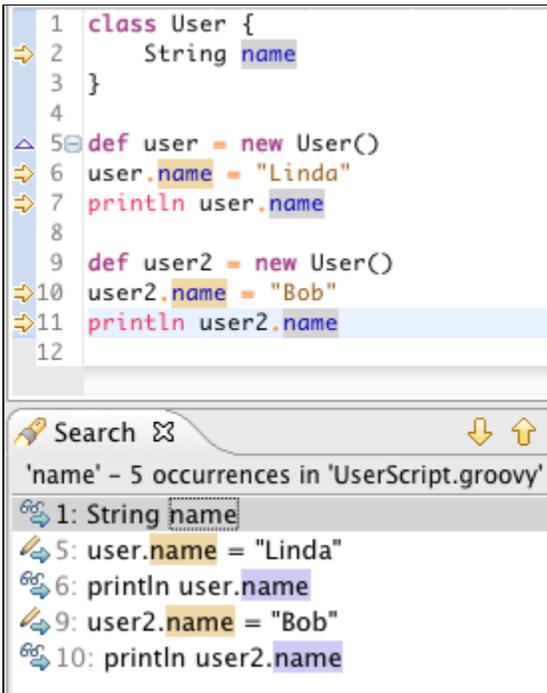
When working with scripts, the outline view displays contents in a more Groovy-like way. The run, and main methods are hidden. Instead, script variable declarations are displayed.



The enhanced outline view is part of Groovy-Eclipse and is available for third party plugins to use their own outline extender. For more information, see the feature request here: [GRECLIPSE-1029](https://bugs.eclipse.org/bugs/show_bug.cgi?id=47999).

*Distinguish read access from write access*

When reperforming a *find references* action in the Groovy editor (*CTRL-Shift-U* on Windows/Linux), read occurrences are now highlighted differently from write occurrences:



```
1 class User {
2     String name
3 }
4
5 def user = new User()
6 user.name = "Linda"
7 println user.name
8
9 def user2 = new User()
10 user2.name = "Bob"
11 println user2.name
12
```

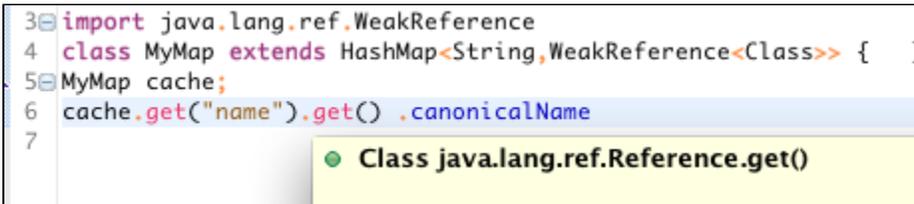
Search   

'name' - 5 occurrences in 'UserScript.groovy'

- 1: String name
- 5: user.name = "Linda"
- 6: println user.name
- 9: user2.name = "Bob"
- 10: println user2.name

#### **Better generics inferencing support**

Groovy-Eclipse now recognizes more complex forms of generics to help seed type inferencing. For example, in the following code, Groovy-Eclipse can determine that the second call to "get()" is declared by the WeakReference class and returns a java.lang.Class object:



```
3 import java.lang.ref.WeakReference
4 class MyMap extends HashMap<String,WeakReference<Class>> { }
5 MyMap cache;
6 cache.get("name").get().canonicalName
7
```

● Class java.lang.ref.Reference.get()

For more information, see [GRECLIPSE-997](#).

#### **Compatibility**

Groovy-Eclipse 2.5.0 includes Groovy 1.7.10 by default. Groovy 1.8.0 can be optionally installed. Groovy-Eclipse is installable on STS 2.6.0, Eclipse 3.6.0, 3.6.1, or 3.6.2. There are builds available targeting Eclipse 3.7 as well. See above for the update sites.

#### **Bug fixes**

Over [45 issues](#) have been fixed for this release.

#### **What's next?**

The 2.5.1 final release is scheduled for the end of June, around the time that Eclipse 3.7 will be released. We will be working on enhancing our DSLD support and including some standard DSLD files inside of Groovy-Eclipse with the next release.

In this release, we are particularly grateful for Maxim Hamm's work on outline view extensibility ([GROOVY-1029](#)).

We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up!

### Groovy-Eclipse 2.5.1 New and Noteworthy

The 2.5.1 release is coming about a month after the 2.5.0 release. Despite the short release cycle, we have included a number of new enhancements to Groovy-Eclipse's DSLD support.

You can use the following update site to install this release:

#### Groovy-Eclipse update site for Eclipse 3.7 and 3.6

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

And a zipped version of the update site is available at:

#### Zipped Groovy-Eclipse update site for Eclipse 3.7 and 3.6

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.5.1.xx-20110628-1600-e37.zip>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.5.1.xx-20110627-1300-e36.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

#### Outline

- [Eclipse 3.7 support](#)
- [groovy-eclipse-compiler for Maven](#)
- [DSL Descriptor enhancements](#)
  - [Groovy DSL Support classpath container](#)
  - [Multiple DSLDs of same name](#)
  - [Disabling DSLD support](#)
- [Built-in AST transforms recognized in the editor](#)
  - [What to expect for the future of DSLDs](#)
- [Launch configurations](#)
- [Compatibility](#)
- [Bug fixes](#)
- [What's next?](#)

### Eclipse 3.7 support

With the 2.5.1 release, Eclipse 3.7 has become the main development branch. Although we have been releasing 3.7-based versions of Groovy-Eclipse for a while, it has not been our primary release until now. We will continue to support the Eclipse 3.6-based version of Groovy-Eclipse, but it will no longer be the main branch of our development.

### groovy-eclipse-compiler for Maven

We have released the groovy-eclipse-compiler for Maven that lets you use the Groovy-Eclipse compiler in your maven projects. This provides a more robust way of compiling mixed Groovy-Java projects with Maven than previous approaches have allowed. See all the documentation at the [Groovy-Eclipse compiler plugin for Maven](#) page.

### DSL Descriptor enhancements

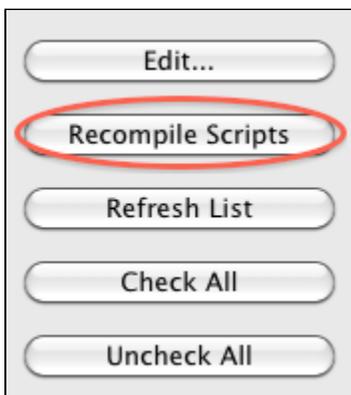
Most of the [DSL](#) enhancements are centered around sharing DSLD files among multiple projects in the workspace. For those of you unfamiliar with DSLDs, please read the [Better DSL Support in Groovy-Eclipse](#) blog post.

## Groovy DSL Support classpath container

All Groovy projects will have the *Groovy DSL Support* classpath container added automatically. No production code is in the classpath container. It is only meant to contain DSLD files. Inside, there are two class folders: `global_dsl_d_support` and `plugin_dsl_d_support`, described below.

## Global DSLDs

The `global_dsl_d_support` folder points to `${user.home}/.groovy/greclipse/global_dsl_d_support`. It is meant to contain DSLDs that should be shared amongst all projects in the workspace. You can place any DSLD files here and it will be picked up by Groovy-Eclipse. Actually, it's not that easy...because of Eclipse [Bug 346928](#), you must go to Preferences -> Groovy -> DSLD and click *Recompile* before the new scripts will be found:

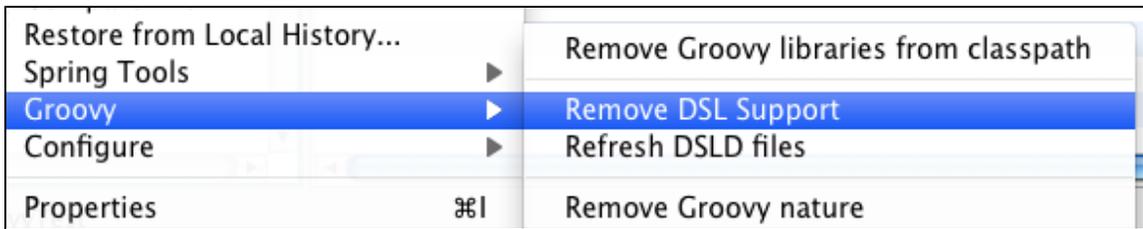


## Plugin DSLDs

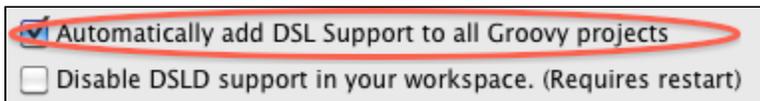
The `plugin_dsl_d_support` folder points to `${eclipse.home}/plugins/org.codehaus.groovy${active.groovy.version}/plugin_dsl_d_support`. This folder contains the DSLD files that are shipped with this installation of Groovy-Eclipse. Since these files will be changed and updated on each upgrade, it is not recommended that users edit these files.

## Controlling the Classpath container

It is possible to manually add and remove the *Groovy DSL Support* classpath container by clicking on a groovy project and selecting add/remove *Groovy DSL Support*:



Alternatively, if you do not want the classpath container added automatically to any project, you can disable this in the Eclipse preferences -> Groovy -> DSLD:



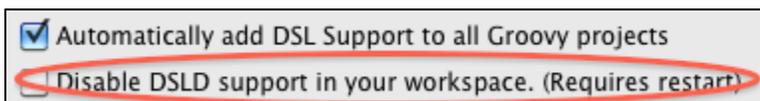
## Multiple DSLDs of same name

In order to support an eco-system of DSLDs and be able to gracefully handle the conflicts that could occur when multiple versions of a single DSLD exist in a project, Groovy-Eclipse now uses the following heuristics when looking for DSLDs:

1. All DSLDs with the same name are considered the same, regardless of which folder or jar they are located in.
2. DSLDs that exist in the project (i.e., not in a jar file and not coming from a binary class folder) take precedence.
3. After that, the classpath ordering takes precedence. In other words, a DSLD located in a jar at the beginning of the classpath will take precedence over a DSLD located in a jar at the end of the classpath.
4. If multiple versions of a single DSLD file is located in the project or in a single classpath entry, then an arbitrary one is chosen. The others are ignored.

## Disabling DSLD support

It is possible to completely disable DSLD support. You can do this from the preferences -> Groovy -> DSLD page:



### *Built-in AST transforms recognized in the editor*

Groovy-Eclipse now ships with several built-in DSLD files that handle some built-in AST transforms like `@Delegate`, `@Mixin`, and `@Singleton`. Additionally, support exists for Swing Builder. The *Groovy DSL Support* classpath container must be added to your project in order for this support to be available. For example, here is the singleton transform in use:



And here is swing builder in use:

```
count = 0
new SwingBuilder().edt {
    frame(title:'Frame', size:[300, 300], show: true)
        BorderLayout()
        textLabel = label(text:"Click the button!",
        button(text: JLabel SwingBuilder.label()
            acti See SwingBuilder.label
            cons
```

All hovers over swing builder components have links to full documentation.

## What to expect for the future of DSLDs

As we gain more experience with DSLDs, we are learning more about the areas of DSLDs that need improvements. As we make changes to DSLD support, we will try hard to maintain backwards compatibility as well as be clear about which parts of the language have been deprecated. There are a number of things that we will be working on.

- Pointcut changes: we are realizing that a number of pointcuts are poorly named and there are others that are ambiguous or superfluous. Expect some changes with the names of available pointcuts. We will try to keep old pointcuts available, but just mark them as deprecated.
- Compatibility changes: IntelliJ's GDSLs solve a similar problem that DSLDs solve. However, the syntax is different. We are in discussion with the IntelliJ's Groovy team to see if there is a way of unifying the syntax of the two languages. This is a long-term goal and may not be ready for the 2.5.2 release. This will require some more significant changes to the syntax, but as much as possible, we will try to ensure that the old syntax still works.

### *Launch configurations*

The Groovy Script launcher is now more robust. Upgrading Groovy-Eclipse versions will not break existing launch configurations. Furthermore, these launch configurations can be saved as files in a project and committed into a VCS to be shared between developers.

### *Compatibility*

Groovy-Eclipse 2.5.1 includes Groovy 1.7.10 by default. [Groovy 1.8.0 can be optionally installed](#). Groovy-Eclipse is installable on STS 2.7.0, Eclipse 3.7.0, 3.6.0, 3.6.1, or 3.6.2. There are different update sites for Groovy-Eclipse targeting Eclipse 3.6 and 3.7. See above for the update sites.

### *Bug fixes*

There have been 30 bug fixes in this release. You can see all of them in the [Codehaus issue tracker](#).

### *What's next?*

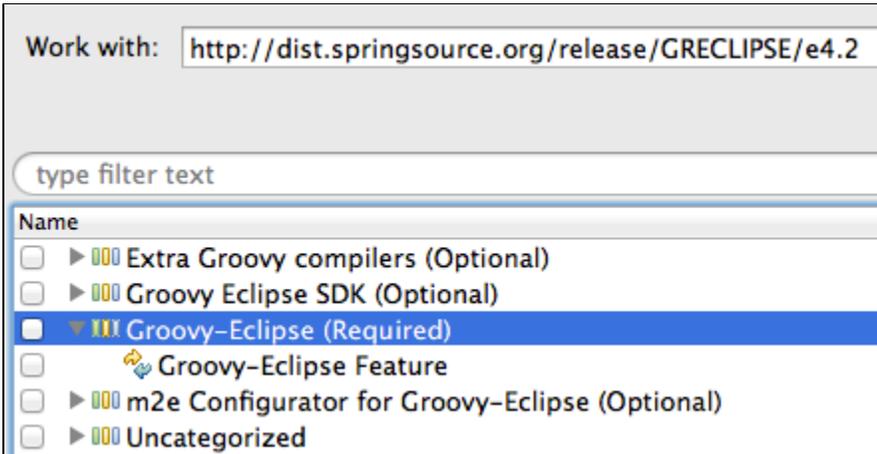
We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up!

## Compiler Switching within Groovy-Eclipse

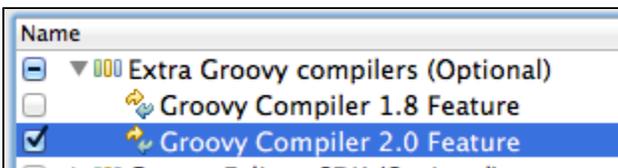
Groovy-Eclipse supports the 2.0, 1.8, and 1.7 compiler for Groovy. However, due to Groovy-Eclipse's tight integration with the compiler, only one version of the compiler can be used in the workspace at one time. Switching

compiler levels requires a restart.

By default, Groovy-Eclipse comes installed with a 1.7 and 1.8 compiler. These are the compilers that are installed when you select the Groovy-Eclipse feature from the Groovy update site:



If you would like to install the 2.0.x compiler, then you go to the Eclipse install manager and select the Groovy 2.0 compiler from the extra compilers section of the update site:



Then click *Next* and follow the instructions to complete the installation. A restart is necessary.

Here are the release update sites for Groovy-Eclipse:

Eclipse level	Release update site
4.2 (Juno)	<a href="http://dist.springsource.org/release/GRECLIPSE/e4.2">http://dist.springsource.org/release/GRECLIPSE/e4.2</a>
3.7 (Indigo)	<a href="http://dist.springsource.org/release/GRECLIPSE/e3.7/">http://dist.springsource.org/release/GRECLIPSE/e3.7/</a>

- [Compiler levels](#)
- [Specifying compiler level on the command line](#)
- [Switching compiler levels in the preference page](#)
- [Uninstalling the 2.0 compiler](#)
- [Switching from outside of Eclipse](#)
- [Another possibility](#)
- [Problems or questions?](#)

#### **Compiler levels**

After installing Groovy-Eclipse without the extra compilers, the Groovy 1.8 compiler is active by default. And the Groovy 2.0 compiler becomes active after installing the extra compiler.

#### **Specifying compiler level on the command line**

As of Groovy-Eclipse 2.5.2, it is possible to specify the compiler level when starting Eclipse from the command line. Just add the following arguments to the command line to start a particular Groovy version when running Eclipse:

Groovy 1.7	<code>-groovy.compiler.level 17</code>
Groovy 1.8	<code>-groovy.compiler.level 18</code>
Groovy 2.0	<code>-groovy.compiler.level 20</code>

It is possible to run multiple instances of Eclipse at the same time targeting different compiler levels as long as they point to different workspaces.

Alternatively, you can add the arguments to the `eclipse.ini` file (must be on separate lines and before the `-vmargs` entry).

**Switching compiler levels in the preference page**

You can switch between the 1.7 and 1.8 compilers for your Eclipse workspace in the *Preferences -> Groovy -> Compiler* page:



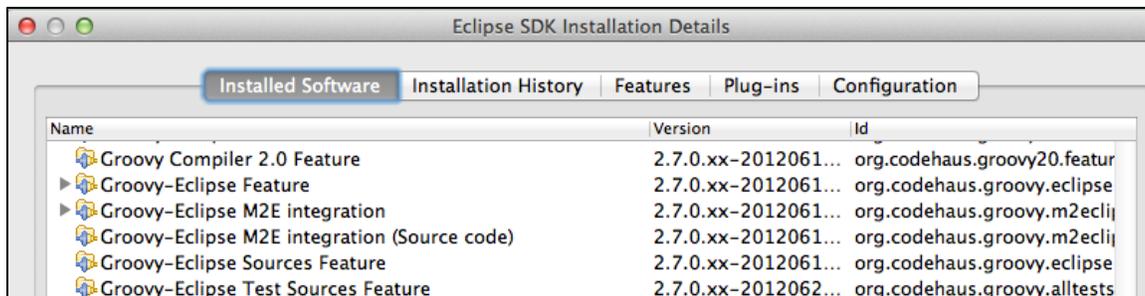
Simply click the button and Eclipse will restart automatically. The hyperlink shown on the preference page will navigate you to this wiki page.

**Uninstalling the 2.0 compiler**

You can uninstall the 2.0 compiler through the update manager. In the lower right corner of the update manager, there is a link that says *What is already installed?*:



Click the link, select the Groovy 2.0 compiler, and then click *Uninstall...*:



Then follow the instructions and restart to complete the uninstallation process. Once the 2.0 compiler is uninstalled,

you can follow the steps above to reinstall it.

#### *Switching from outside of Eclipse*

If you cannot switch compilers from the Groovy Compiler preference page, follow the instructions to perform the switch from outside of Eclipse:

1. Shut down Eclipse
2. Go to `eclipse/configuration/org.eclipse.equinox.simpleconfigurator`
3. Make a backup copy of `bundles.info`
4. Open `bundles.info` in a text editor
5. Find the line for `org.codehaus.groovy_1.8` and delete (thus leaving the 1.7 compiler enabled)
6. Restart eclipse

To re-enable the 1.8 compiler, simply reinstate the backed up copy of `bundles.info`.

#### *Another possibility*

Over at [AppSatori](#) there is another suggestion on how to do compiler switching. It requires a little more setup than the proposal here, but it allows you to run two workspaces simultaneously using each compiler.

#### *Problems or questions?*

There are several possibilities for getting support on your Groovy-Eclipse installation. See [here](#) for your support options.

## **Groovy-Eclipse compiler plugin for Maven**

Groovy-Eclipse provides a compiler plugin for Maven. Using the compiler plugin, it is possible to compile your maven projects using the Groovy-Eclipse compiler.

- [How to use the compiler plugin---Setting up the POM](#)
- [Setting up the source folders](#)
  - [Do nothing](#)
  - [Do almost nothing](#)
  - [Use the groovy-eclipse-compiler mojo for configuring source folders](#)
  - [Use the build-helper-maven-plugin](#)
- [Sample project and source code](#)
- [Why another Groovy compiler for Maven? What about GMaven?](#)
- [Project Lombok](#)
- [Groovy-Eclipse configurator for m2Eclipse](#)

#### Release Builds

#### Development Builds

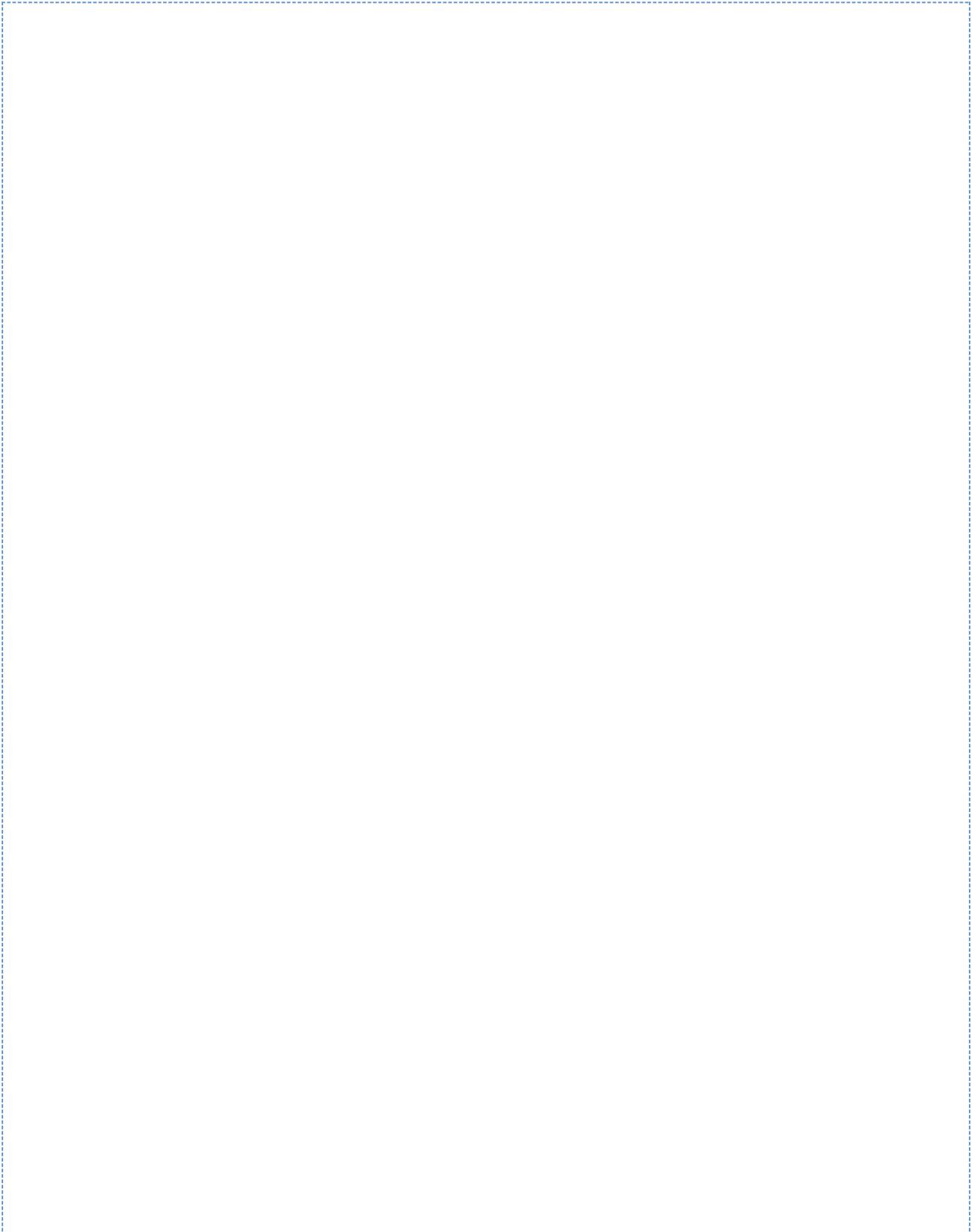
- [Where to find more information and ask questions](#)

**i** The most recent version of the Groovy-Eclipse-Compiler plugin for maven is **2.7.1-01**. **2.8.0-01-SNAPSHOT** is now available from <http://nexus.codehaus.org/snapshots/>.

The most recent version of the `groovy-eclipse-batch` artifact is 2.1.3-01 (for Groovy 2.1) and 2.0.7-01 (for Groovy 2.0). They are both available from maven central.

***How to use the compiler plugin---Setting up the POM***

In your plugin section, you must change the compiler used by the [maven-compiler-plugin](#). Like the [javac ant task](#), the maven-compiler-plugin does not actually compile, but rather delegates the compilation to a different artifact (in our case, the `groovy-eclipse-batch` artifact):



```
<build>
...
<plugins>
  <plugin>

<artifactId>maven-compiler-plugin</artifactId>
  <version>2.3.2</version>
  <configuration>

<compilerId>groovy-eclipse-compiler</compilerId>
  <!-- set verbose to be true if you
want lots of uninteresting messages -->
  <!-- <verbose>true</verbose> -->
  </configuration>
  <dependencies>
    <dependency>

<groupId>org.codehaus.groovy</groupId>

<artifactId>groovy-eclipse-compiler</artifactId>
  <version>2.7.0-01</version>
  </dependency>
  </dependencies>
</plugin>
...
</plugins>
</build>
```

By default, this will use Groovy 2.0.4 to compile your code. If you would prefer to use 1.8.6 or 1.7.10, then add another dependency to the maven-compiler-plugin:

```
<dependency>  
  <groupId>org.codehaus.groovy</groupId>  
  
  <artifactId>groovy-eclipse-batch</artifactId>  
  >  
    <version>1.8.6-01</version>  
    <!-- <version>1.7.10-06</version> -->  
</dependency>
```

This will allow Groovy files to be compiled. The groovy-eclipse-compiler recognizes all settings supported by the [maven-compiler-plugin](#).

Remember that you still need to specify a groovy artifact as a build dependency in addition to the maven-compiler-plugin dependency. The groovy dependency version should match the compiler version. Something like this:

```
<dependencies>  
  ...  
  <dependency>  
    <groupId>org.codehaus.groovy</groupId>  
    <artifactId>groovy-all</artifactId>  
    <version>1.8.6</version>  
  </dependency>  
  ...  
</dependencies>
```

Note that the groovy-eclipse-compiler and groovy-eclipse-batch artifacts are available in Maven-central, so there is no need to explicitly declare any extra repositories.

There are several ways to set up your maven project to recognize Groovy source files

## Do nothing

The simplest way to set up your source folders is to do nothing at all: add all of your Groovy files to `src/main/java` and `src/test/java`. This requires absolutely no extra configuration and is easy to implement. However, this is not a standard maven approach to setting up your project. If you require a more standard maven approach, then it is possible to put your Groovy files in `src/main/groovy` and `src/test/groovy` and your Java files in `src/main/java` and `src/test/java`. There are several ways of doing this.

## Do almost nothing

If there is *at least one* file (Java or not) in `src/main/java`, then all files in `src/main/groovy` will be found. If, however, `src/main/java` is empty, then `src/main/groovy` will be ignored. You can get around this by placing an empty file in `src/main/java` just so that `src/main/groovy` will be recognized. The same is true for `src/test/java` and `src/test/groovy`. This is actually a workaround for [GRECLIPSE-1221](#).

## Use the `groovy-eclipse-compiler` mojo for configuring source folders

(You only need this approach if your project has an empty `src/main/java` or `src/test/java`.)

If your project has no Java files and you don't want to add an empty file in `src/main/java`, then you can configure source files by referencing the `groovy-eclipse-compiler` mojo. Just add this to the `plugins` section of your pom:

```
<build>
  ...
  <plugin>
    <groupId>org.codehaus.groovy</groupId>

    <artifactId>groovy-eclipse-compiler</artifactId>
    <version>2.7.0-01</version>
    <extensions>>true</extensions>
  </plugin>
  ...
</build>
```

The `<extensions>true</extensions>` section is important because this redefines the default lifecycle of your project so that an extra phase is added. This phase has an extra goal attached to it that adds the two Groovy source

folders.

## Use the build-helper-maven-plugin

(You only need this approach if your project has an empty `src/main/java` or `src/test/java`.)

The `build-helper-maven-plugin` allows you to do things like adding extra source folders to your project without needing to redefine the default lifecycle. You need to add this configuration to your build plugin section:

```
<build>
  ...
  <plugin>
    <groupId>org.codehaus.mojo</groupId>

<artifactId>build-helper-maven-plugin</artif
actId>
  <version>1.5</version>
  <executions>
    <execution>
      <id>add-source</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>add-source</goal>
      </goals>
      <configuration>
        <sources>
          <source>src/main/groovy</source>
        </sources>
      </configuration>
    </execution>
    <execution>
      <id>add-test-source</id>
      <phase>generate-test-sources</phase>
      <goals>
```

```
    <goal>add-test-source</goal>
  </goals>
  <configuration>
    <sources>
      <source>src/test/groovy</source>
    </sources>
  </configuration>
</execution>
</executions>
```

```
</plugin>
...
</build>
```

The benefit of using this approach is that you do not need to make any changes to the default lifecycle. The downside is, of course, that you need 31 lines of configuration to do this!

*Sample project and source code*

There is an archetype available for this project. You can use it through a command like this:

```
mvn archetype:generate \
    -DarchetypeGroupId=org.codehaus.groovy \
    -DarchetypeArtifactId=groovy-eclipse-quickst
art \
    -DarchetypeVersion=2.5.2-01 \
    -DgroupId=foo \
    -DartifactId=bar \
    -Dversion=1 \
    -DinteractiveMode=false \

-DarchetypeRepository=https://nexus.codehaus
.org/content/repositories/snapshots/
```

A [sample project](#) using the compiler plugin as well as the source code for the plugin itself are available from the [github repository](#):

 The full URL to the maven projects is:

<https://github.com/groovy/groovy-eclipse/tree/master/extras>

There are several projects in the repository:

- `groovy-eclipse-compiler` : the compiler plugin itself (an m2eclipse project).
- `groovy-eclipse-batch-builder` : a set of ant scripts and configuration files used to build the `groovy-eclipse-batch` artifact. This artifact is an amalgamation of all jars required for compiling Groovy and Java code in Eclipse, including ecj (the Eclipse compiler for Java), the non-UI components of Groovy-Eclipse, the Groovy jars, and various required Eclipse bundles.
- `groovy-eclipse-maven-tests` : a sample project that uses the compiler plugin (an m2eclipse project).
- `org.codehaus.groovy.m2eclipse` : an Eclipse plugin that provides integration between Groovy-Eclipse and m2eclipse (the Maven tooling for Eclipse).
- Feature `org.codehaus.groovy.m2eclipse` : an Eclipse feature that is required for building and releasing the `org.codehaus.groovy.m2eclipse` plugin.

The sample project and archetype is not maintained as well as we would like. Some community help with this would be greatly appreciated. Please see [GRECLIPSE-1285](#).

*Why another Groovy compiler for Maven? What about GMaven?*

GMaven 2.0 and later no longer supports compilation. You can read more about this on the [GMaven 2 project page](#). [GMaven 1.x](#) is now deprecated.

GMaven 1.x had limitations over the `groovy-eclipse-compiler` and for the following reasons GMaven 2.0 compilation is no longer supported:

1. The compiler plugin does not require the creation of Java stubs so that your Groovy files can compile against Java files. This will prevent some arcane compile errors from appearing.
2. The Groovy-Eclipse compiler is the same inside Eclipse and inside Maven, and so configuration across the two platforms can be simplified.
3. The compiler plugin is a [standard compiler plugin](#) for Maven. It therefore follows all allows all the same standard configuration that the Javac compiler plugin uses. This makes it simpler to introduce Groovy into an existing Maven project. All you need to do is change the compiler plugin that the pom references.

There are still some reasons to use GMaven:

1. GroovyDoc tool is not supported because the compiler plugin does not produce stubs.
2. Groovy Mojos are not supported.
3. Groovy scripts cannot be executed in your poms.

Whether or not the Groovy-Eclipse compiler plugin for Maven is appropriate for your project will depend on your requirements.

**Project Lombok**

[Project Lombok](#) is compatible with the `groovy-eclipse-compiler`. There is some extra configuration that you need to do. The `lombok` jar needs to be added to *both the build and compile dependencies sections*:

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>0.10.4</version>
</dependency>
```

Also, the following configuration needs to be added to the maven-compiler-plugin configuration:

```
<configuration>

<compilerId>groovy-eclipse-compiler</compilerId>
  <verbose>>true</verbose>
  <compilerArguments>

<javaAgentClass>lombok.core.Agent</javaAgentClass>
  </compilerArguments>
  <fork>>true</fork>
</configuration>
```

#### *Groovy-Eclipse configurator for m2Eclipse*

If you are going to be working with your maven project inside of Eclipse, it is strongly recommended that you use [m2eclipse](#). And to use your Groovy projects with m2eclipse, you will need to install the Groovy-Eclipse configurator for m2eclipse. This feature is available any of the Groovy-Eclipse update sites (e.g., nightly, milestone, or release). Just go to your Eclipse update manager and add the Groovy-Eclipse update sites (if you haven't done so already). Select the *Groovy-Eclipse M2E integration*.

#### Release Builds

Eclipse level	Release update site
4.2 (Juno)	<a href="http://dist.springsource.org/release/GRECLIPSE/e4.2">http://dist.springsource.org/release/GRECLIPSE/e4.2</a>

3.7 (Indigo)	<a href="http://dist.springsource.org/release/GRECLIPSE/e3.7">http://dist.springsource.org/release/GRECLIPSE/e3.7</a>
--------------	---

#### Development Builds

Eclipse level	Development update site
4.2 (Juno)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e4.2">http://dist.springsource.org/snapshot/GRECLIPSE/e4.2</a>
3.7 (Indigo)	<a href="http://dist.springsource.org/snapshot/GRECLIPSE/e3.7">http://dist.springsource.org/snapshot/GRECLIPSE/e3.7</a>

The Groovy-Eclipse configurator for m2eclipse is not compatible with AspectJ or Scala. So you cannot use a joint AspectJ/Scala/Groovy project in Eclipse. These languages must be separated into separate sub-projects.

#### Where to find more information and ask questions

**i** Please ask all questions on the [Groovy-Eclipse mailing list](#). Any bugs or feature enhancements should go on [Groovy-Eclipse's jira](#).

The compiler plugin was originally described [here](#) and [here](#), but these posts are no longer updated and this page will always contain the more recent information.

#### Groovy-Eclipse 2.1.0RC1 New and Noteworthy

Please see [Groovy-Eclipse 2.1.0RC2 New and Noteworthy](#).

#### Groovy-Eclipse 2.5.2 New and Noteworthy

The Groovy-Eclipse team is proud to announce the release of Groovy-Eclipse 2.5.2. There are many new features available with this release, including Eclipse 3.7.1 support, Java 7 support, and Groovy 1.8.2 support

You can use the following update site to install this release:

#### **i** Groovy-Eclipse update site for Eclipse 3.7 and 3.6

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

If you want the Groovy 1.8 compiler, you must explicitly include it by checking the "Extra compilers" category.

And a zipped version of the update site is available at:

## **i** Zipped Groovy-Eclipse update site for Eclipse 3.7 and 3.6

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.5.2.xx-20110929-1800-e37.zip>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.5.2.xx-20110929-1700-e36.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

### **Outline**

- [Groovy 1.8.2](#)
- [Java 7 and Eclipse 3.7.1](#)
- [Compiler switching through command line argument](#)
- [Enhanced Maven support](#)
- [User Contributed Inferencing Suggestions](#)
- [Inferencing improvements](#)
- [DSL Support](#)
- [Content assist improvements](#)
- [Static type checking \(experimental\)](#)
- [Remove trailing whitespace and semi-colons save actions](#)
- [Editing support for Groovy files outside of the build path](#)
- [Fix import statements after move/copy](#)
- [More precise searching for overloaded methods](#)
- [More precise rename refactoring of overloaded methods](#)
- [Compatibility](#)
- [Bug fixes](#)
- [Shout outs](#)
- [What's next?](#)

### **Groovy 1.8.2**

Groovy 1.8.2 is now available as an optional install from the update sites.

### **Java 7 and Eclipse 3.7.1**

Groovy-Eclipse now ships with Java 7 compatibility and can install on Eclipse 3.7.1.

### **Compiler switching through command line argument**

It is now possible to use command line arguments to specify a compiler level for Groovy-Eclipse. You can add the `-groovy.compiler.level` argument to the launch command to control which compiler level is started.

- Use `-groovy.compiler.level 18` to specify Groovy 1.8
- Use `-groovy.compiler.level 17` to specify Groovy 1.7

For example, to start Eclipse with a 1.7 compiler:

```
eclipse -data /my/groovy17/workspace -groovy.compiler.level 17
```

Use '18' to specify the 1.8 compiler.

See [Compiler Switching within Groovy-Eclipse](#) and [STS-1844](#) for more information.

#### **Enhanced Maven support**

We have released, 2.5.2-01 of the groovy-eclipse-compiler. Additionally, the Groovy-Eclipse configurator for m2e v1.0 is now available from the release update site. Full documentation on Groovy-Eclipse's maven support is available at [Groovy-Eclipse compiler plugin for Maven](#).

## **groovy-eclipse-compiler plugin for Maven**

There have been many improvements to the groovy-eclipse-compiler plugin for Maven. In addition to stability and bug fixing, here is what is new and interesting:

- Now based off of Groovy 1.8.2 (by default) and 1.7.10 (if specified).
- The Java compiler is now Java 7 based (from Eclipse JDT 3.7.1).
- No longer need to use the build helper plugin to specify the `src/main/groovy` and `src/test/groovy` source folders. They are added to the build automatically if they exist.
- Updated groovy-eclipse-compiler archetype project. This archetype is available at: <https://nexus.codehaus.org/content/repositories/snapshots/>

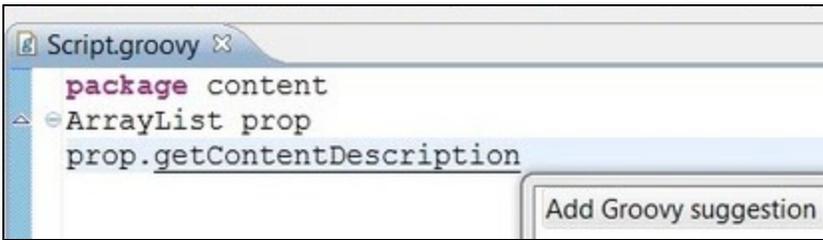
## **M2E support**

Groovy-Eclipse's m2e configurator now works with m2e 1.0 and is also available from the release update site. The old version of the Groovy-Eclipse configurator is still available from the dev update site and we will keep it available for legacy reasons. It is the only version that is compatible with pre 1.0 versions of m2eclipse. However, we recommend against using it if possible.

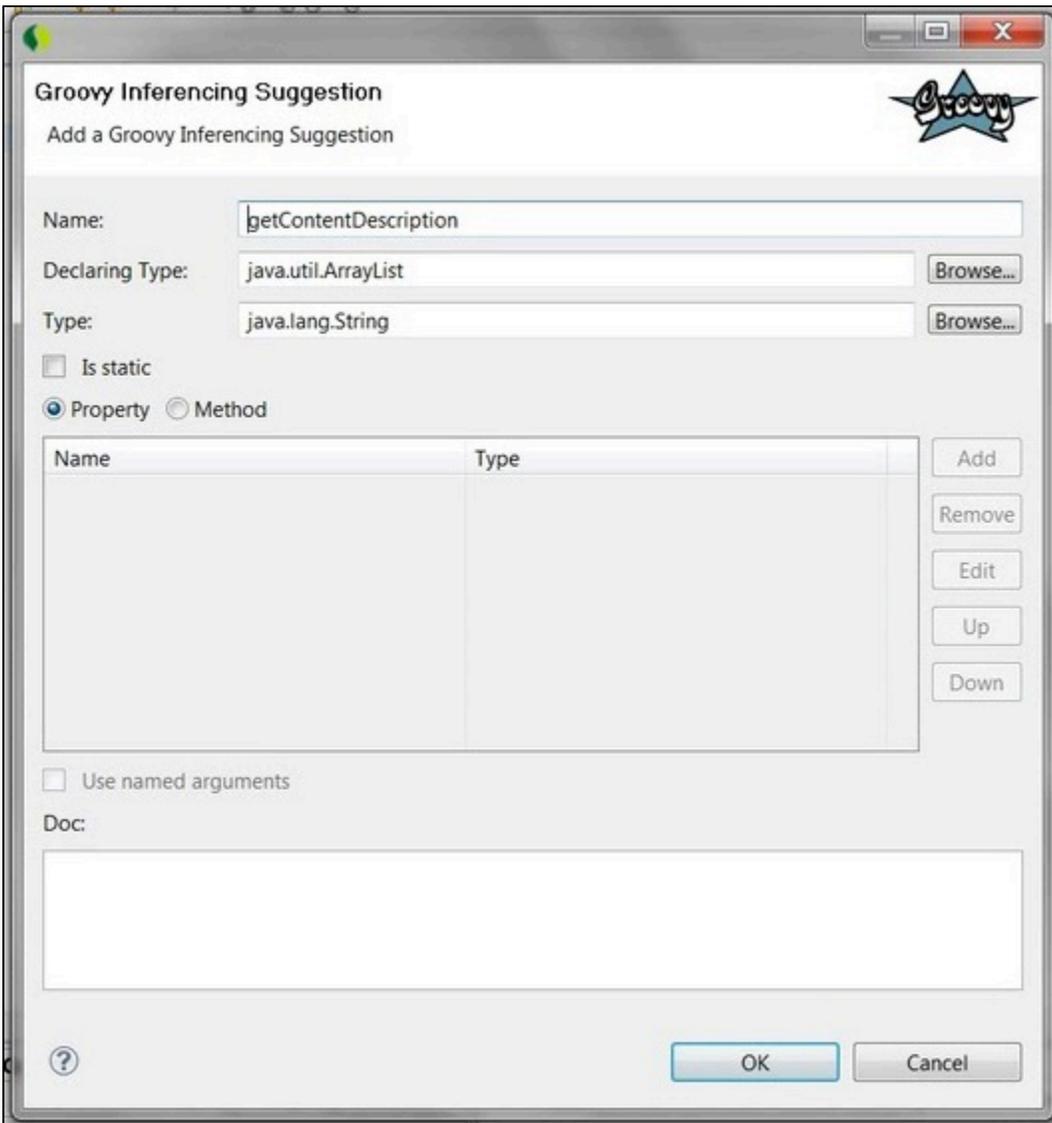
#### **User Contributed Inferencing Suggestions**

There is now another way to extend Groovy-Eclipse's inferencing engine. Using quick assists and a preferences page, users can now contribute dynamic properties and methods to existing classes for a customized editing experience.

As seen in the image below it is now possible to invoke a Quick-Assist (CTRL-1 or CMD-1 on Mac) in order to invoke the *suggestions dialog*. In this example, an unresolved property called `getContentDescription` in the declaring type `java.util.ArrayList` is referenced in a Groovy script. The user then presses CTRL+1 on the editor selection and can add the property via the "Add Groovy suggestion" quick assist.

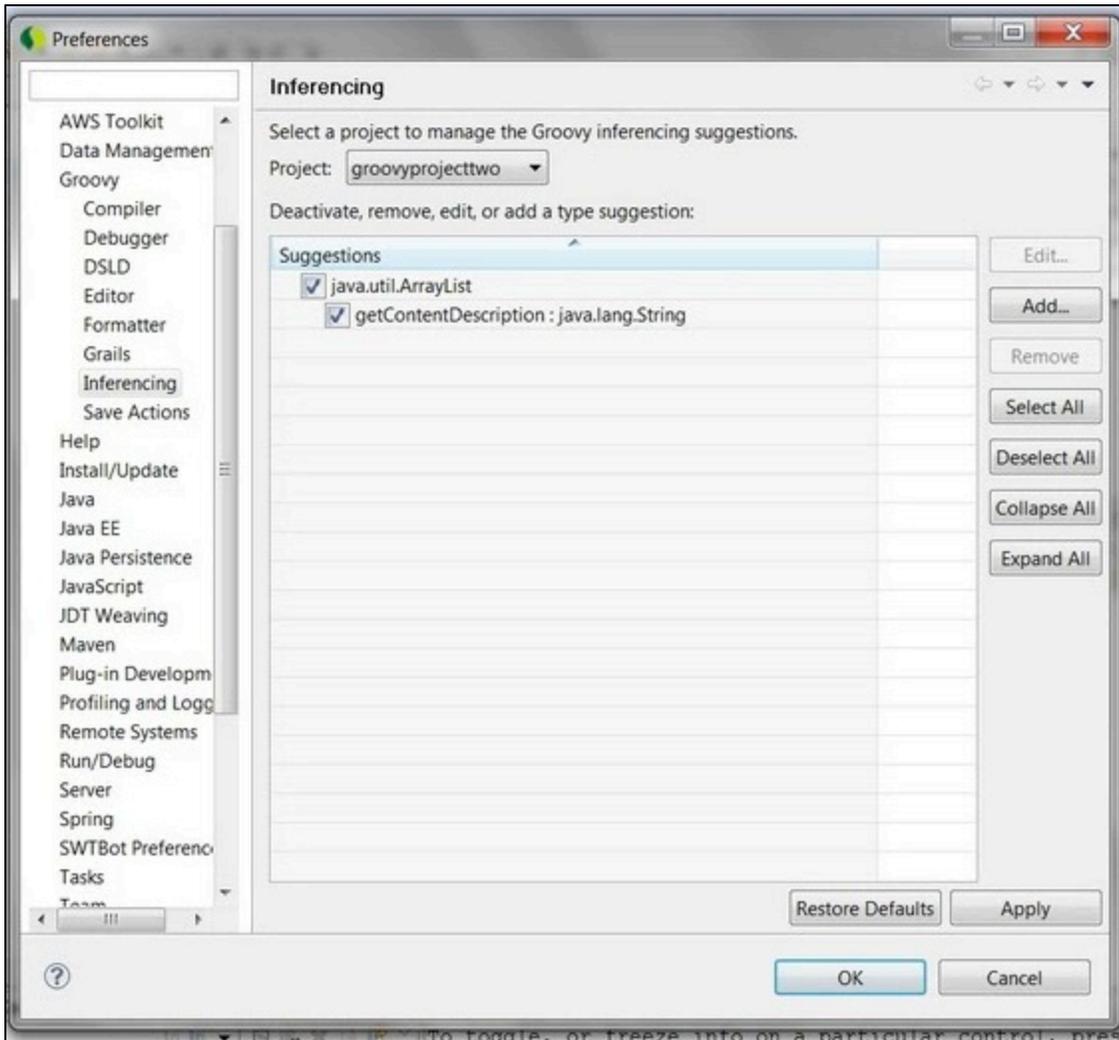


The Groovy Suggestions dialogue then opens, pre-populated with values related to the current selection, like the declaring type, property name and type.



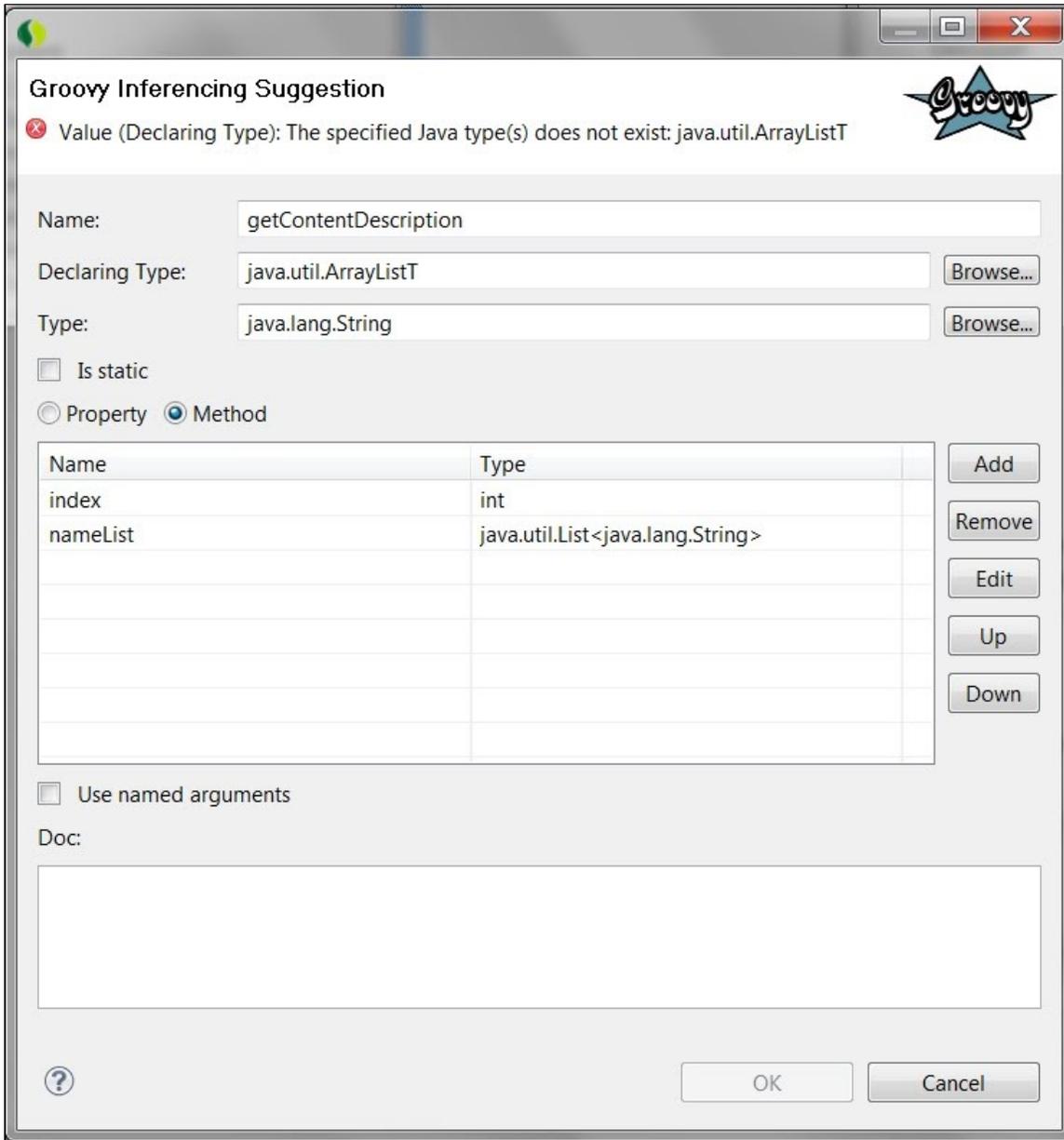
The user can add additional information like doc hovers, or change the suggestion from a property to a method and add parameters.

Once the dialogue is closed, the suggestion is added to the Groovy inferencing engine and the property will now be resolved. The user can add further suggestions, edit or remove existing ones, in the Preferences -> Groovy -> Inferencing preferences page. Additionally, a user can activate or deactivate a suggestion by clicking a check box. Deactivating a suggestion does not remove the suggestion from the list of suggestions for the project, but any references for that suggestion will be unresolved until the user activates it again.

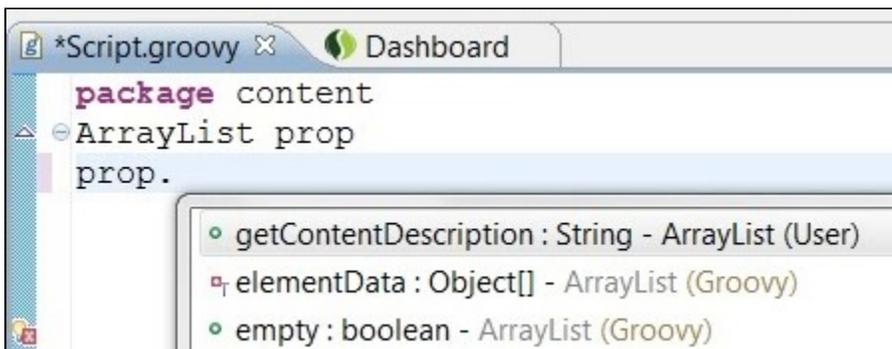


Suggestions are persisted in the `.groovy/suggestions.xdsl` file at the root of all Groovy projects. Each Groovy project has its own `suggestions.xdsl` and so they can be persisted in version control for the project. As of 2.5.2, this feature only supports edits of this file through the preference page or suggestions dialogue. Manual edits of the file are not yet supported.

The suggestions dialogue performs type validation where appropriate, as seen in the following screen shot.



Any suggestions added to a project that are active also appear in content assist, as seen below where getContentDescription appears as an option in content assist:



*Inferencing improvements*

We have added many enhancements with the Groovy-Eclipse inferencing engine. A list of all inferencing issues addressed [is available here](#).

Here are the highlights.

## Better DGM inferencing

Default Groovy Methods now have better inferencing inside of associated closure blocks. For example, the `unique` DGM method now supports inferencing of the parameter on its closure:

```
class Simple { int val }
[new Simple(), new Simple()].unique {
    it.val
}
```

In addition to `unique`, all methods inside the `DefaultGroovyMethods` class now support inferencing on parameters in closures where appropriate. A complete list is available in [GRECLIPSE-1143](#).

## Type inferencing of DGM methods with non-collection arguments

Now, when non-collections types are targets of DGM methods, the iterator variable type is correctly inferred:

```
1. each {
    ... it
}
```

Integer it - STS280M2.run()

## Other implicit category classes

Support for the implicit category classes `DateGroovyMethods`, `SwingGroovyMethods`, `XmlGroovyMethods`, and `ProcessGroovyMethods`.

These classes are now handled just like `DefaultGroovyMethods` in terms of content assist, hovers, and navigation.

For example:

`XmlGroovyMethods`:

```
import org.w3c.dom.NodeList

def method2(NodeList list) {
    list.iterator()
}
```

Iterator<Node> org.codehaus.groovy.runtime.XmlGroovyMethods.iterator(NodeList nodeList)

`ProcessGroovyMethods`:

```
def method2(Process p) {
    p.consumeProcessOutput
}
```

- consumeProcessOutput() : void - ProcessGroovyMethods (Category: ProcessGroovyMethods)
- consumeProcessOutput(Appendable output, Appendable error) : void - ProcessGroovyMethods (Category: ProcessGroovyMethods)
- consumeProcessOutput(OutputStream output, OutputStream error) : void - ProcessGroovyMethods (Category: ProcessGroovyMethods)
- consumeProcessOutputStream(Appendable output) : Thread - ProcessGroovyMethods (Category: ProcessGroovyMethods)
- consumeProcessOutputStream(OutputStream output) : Thread - ProcessGroovyMethods (Category: ProcessGroovyMethods)

For more information, see [GRECLIPSE-1131](#), [GRECLIPSE-1143](#), [GRECLIPSE-1145](#), [GRECLIPSE-1153](#), [GRECLIPSE-1154](#) and [GRECLIPSE-1155](#).

## Multiple assignment statements

The types of variables assigned in multi-assignment statements are now discovered during inferencing. For example, when assigning a list to multiple variables, the static type of the list elements will be assigned to each variable:

```
class Simple { int val }
class Simple2 { int val2 }

def (x, y) = [new Simple(), new Simple2()]
x.val
y.val2
```

Also, when a method returns a list or array, the type parameter or component type is used as the type of assigned variables:

```
class Simple { int val }

List<Simple> method () {
    return [new Simple(), new Simple()]
}

def (x, y) = method()
x.val
y.val
```

For more information, see [GRECLIPSE-1140](#) and [GRECLIPSE-1136](#).

## Spread-dot type inferencing

The spread-dot operator is now correctly handled by the inferencing engine:

```
[x:1,y:2,z:3]*.key[0].toLowerCase()
[x:1,y:2,z:3]*.value[0].intValue()
Integer java.util.Map.Entry.getValue()
```

## Match operator inferencing

Match operator expressions now support inferencing:

```
("" =~ /pattern/).hasGroup()
("" ==~ /pattern/).booleanValue()
```

For more information, see [GRECLIPSE-1159](#).

### DSL Support

We are continuing to improve on our DSL support in Groovy-Eclipse through our DSL descriptors. Below are the significant enhancements. And all enhancements are described in more detail in [DSL Descriptors for Groovy-Eclipse](#)

# namedParams and optionalParams for method contributions

It is now possible to distinguish between regular, named, and optional parameters in DSLD method contributions. Regular parameters are always filled in when performing content assist for the method and they are not prefixed by a name. Named parameters are prefixed by a name and are also always filled in during content assist. And they are placed after all the regular parameters. Lastly, optional parameters are not filled in via content assist of the method. Instead, optional parameters are only available for content assist of named parameters. All parameters are visible during hovers.

Here is an example to make this clear. Consider the following simple DSLD file that adds editing support for the static create method to the Person class:

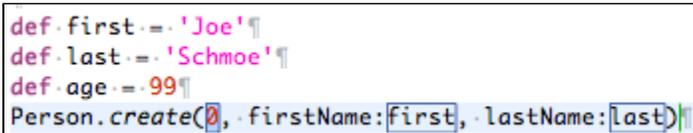
```
currentType("Person").accept {
    method name:"create", type:"Person",
    params:[id:Long],
    namedParams:[firstName:String,
    lastName:String],
    optionalParams:[age:Integer],
    isStatic:true, provider:"Person DSL"
}
```

The method contribution defines params, namedParams, and optionalParams. The first thing to see is that when doing content assist, only the regular and named parameters will be shown:



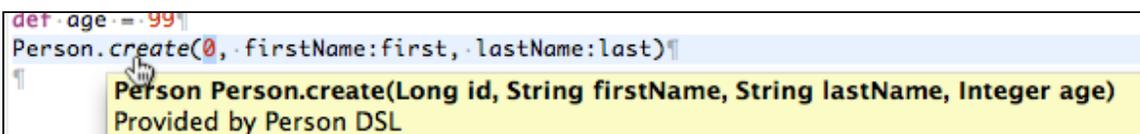
```
Person.c[
  create(Long id, String firstName, String lastName) : Person - Person (Person DSL)
```

And when applying that proposal, only those parameters are displayed (the first parameter is regular, and the second two are named):



```
def first = 'Joe'
def last = 'Schmoe'
def age = 99
Person.create(0, firstName: first, lastName: last)
```

The optional parameters are shown in hovers (along with all other parameters):



```
def age = 99
Person.create(0, firstName: first, lastName: last)
Person Person.create(Long id, String firstName, String lastName, Integer age)
Provided by Person DSL
```

They can also be seen in content assist, when named parameters would be visible:

```
Person.create(0, firstName: first, lastName: last, b
age : __ - Integer : named parameter : (Person DSL)
create(Long id, String firstName, String lastName) : Person - Person (Person DSL)
```

Note that the editor is smart enough to only show parameter proposals if that parameter doesn't already exist, so if we remove the `lastName` named parameter and do content assist, it will show up as a proposal:

```
def age = 99
Person.create(0, firstName: first, b
age : __ - Integer : named parameter : (Person DSL)
lastName : __ - String : named parameter : (Person DSL)
create(Long id, String firstName, String lastName) : Person - Person (Person DSL)
```

*Content assist improvements*

## Named parameter content assist

Methods that expect named parameters (typically specified in a DSL descriptor, see above) can have their named parameters filled in via content assist, as shown in this screenshot:

```
class Person { }
Person.create()
age : __ - Integer : named parameter : (Person DSL)
firstName : __ - String : named parameter : (Person DSL)
lastName : __ - String : named parameter : (Person DSL)
create(String firstName, String lastName, Integer age) : Person - Person (Person DSL)
```

Notice in the screenshot that there are 3 named parameters displayed: `firstName`, `lastName`, and `age`. These are defined in a [DSL descriptor](#) in the project.

As with selecting method proposals during content assist, Groovy-Eclipse will try to find reasonable values to fill in when selecting a named parameter:

```
def first = 'Joe'
def last = 'Schmoe'
def age = 99
Person.create(firstName: , .)
first
last
null
```

There is a caveat. Named parameter proposals will not appear in the proposal list if there is a prefix already typed. So, in the following example, there will be no named parameters proposed since the prefix 'f' already exists:

```
def first = 'Joe'
def last = 'Schmoe'
def age = 99
Person.create(f)
first :
```

## Method context information

Groovy-Eclipse now follows the JDT convention of displaying context information for methods when invoking content assist inside of a method call and there is no prefix. For example, when invoking content assist just after an opening paren, you are only shown a list of known ways to invoke the target method:

```
def doSomething(int a, int b, int c = 9) { }
def doSomething(String first, String second = "") { }
```

doSomething()

- doSomething(String first) : Object - Script (Groovy)
- doSomething(int a, int b) : Object - Script (Groovy)
- doSomething(String first, String second) : Object - Script (Groovy)
- doSomething(int a, int b, int c) : Object - Script (Groovy)

Selecting one of the proposals will show that method's information in tooltips above the caret location:

```
doSomething(String first, String second)
```

Similarly, invoking content assist after a comma will show context information as well:

```
doSomething("", )
```

- doSomething(String first) : Object - Script (Groovy)
- doSomething(int a, int b) : Object - Script (Groovy)
- doSomething(String first, String second) : Object - Script (Groovy)
- doSomething(int a, int b, int c) : Object - Script (Groovy)

For more information, see [GRECLIPSE-674](#).

## Content assist now recognizes closure parameters

When a field has a closure for an initializer, content assist will now reflect this and show method proposal variants for the field:

```
class ClosureClass {
    def field = { String str, int num -> print str + num }
}

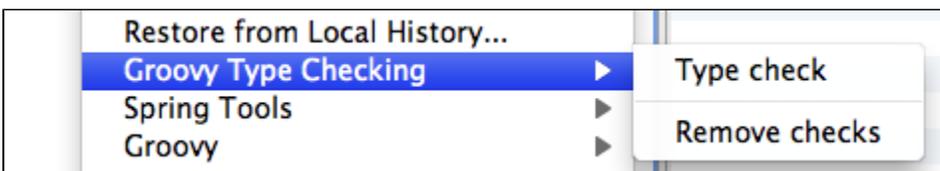
new ClosureClass().field
```

- field : Object - ClosureClass (Groovy)
- field(String str, int num) : Object - ClosureClass (Groovy)

For more information, see [GRECLIPSE-1139](#).

### Static type checking (experimental)

Groovy-Eclipse 2.5.2 ships with an experimental way to perform static type checking on your Groovy projects. There is a new menu category that appears whenever any Groovy files, packages, or projects are selected:



When invoking this command, all groovy files currently selected will be type checked (i.e., sent to the inferencing engine). Warning markers are added to any expressions that would be otherwise underlined in the editor:

```
1 def x = "Some string"
2
3 print x.substring(2)
4 print x.oops
5
```

Type of expression is statically unknown: oops

If you want to remove all of the type check warnings simply select "Remove checks" from the "Groovy Type Checking" menu while selecting the files you want to clean.

You can also add some type assertions to ensure that the static type of an expression is correctly determined by the inferencing engine (this is particularly useful for testing your DSLs):

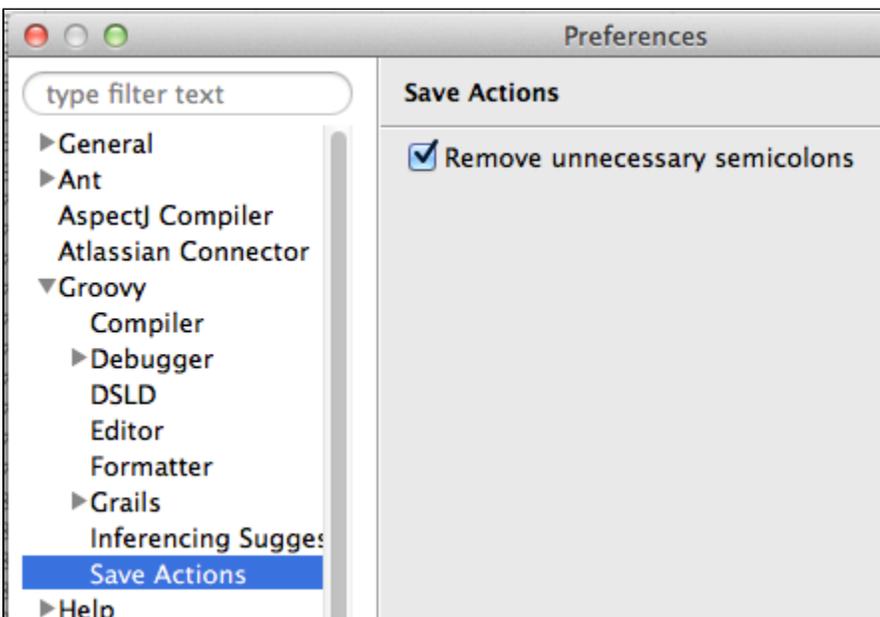
```
1 def x = "Some string"
2
3 print x.substring(2)
4 print x.oops
5
6 def yyy = x.substring(0)
7 yyy //TYPE:java.lang.String
8 yyy //TYPE:com.foo.Oops
9
```

Invalid inferred type. Expected: com.foo.Oops Actual: java.lang.String

This feature is still experimental and we are looking for feedback on it. If you find this feature useful, if you have any suggestions for it, or if you have found problems with it, please contact us on mailing list to discuss.

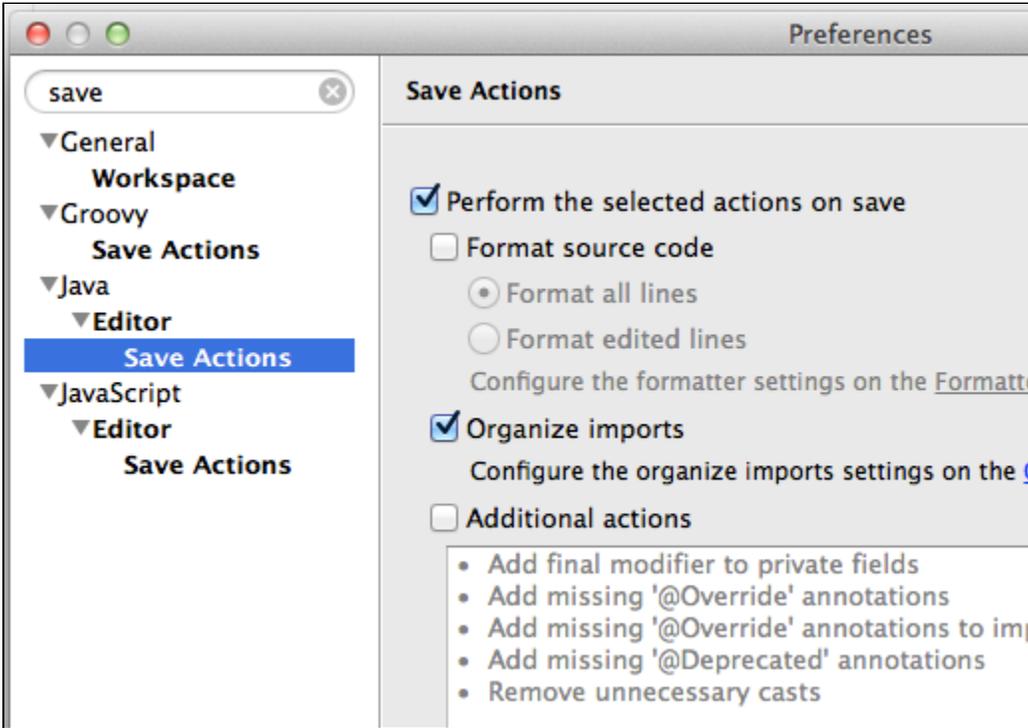
#### *Remove trailing whitespace and semi-colons save actions*

It is now possible to configure Groovy-Eclipse to automatically remove trailing whitespace and semi-colons on file-save. Go to Preferences -> Groovy -> Save actions to enable it:



It is also necessary to enable save actions for Java editors, which you can do by going to Preferences ->

Java -> Editor -> Save actions:



Enabling the save action will convert this file:

```
Script.groovy - Flar/src
1
2
3 def x = "Help me";
4 x << " more stuff" ;
5 println x
```

into this file:

```
Script.groovy - Flar/src
1
2
3 def x = "Help me"
4 x << " more stuff"
5 println x
```

after a save.

#### *Editing support for Groovy files outside of the build path*

Scripts that are not placed on the build path now provide many of the standard editing features that is used by scripts on the build path, such as content assist, type inferencing, and mark occurrences. The classpath for the script is assumed to be the same build path as the one for the rest of the project.

Note that missing import references will not be reported, and references in scripts will not show in search results, nor will refactoring work in these files.

### Fix import statements after move/copy

Import statements are now properly cleaned up after a move or copy of a Groovy file to a new location. See [GRECLI PSE-682](#) for more information.

### More precise searching for overloaded methods

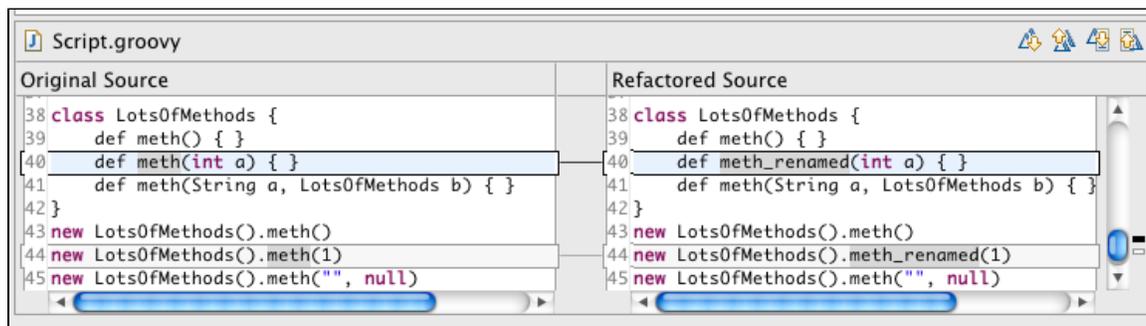
The number of parameters of method declarations are now used to more precisely determine the actual declaration of a method reference:

```
class LotsOfMethods {
    def meth() { }
    def meth(int a) { }
    def meth(String a, LotsOfMethods b) { }
}
new LotsOfMethods().meth()
new LotsOfMethods().meth(1)
new LotsOfMethods().meth("", null)
```

See [GRECLIPSE-1138](#) for more information.

### More precise rename refactoring of overloaded methods

This new technique for method searching is used to ensure that overloaded methods do not interfere with rename refactoring:



### Compatibility

Groovy-Eclipse 2.5.2 includes Groovy 1.7.10 by default. [Groovy 1.8.2 can be optionally installed](#). This version of Groovy-Eclipse is recommended to be installed on STS 2.7.2, 2.8.0.M2, Eclipse 3.7.0, 3.7.1, 3.6.0, 3.6.1, or 3.6.2. There are different update sites for Groovy-Eclipse targeting Eclipse 3.6 and 3.7 (see above).

### Bug fixes

We have fixed over 100 bugs for this release. See the details on our [issue tracker](#).

### Shout outs

We've received quite a bit of help with this release from the community, especially with the maven support. So, a special thank you to:

- Travis Schneeberger for work on maven integration, helping to test snapshot versions, and providing suggestions on how to fix bugs.
- Bob Tiernay for being thorough and checking through all of our type inferencing for DefaultGroovyMethods for anything we missed (and there was a lot).
- Rene Scheibe for his work on the *Save action* to remove whitespace and semi-colons.
- Benson Margulies for helping to migrate the Groovy-Eclipse m2e configurator to support m2e version 1.0. I

could not have done this one alone.

And of course, thank you to everyone else who raised bugs on jira and asked questions on the mailing lists.

#### *What's next?*

We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up!

### **Groovy-Eclipse 2.6.0 New and Noteworthy**

The Groovy-Eclipse team is proud to announce the release of Groovy-Eclipse 2.6.0. As usual, we have some exciting new features in this release, including quick fixes/quick assists, improved content assist, and improved search and refactoring. Groovy-Eclipse 2.6.0 includes 1.8.4 support as an optional add-on. Once Grails 2.0 is released officially, we will make Groovy 1.8.x the default compiler in Groovy-Eclipse.

You can use the following update site to install this release:

#### **Groovy-Eclipse update site for Eclipse 3.7 and 3.6**

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

If you want the Groovy 1.8 compiler, you must explicitly include it by checking the "Extra compilers" category.

And a zipped version of the update site is available at:

#### **Zipped Groovy-Eclipse update site for Eclipse 3.7 and 3.6**

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.6.0.xx-20111212-0900-e37-RELEASE.zip>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.6.0.xx-20111212-0800-e36-RELEASE.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

#### **Outline**

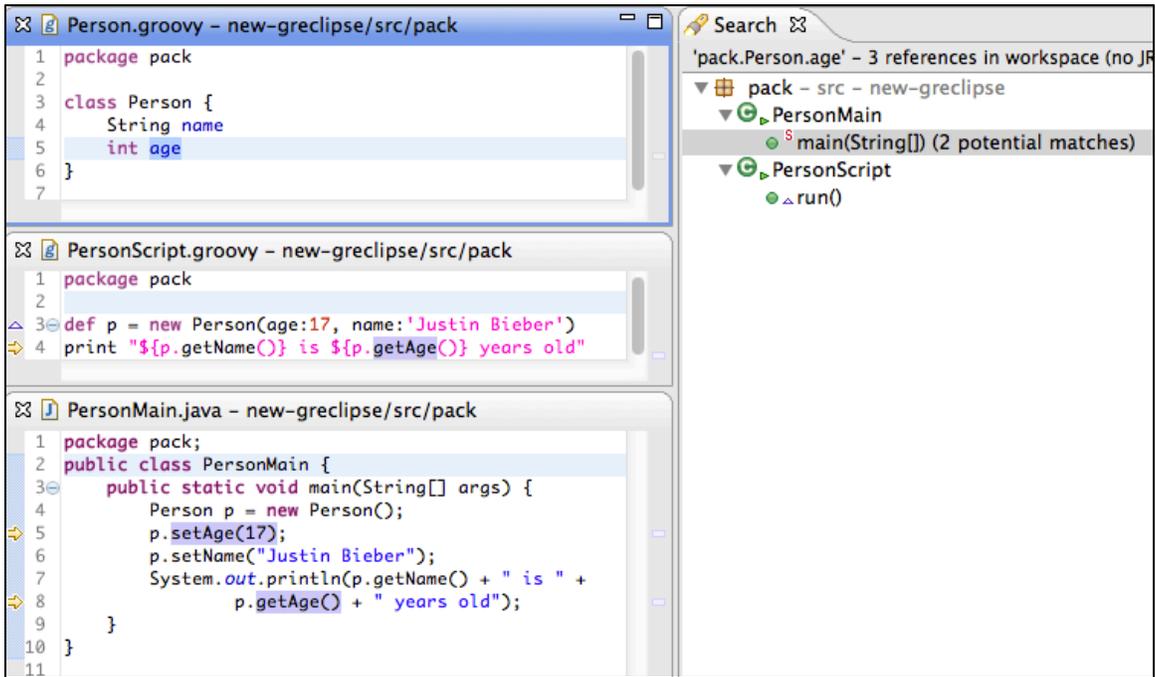
- [Search and refactoring](#)
  - [Search and refactoring of generated getters, setters, and properties](#)
  - [Better default parameter support](#)
  - [Better extract local variable refactoring](#)
- [Parser Recovery](#)

- [When a trailing close paren is missing](#)
- [When working on if conditions](#)
- [Content assist](#)
  - [Suppressing default groovy methods \(DGMs\) from content assist](#)
  - [Support for named arguments in constructors](#)
  - [Better content assist for methods with closure arguments](#)
  - [Better content assist in closures](#)
- [Quick fixes and Quick assists](#)
  - [Add unimplemented methods/Make class abstract Quick fixes](#)
  - [Add groovy classpath container Quick fix](#)
  - [Convert to closure and convert to method quick assists](#)
  - [Convert to single line/multi line string](#)
  - [Remove unnecessary semi-colons](#)
- [Compiler Integration](#)
  - [Groovy 1.8.4 support](#)
  - [Better Grab support](#)
  - [More binary dependencies](#)
- [Maven integration](#)
- [Compatibility](#)
- [Bug fixes](#)
- [Shout outs](#)
- [What's next?](#)

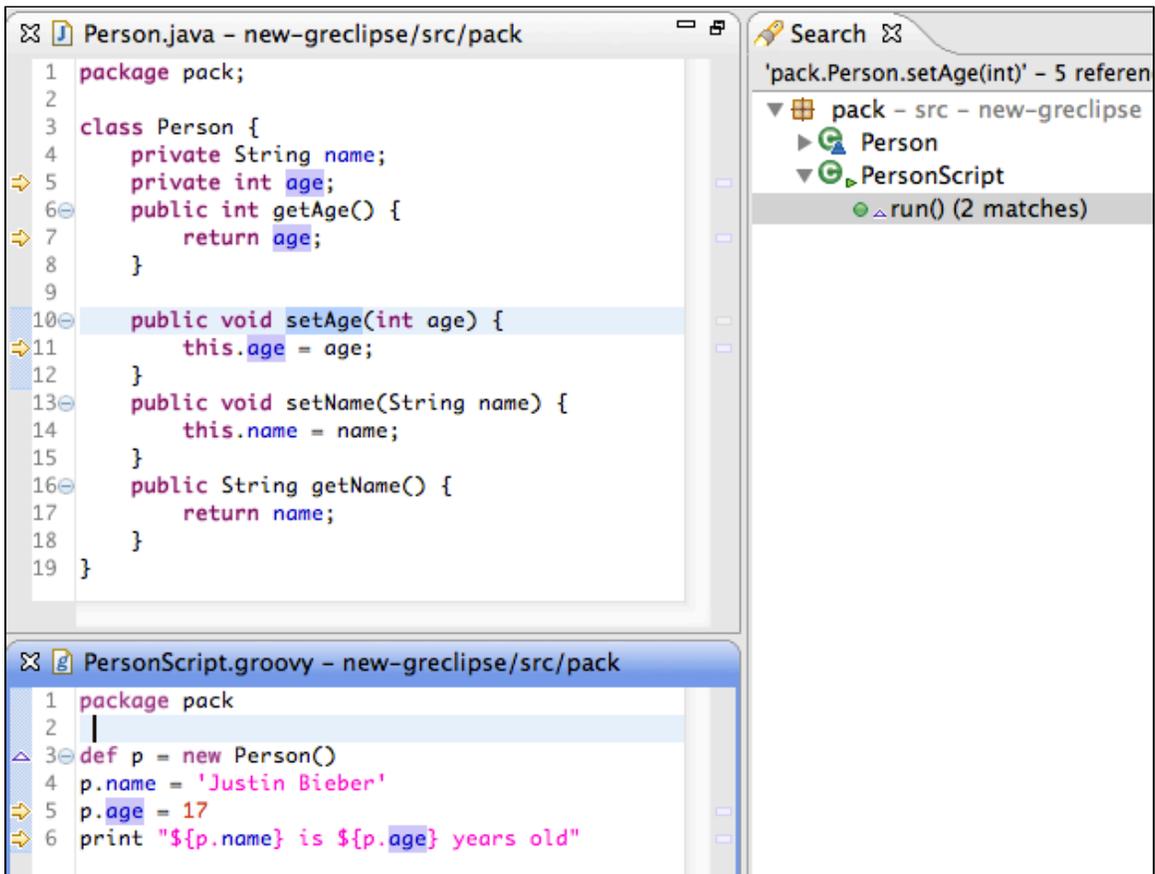
#### *Search and refactoring*

## **Search and refactoring of generated getters, setters, and properties**

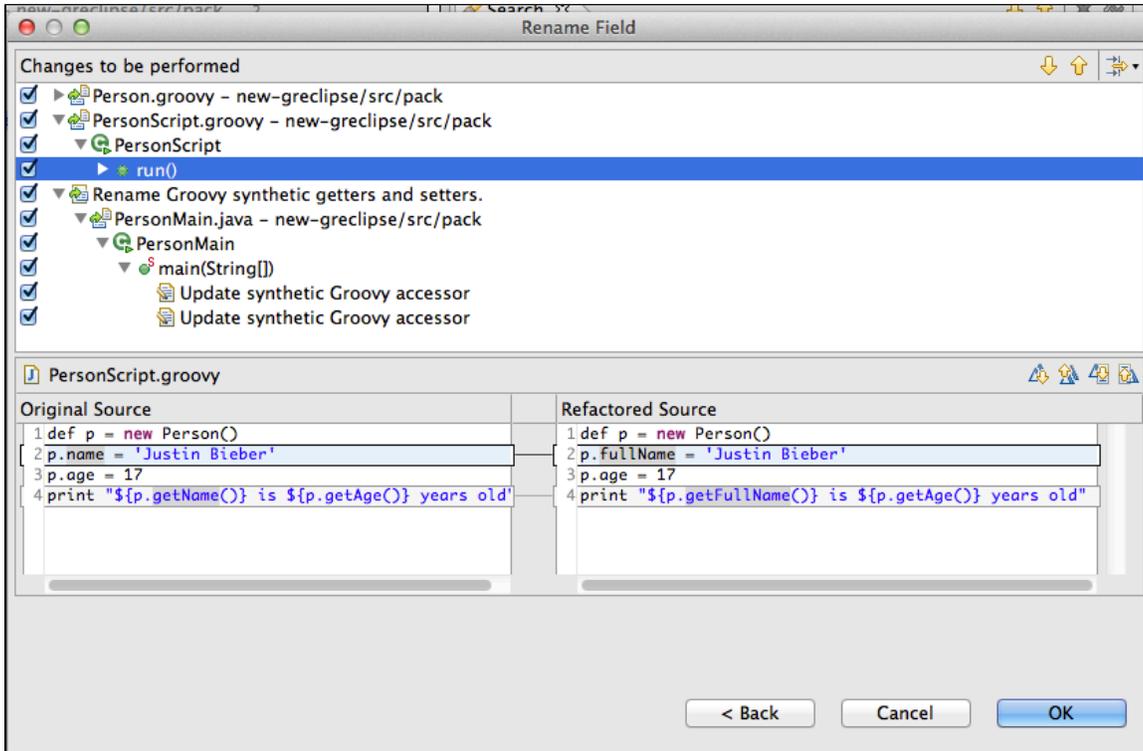
Groovy-Eclipse now allows you to search for references to generated getters, setters and properties. For example, searching for references to a Groovy property (such as age in this example) will find all references to `getAge` and `setAge` in both Java and Groovy files:



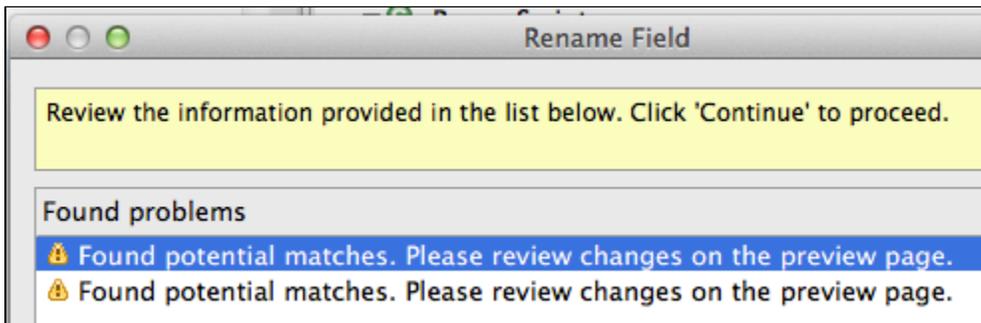
Similarly, references Java to getters and setters can be found inside of Groovy files even when they are referenced as properties. In this example, the class `Person` is defined in Java with explicit getters and setters. Searching for references on `setAge` will return references to the generated `age` property in the Groovy



This also works for refactoring. As in the first example, when `Person` is defined as a Groovy class, we can rename `name` to `fullName` and the synthetic getters and setters will be renamed in both Java and Groovy files:



Note that you will sometimes see warnings during refactoring like this:

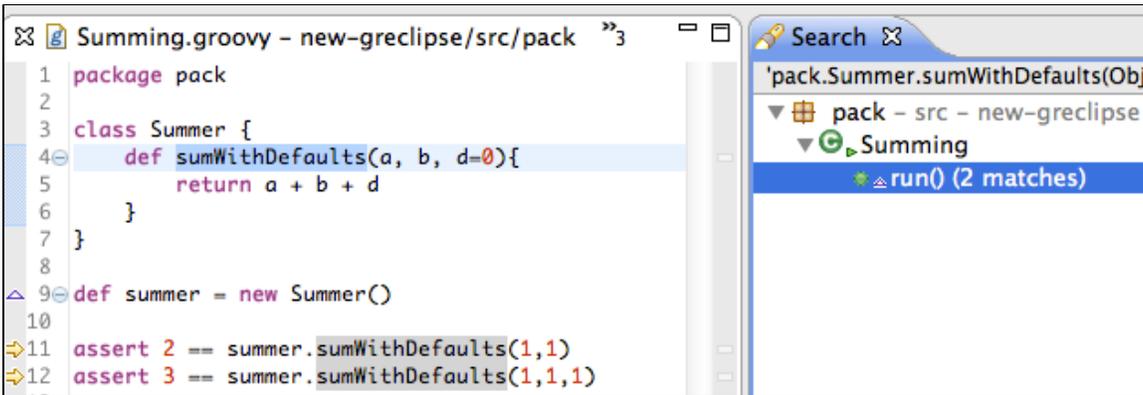


This warning comes about since some of the synthetic references in Java files cannot be determined to be precise by the Java search engine. By the nature of the language, refactoring of Groovy code can never be as precise as Java code is. It is always recommended to view the preview page before executing a refactoring.

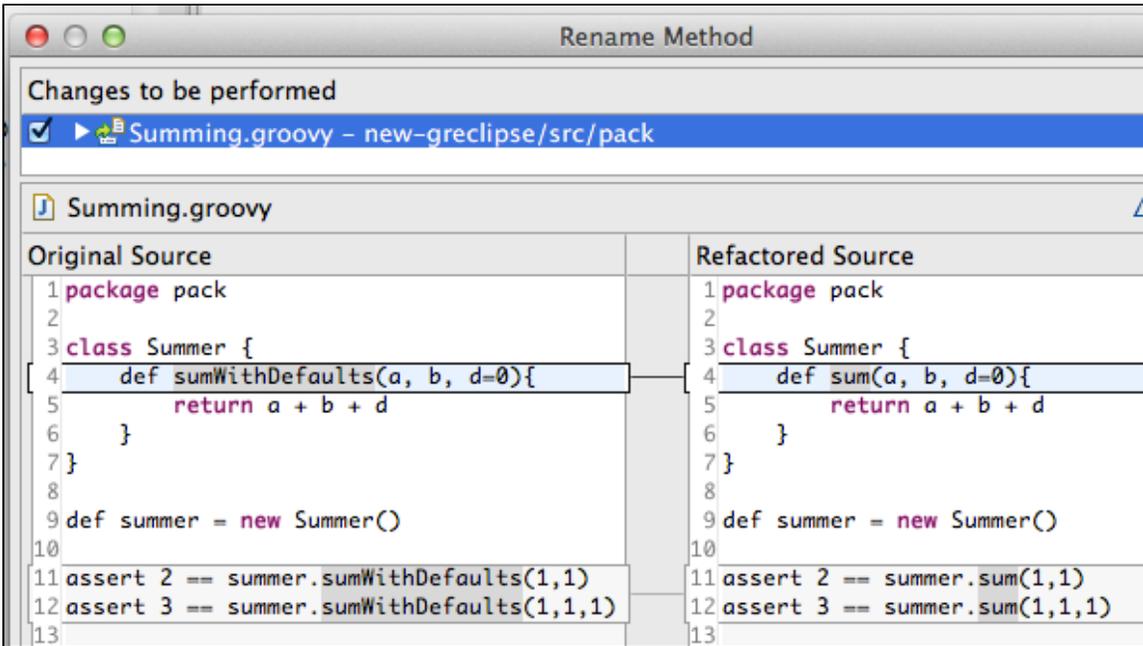
For more information on this feature, see issues [GRECLIPSE-1204](#), [GRECLIPSE-1010](#), and [GRECLIPSE-1205](#).

## Better default parameter support

For this release, we have done significant work with searching for and refactoring methods with default parameters. Now, searching for references to a method that has default parameters will locate all references to that method, regardless of how many parameters that reference uses:



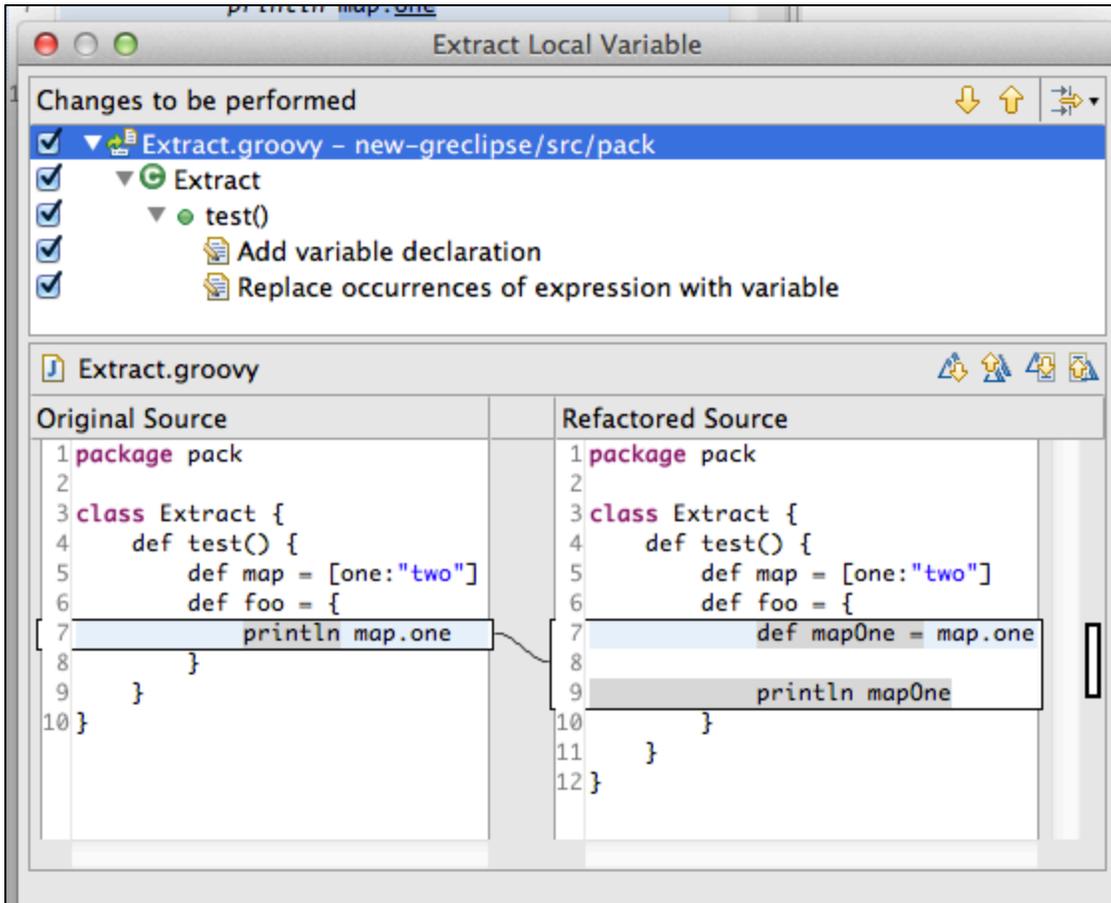
Similarly, rename refactor will correctly rename all references no matter how many parameters are used:



For more information on this feature, see issues [GRECLIPSE-1255](#) and [GRECLIPSE-1233](#).

## Better extract local variable refactoring

*Extract local variable* is now more precise as to where the variable is extracted to. Now, the variable is placed in the statement immediately preceding the variable's first use. See this example, where `map.one` is extracted to a variable and placed inside the enclosing closure.



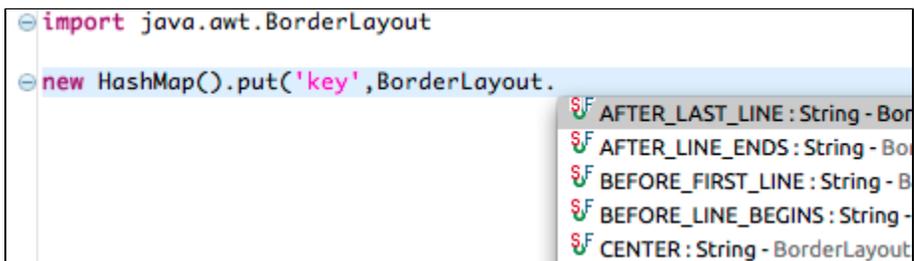
See issue [GRECLIPSE-907](#).

#### Parser Recovery

Further recovery enhancements have been made to the Groovy Parser. This enables it to cope better with malformed code and that enables content assist to offer suggestions in more places than before. Here are a couple of examples of the 2.6.0 improvements:

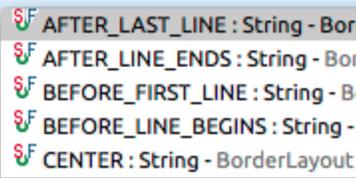
## When a trailing close paren is missing

These two situations show that correct content assist options are available even though there is a missing close paren:



```
import java.awt.BorderLayout

String message = ('foo' + BorderLayout.
```

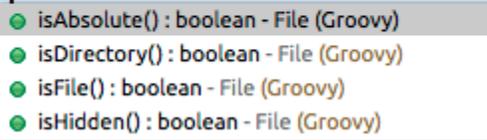


## When working on if conditions

It is now possible to work on `if` conditions without the then block `{...}` being defined

```
File f = new File("abc")

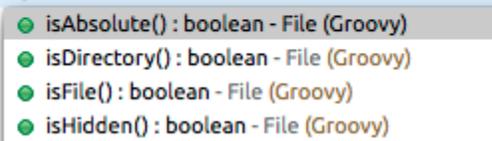
if (f.is
```



It will even work if the trailing paren of the `if` condition isn't specified:

```
File f = new File("abc")

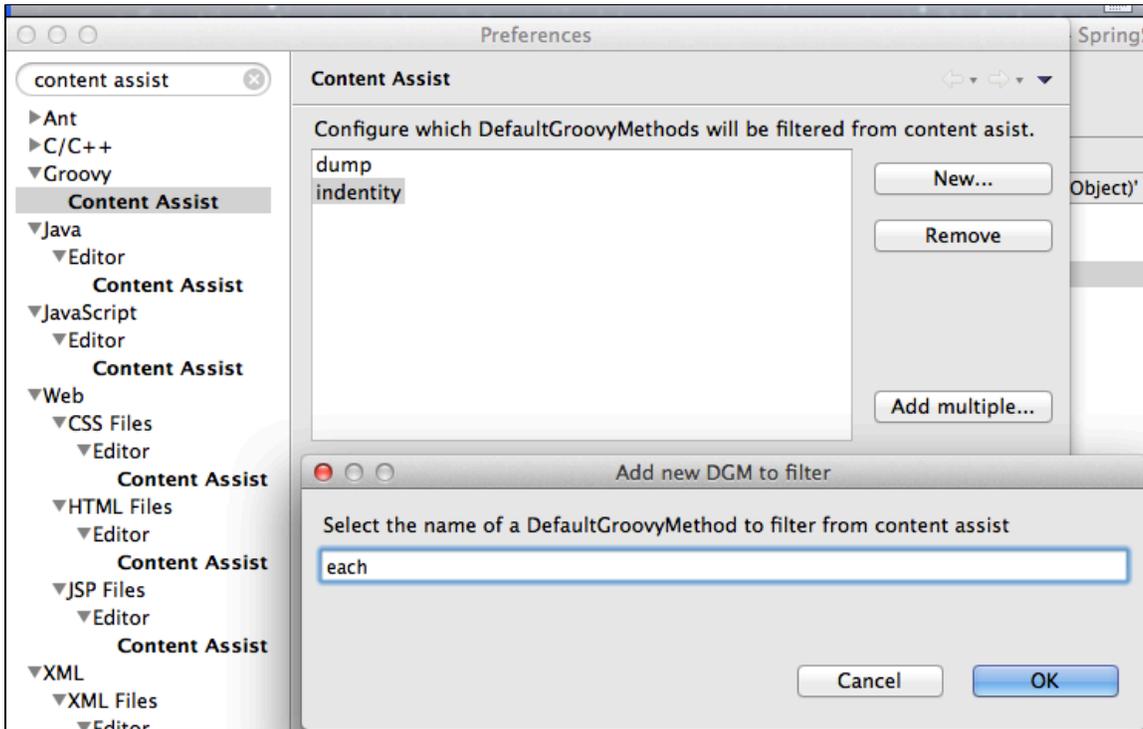
if (f.is
```



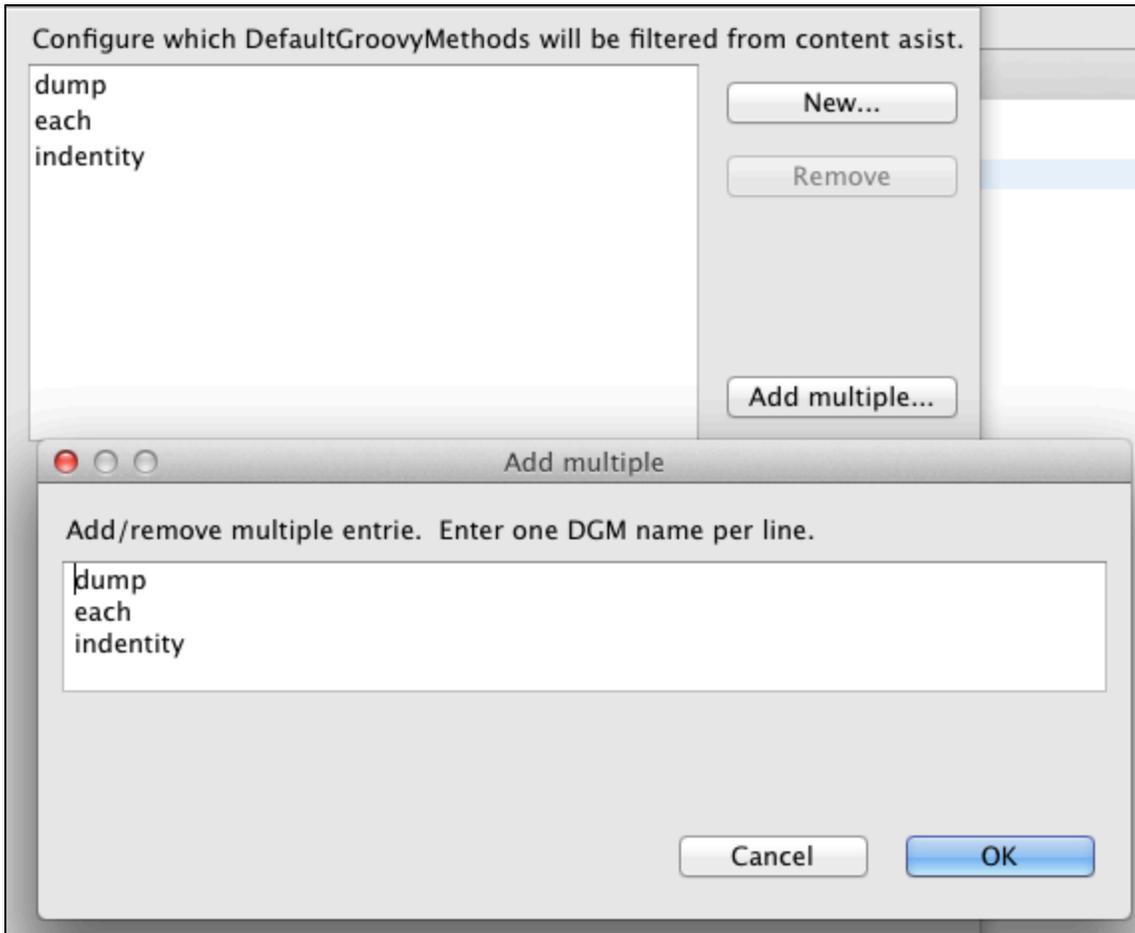
*Content assist*

## Suppressing default groovy methods (DGMs) from content assist

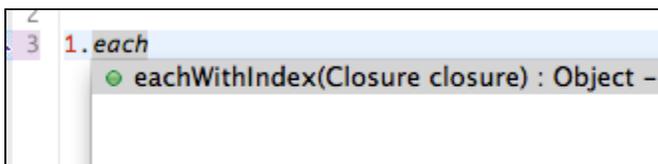
It is now possible to selectively suppress DGMs from cluttering content assist. There is a new Preferences -> Groovy -> Content Assist preferences page:



This page contains a list of names of DGMs to be filtered from content assist. You can add and remove names individually by clicking on *New...* and *Remove*. Alternatively, you can edit the entire list at once by clicking on *Add multiple....* This opens a dialog box with a multi-line text editor where you can easily add and remove multiple entries at once. See here:



As expected, when in the editor, entries that have been suppressed no longer appear in content assist. In this case, each has been filtered, but eachWithIndex has not:



For more information, see issue [GRECLIPSE-1182](#).

## Support for named arguments in constructors

When a Groovy class has no explicit constructor, it is possible to build an instance to the class using named arguments as described in [Groovy Beans](#). Groovy-Eclipse now provides content assist support for this kind of constructor call. In the following example code, performing content assist inside of the parens of the constructor call will bring up all remaining available



And, like all parameters applied in content assist, Groovy-Eclipse guesses some likely values for the parameter:

```
4 class Customer {
5     Integer id
6     String name
7     Date dob
8
9     static void main(args) {
10        def hisName = "Justin Bieber"
11        def myName = "Andrew Eisenberg"
12        def customer = new Customer(dob: new Date(), name: )
13        println("Hello ${customer.name}")
14    }
15 }
```



Note: named parameter content assist will only be available if there is no prefix. I.e., it will be available here:

```
new Customer( /**/ )
```

but not here:

```
new Customer(na/**/ )
```

(/\*\*/ is the location where content assist is invoked)

For more information, see [GRECLIPSE-1228](#).

## Better content assist for methods with closure arguments

When performing content assist on a method and the last parameter is a closure, the proposal will be applied with an opening curly "{", but no closing curly as here:

```
3 def x = 9
4 def method(first, Closure second) { /* ... */ }
5 method(x) {
6 }
```

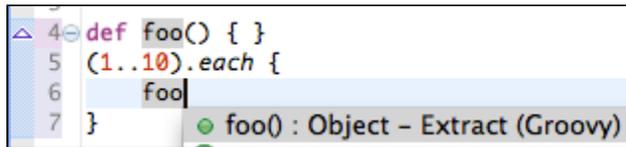
As a user, you can choose to delete this and add your own content, or you can press enter and the closure will be completed for you:

```
3 def x = 9
4 def method(first, Closure second) { /* ... */ }
5 method(x) {
6     |
7 }
```

For more information, see issue [GRECLIPSE-1232](#).

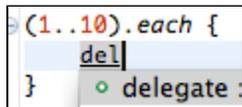
## Better content assist in closures

When inside of a closure, methods defined in the enclosing class are now available in content assist ([GRECLIPSE-114](#)):



```
4 def foo() { }
5 (1..10).each {
6     foo
7 }
```

Similarly, the fields `closure` and `owner` are now available in content assist when inside a closure ([GRECLIPSE-1267](#)):



```
(1..10).each {
}
```

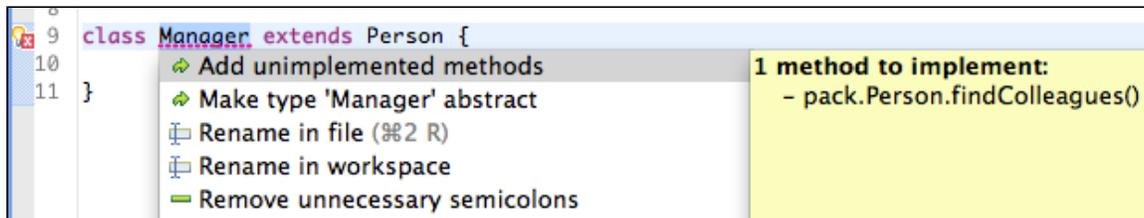
### Quick fixes and Quick assists

Thanks to some help at a Groovy-Eclipse hackathon, we now have quite a few quick fixes and quick assists. Quick fixes are available based on a particular error marker in the editor. And quick assists are available based on the structure of the syntax tree.

Both *quick fixes* and *quick assists* can be invoked by pressing CTRL-1 (or CMD-1 in Mac) on a selection in the editor.

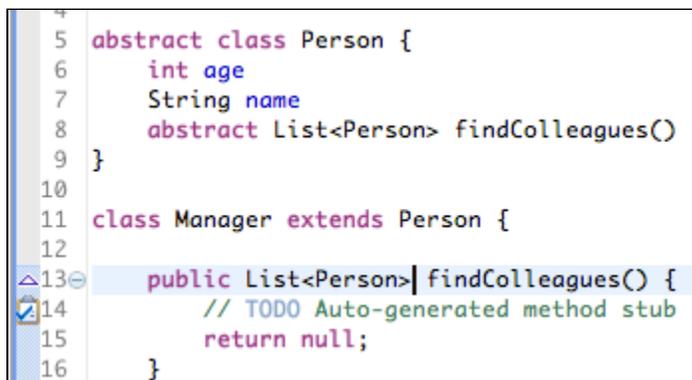
## Add unimplemented methods/Make class abstract Quick fixes

When a concrete base class implements an abstract super class with abstract methods, there are two quick fixes available:



```
9 class Manager extends Person {
10
11 }
```

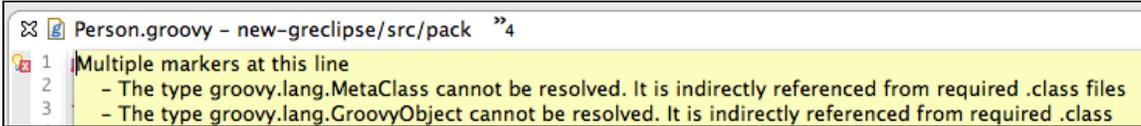
- Make class abstract, which adds the "abstract" modifier to the sub-class
- Add unimplemented methods, which adds method stubs for all unimplemented methods, as shown here:



```
5 abstract class Person {
6     int age
7     String name
8     abstract List<Person> findColleagues()
9 }
10
11 class Manager extends Person {
12
13     public List<Person> findColleagues() {
14         // TODO Auto-generated method stub
15         return null;
16     }
```

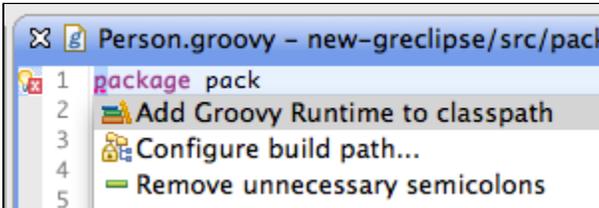
# Add groovy classpath container Quick fix

When errors like these are seen on the first line of the editor, it means that the Groovy libraries cannot be found:



```
Person.groovy - new-greclipse/src/pack  »4
1 | Multiple markers at this line
2 |   - The type groovy.lang.MetaClass cannot be resolved. It is indirectly referenced from required .class files
3 |   - The type groovy.lang.GroovyObject cannot be resolved. It is indirectly referenced from required .class
```

There is now a quickfix that will automatically add the Groovy classpath container to the project:

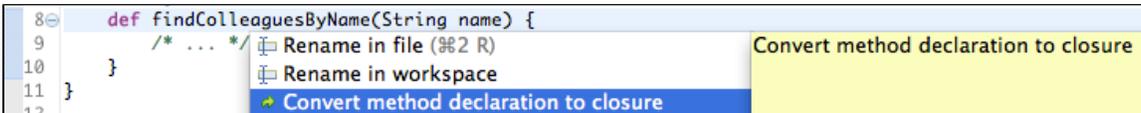


```
Person.groovy - new-greclipse/src/pack
1 | package pack
2 | Add Groovy Runtime to classpath
3 | Configure build path...
4 |
5 | Remove unnecessary semicolons
```

# Convert to closure and convert to method quick assists

This pair of quick assists can be invoked when inside of a method or closure declaration (the closure declaration must be assigned to a field), and allows a quick conversion between the two.

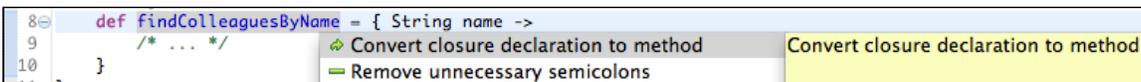
For example, this method declaration:



```
8 | def findColleaguesByName(String name) {
9 |     /* ... */
10 | }
11 |
12 |
```

Quick assist menu options: Rename in file (%%2 R), Rename in workspace, Convert method declaration to closure.

is converted into this closure:



```
8 | def findColleaguesByName = { String name ->
9 |     /* ... */
10 | }
11 |
```

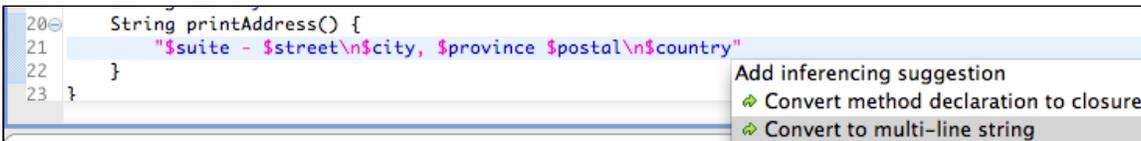
Quick assist menu options: Convert closure declaration to method, Remove unnecessary semicolons.

And the closure can be converted back into the method declaration.

# Convert to single line/multi line string

This pair of quick assists converts between single and multi line strings. When converting between string variants, newlines, tabs, etc are properly (un-)escaped.

Here, a single line string is converted into a multiline string:



```
20 | String printAddress() {
21 |     "$suite - $street\n$city, $province $postal\n$country"
22 | }
23 |
```

Quick assist menu options: Add inferencing suggestion, Convert method declaration to closure, Convert to multi-line string.

And, it can be converted back:

```
20 String printAddress() {
21     ""$suite - $street
22     $city, $province $postal
23     $country""
24 }
25 }
```

Add inferencing suggestion  
Convert method declaration to closure  
Convert to single-line string

## Remove unnecessary semi-colons

This quick fix will remove all unnecessary semi-colons from a Groovy file. For example, this file:

```
package pack;
for (i = 0; i < 5; i++) {
    println i; println i;
}
```

Remove unnecessary semicolons

Will have all unnecessary semi-colons removed, but required ones will remain:

```
1 package pack
2 for (i = 0; i < 5; i++) {
3     println i; println i
4 }
```

### Compiler Integration

## Groovy 1.8.4 support

Although Groovy-Eclipse still ships with the 1.7.10 compiler by default, 1.8.4 is available as an optional add-on. Groovy-Eclipse always ships with the compiler version that is compatible with the current production release of Grails. Since Grails 2.0 is not available in production yet, we do not ship the Groovy 1.8.x compiler by default.

## Better Grab support

Groovy-Eclipse is careful to not allow AST transforms to run during reconciling. Reconciling is the special compile done on the editor contents whilst they are actively being worked on, prior to a save. AST transforms are prevented from running because they can damage source locations/etc that in turn damage other editor features (breaking search/refactoring/etc). However, in 2.6.0 the Grab transformation is being allowed to run since it doesn't modify the code structure but instead just pulls in jars to be on the compilation classpath. This should mean that when working on scripts/etc that are Grab'ing dependencies, there should be no errors in the editor view.

## More binary dependencies

The Groovy-Eclipse classpath container now includes the ivy, jline, and bsf jars by default. Even though these libraries are not typically used directly in user code, including them on the classpath will help with searching for binary references. See [GRECLIPSE-1211](#).

### Maven integration

There is now better ordering on the Java classpath of Groovy source folders when importing maven projects that user Groovy into Eclipse and STS. See [GRECLIPSE-1222](#).

Also, we have updated the [Groovy-Eclipse compiler plugin for Maven](#) docs to be explicit about how to configure your

extra source folders.

### ***Compatibility***

Groovy-Eclipse 2.6.0 includes Groovy 1.7.10 by default. [Groovy 1.8.4 can be optionally installed](#). This version of Groovy-Eclipse is recommended to be installed on STS 2.8.1 or 2.9.0.M1, Eclipse 3.7.1 or 3.6.2. There are different update sites for Groovy-Eclipse targeting Eclipse 3.6 and 3.7 (see above).

### ***Bug fixes***

We have fixed over 70 bugs for this release. See the details on our [issue tracker](#).

### ***Shout outs***

We had a successful [Eclipse hackathon](#) this November in Vancouver, B.C. and we received a number of quality patches from external contributors:

- Renato Garcia on [GRECLIPSE-1269](#), [GRECLIPSE-1250](#)<http://jira.codehaus.org/browse/GRECLIPSE-1250>
- Adrian Kuhn on [GRECLIPSE-1255](#)
- Nick Sawadsky on [GRECLIPSE-1252](#)
- Curtis on [GRECLIPSE-848](#)

Thanks for your work. And of course, thank you to everyone else who contributed in other ways by raising bugs on jira or asking questions on the mailing lists.

### ***What's next?***

We are tentatively planning a 2.6.1 release in February. You can take a look at all issues we are [planning to fix for this release](#).

We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up!

## **UCOSP**

This page describes the Groovy-Eclipse projects for [UCOSP](#) (the undergraduate capstone open source projects).

This list does not describe the projects in detail, but is meant as a place to start. Details can only be filled in as the projects move along.

### ***Project 1: Code formatter***

The Groovy-Eclipse code formatter has historically been one of the messiest parts of the code base. In the last couple of years, though we have patched together things so that it works much better than before. However, we have gotten ourselves into a state where the existing code is too tangled and fixing bugs becomes increasingly more difficult. There are many open formatting bugs, but we have not had the resources to fix them.

With this project, there are two possibilities:

1. Work with the existing formatter and incrementally improve, fixing one bug at a time. The risk here is that some bugs cannot be fixed without starting from scratch
2. Start from scratch with an entirely new formatter and a new approach. The risk with this approach is that it could be difficult to build a new formatter and there is no guarantee that the new formatter would behave better than the original.

This is a large project and quite difficult. But, it is also very important. Fixing formatting-related bugs is highly requested. Since this project is more complicated, it may be worthwhile that two students work on this together.

See [here for a list of open formatter issues](#).

#### *Project 2: Quick fixes and quick assists*

With Groovy-Eclipse, we strive to have the same level of functionality as you get from the JDT. We are not there yet. Part of what we need is good quick-fixes and quick assists. Quick fixes are small code manipulations that are associated with errors. For example, classes that do not implement all methods of an interface have a quick fix associated with the error marker. Quick assists however, are associated with syntax. Pressing Ctrl-1 or Cmd-1 will bring up a panel that shows all available quick assists at the given location.

Groovy-Eclipse already implements several quick fixes and quick assists, but it would be great to have more. The nice thing about this project is that each quick assist/fix is largely self-contained and you can look at existing implementations to see how to build new ones. Quick fixes/assists are nice to have and provide a bit of polish on Groovy-Eclipse that is currently missing.

See [here for a list of open quick assist issues](#).

#### *Project 3: Refactoring*

Groovy-Eclipse implements some of the basic refactorings: rename, extract local variable, extract constant, and extract method. To be a full fledged language IDE, we should be offering a wider range of refactorings, including (but not limited to): introduce parameter, change method signature and convert local variable to field. Also, some Groovy-specific refactorings like extract to category, push from category, and convert to explicit typing.

This project is more difficult than implementing quick fixes/assists since each refactoring requires significant custom work, but there are existing refactorings on which to base the new work. See [here for a list of open refactoring issues](#).

#### *Project 4: New AST Viewer*

The AST viewer in Groovy-Eclipse displays the entire AST of a file in tree form. Users can navigate the tree and this is helpful for people writing AST transforms and people who just want to learn about how Groovy works. This implementation is loosely based on the AST viewer that ships with the Groovy console. However, the viewer in Eclipse is messier and provides less functionality. This is a partial list of required improvements:

1. Filtering of tree nodes that are uninteresting.
2. Tree collapses after every edit of the file
3. Selecting a node in the tree should select the appropriate location in the file (and vice versa)

This is a low priority project since this feature is really only useful to a small set of users. However, this project may be interesting as a backup project.

### **Groovy-Eclipse 2.6.1 New and Noteworthy**

The Groovy-Eclipse team is proud to announce the release of Groovy-Eclipse 2.6.1. Since this is the first release since Grails 2.0 has been available, we have made Groovy 1.8 the default compiler in Groovy-Eclipse. Also, we have started releasing snapshots of our 4.2-based Groovy-Eclipse.

You can use the following update sites to install this release:

### **i Groovy-Eclipse update site for Eclipse 4.2, 3.7, and 3.6**

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/>

For Eclipse 4.2 (this is not a release, but a first snapshot build that we are making available for early adopters of Eclipse Juno):

<http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e4.2/>

If you want the Groovy 1.8 compiler, you must explicitly include it by checking the "Extra compilers" category.

And a zipped version of the update site is available at:

### **i Zipped Groovy-Eclipse update site for Eclipse 4.2, 3.7, and 3.6**

For Eclipse 3.7:

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.6.1.xx-20120301-1000-e37-RELEASE.zip>

For Eclipse 3.6:

<http://dist.springsource.org/release/GRECLIPSE/e3.6/archive-2.6.1.xx-20120301-1300-e36-RELEASE.zip>

For Eclipse 4.2:

<http://dist.codehaus.org/groovy/distributions/greclipse/snapshot/e4.2/old>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions.

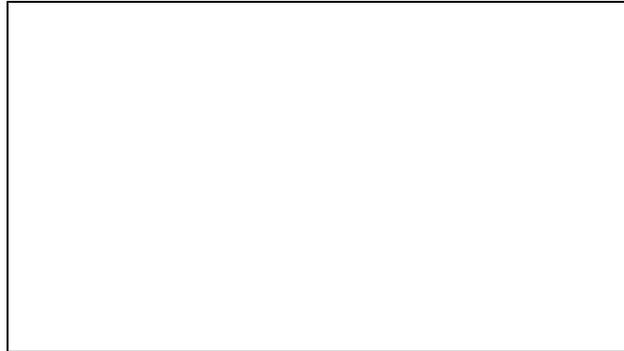
- [Inferencing enhancements](#)
  - [Inferring type of overloaded operators](#)
  - [Better inferencing of list and map literals](#)
  - [Code select and inferencing for static imports](#)
- [Quick fixes and quick assists](#)
  - [Move Package and change package declaration quick fixes](#)
  - [Convert to closure now available from refactoring menu](#)
  - [Split assignment/declaration quickfix](#)
  - [Swap operands quickfix](#)
- [Other improvements](#)
  - [Full Javadoc capability for inferencing suggestions and DSLDs](#)
  - [Hover and navigation in constructors with named arguments](#)
  - [Better content assist for missing methods](#)
  - [Editor option to turn off highlighting for dollar slashy strings](#)
  - [Per-project script folders](#)
- [Compatibility](#)

- [Bug fixes](#)
- [Shout outs](#)
- [What's next?](#)

*Inferencing enhancements*

## Inferring type of overloaded operators

Groovy-Eclipse will now correctly infer the types of overloaded binary and unary operators. For example, in the following screenshot, you can see that `val` is inferred to be a member of the `Tree` class. This is because the inferencing engine has determined that the `+` operation is overloaded and has a return type of `Tree`:



Also, inside of DSLD scripts, method contributions can be used to overload an operator in an editor. Something like this script would have the same effect as above:

```
contribute(currentType('pack.Tree')) {  
    method name: 'plus', params:  
    [other: 'pack.Tree'], type: 'pack.Tree'  
}
```

## Better inferencing of list and map literals

Groovy-Eclipse now uses more precise techniques to infer the types of list and map literals. Previously, the types of list and map literals were determined by the static type of the first element of the collection. Now, as you can see in the screenshot below, Groovy-Eclipse uses the inferred types of the list and map elements to build the type of the collection:



## Code select and inferencing for static imports

We have done significant work around supporting static imports. References to static imports now allow navigation, static imports are now appropriately renamed during refactoring, and they are found during search.

### *Quick fixes and quick assists*

Quick fixes and quick assists are available in the editor by pressing CTRL+1 (or CMD+1 on Mac OSX). Groovy-Eclipse has introduced several new quick fixes and quick assists in 2.6.1.

## Move Package and change package declaration quick fixes

Groovy-Eclipse now shows quick fixes for invalid package declarations. When hovering over an error marker for an invalid package declaration, there are two quick fixes available: move compilation unit, and change package declaration. See below for an example:

---

The behavior is identical to the quick fixes of the same name available in the Java editor. Moving the compilation unit will not only move the file, but also update all appropriate references to the package. Changing the package declaration, will simply change the text at the beginning of the file so that it matches its current directory.

## Convert to closure now available from refactoring menu

The Convert to closure and Convert to method quick assists are now available from the context menu under the Groovy Refactor section:

---

The keybindings are Alt-G F and ALT-G M respectively.

## Split assignment/declaration quickfix

The split assignment and split declaration quickfixes are available on any assignment or declaration expression and works like this:

---

becomes this:

---

## Swap operands quickfix

The swap operands quickfix reorders the left and right sides of binary expressions. This code:

---

becomes this:

---

### *Other improvements*

## Full Javadoc capability for inferencing suggestions and DSLDs

It is now possible to use Javadoc tags inside of inferencing suggestions. In the inferencing suggestion window (available from Preferences -> Groovy -> Inferencing Suggestions, or from a quick assist in the editor CTRL+1), you can insert full Javadoc comments inside of the *Doc* text box.

**Inferencing Suggestion** 

Edit a Groovy inferencing suggestion

Name:

Declaring Type:

Type:

Is static

Property  Method

Name	Type
data	java.lang.String
depth	int

Use named arguments

Doc:

```
Analyzes some data to the given depth.
@author Andrew Eisenberg
@since 2.6.1
@param data the data to analyze
@param depth the depth to analyze to
@returns the result of analyzing the data.
```

Which gets displayed in hovers as this:

```
def analyzer = new Analyzer()
String results = analyzer.analyzeThis(getData(), 3)
```

**String Analyzer.analyzeThis(String data, int depth)**  
Analyzes some data to the given depth.  
**Parameters:**  
    **data** the data to analyze  
    **depth** the depth to analyze to  
**Author:**  
    Andrew Eisenberg  
**Since:**  
    2.6.1  
**@returns**  
    the result of analyzing the data.

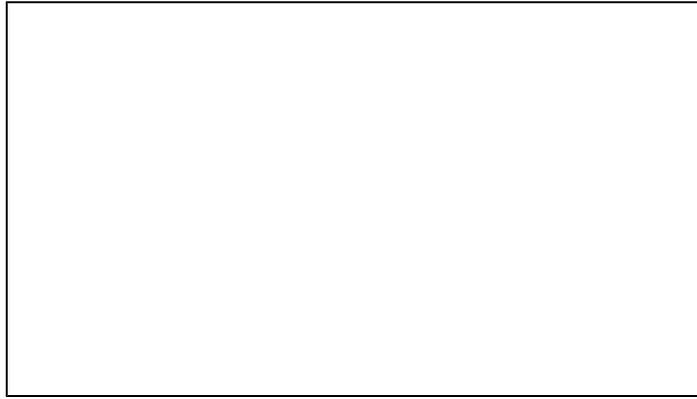
Similarly, you can use Javadoc tags inside of DSLD doc tags. For example, this gives a similar effect as above:

```
contribute(currentType("Analyzer")) {  
    method name:"analyzeThis", type:String,  
params:[data:String, depth, Integer],  
doc:""  
    Analyzes some data to the given depth.  
    @author Andrew Eisenberg  
    @since 2.6.1  
  
    @param data the data to analyze  
    @param depth the depth to analyze to  
    @return the result of analyzing the data  
""
```

At this point, @link tags are not generating a proper hyperlink.

## Hover and navigation in constructors with named arguments

It is now possible to hover over and navigate to field references used as named parameters in default constructor invocations:



## Better content assist for missing methods

Content assist inside of a class body shows all overridable methods. This has been available since Groovy 2.0.

Now, we have improved this support and the resulting content assist text will appropriately include argument names and types, will organize imports, and add a doc stub inside the method body if configured to do so.

For example, this:

---

Becomes this:

---

## Editor option to turn off highlighting for dollar slashy strings

Groovy 1.8 introduced dollar slashy strings ([link](#)) that allow the specification of multi-line strings like this: `$/ ... /$`. However, some users found problems with files that made heavy use of regular expressions. For example, in this file the space between the `$/` and `/$` is incorrectly interpreted as a multi-line string:

---

It is now possible to disable slashy strings by going to the Preferences --> Groovy -> Editor preferences page:

---

And the file is now highlighted correctly (the underlines are expected).

---

You must close and re-open files before this change comes into effect.

## Per-project script folders

Groovy-Eclipse now allows the specification of script folders on a per-project basis. You can control script folders in the Groovy Compiler project properties page:

---

When clicking on "Enable project specific settings", the script folder settings for an individual project overrides the workspace settings.

Script folders describe locations in your project that contain Groovy scripts. Groovy scripts should not be compiled into .class files and they may or may not be copied to the output folder.

#### **Compatibility**

Groovy-Eclipse 2.6.0 includes Groovy 1.8.6 by default. Groovy 1.7.10 can be enabled optionally. This version of Groovy-Eclipse is recommended to be installed on STS 2.9.0, Eclipse 3.7.2. There is also a 4.2-based Groovy-Eclipse now available. See above for the update site.

#### **Bug fixes**

We have fixed over 100 bugs for this release. See the details on our issue tracker.

#### **Shout outs**

We have 3 students working on Groovy-Eclipse and contributing to our quick fixes. So far, Elvis Martinez, Stephanie Van Dyk, and Daniel Phan have contributed the implementations behind the split assignment and the swap operands quickfixes. Thanks! Looking forward to some more great work

#### **What's next?**

We are tentatively planning a 2.7.0 release in June. You can take a look at all issues we are planning to fix for this release.

We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the mailing list. For any issues you have (or enhancements you would like to see), please raise them in our issue tracker. If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up!

### **Groovy-Eclipse 2.7.0 New and Noteworthy**

The Groovy-Eclipse team is proud to announce the release of Groovy-Eclipse 2.7.0. This is the first release that includes Groovy 2.0 final and that is available on Eclipse 4.2 (Juno). Now that Eclipse 4.2 is released, we will be dropping our Eclipse 3.6 support.

You can use the following update sites to install this release:

#### **Groovy-Eclipse update site for Eclipse 4.2 and 3.7**

For Eclipse 4.2 (Juno):

<http://dist.springsource.org/release/GRECLIPSE/e4.2/>

For Eclipse 3.7 (Indigo):

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

The Groovy 2.0 compiler is not installed by default. If you want to install it, you can do so from the *Extra Compilers* category in the update site.

And a zipped version of the update site is available at:

### **i** Zipped Groovy-Eclipse update site for Eclipse 4.2 and 3.7

For Eclipse 4.2 (Juno):

<http://dist.springsource.org/release/GRECLIPSE/e4.2/archive-2.7.0.xx-20120703-1400-e42-RELEASE.zip>

For Eclipse 3.7 (Indigo):

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.7.0.xx-20120703-1400-e37-RELEASE.zip>

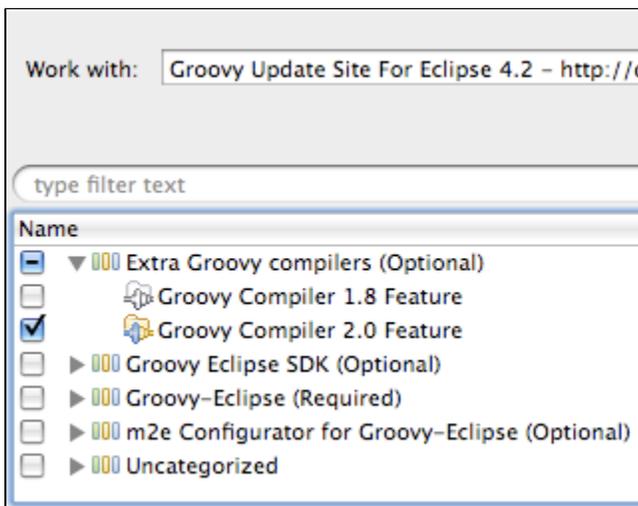
You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions. *Do not unzip into your Eclipse dropins folder.*

- [Groovy 2.0](#)
- [Maven integration with groovy-eclipse-compiler](#)
- [@TypeChecked and @CompileStatic Support](#)
- [Assign statement to new local variable quickfix](#)
- [Convert variable to field](#)
- [Doc hovers with @link and @see](#)
- [Source code to github](#)
- [Compatibility](#)
- [Bug fixes](#)
- [What's next?](#)

#### **Groovy 2.0**

Groovy 2.0 is now available in Groovy-Eclipse as an optional add-on. We have not yet made the compiler default because we make our default compiler compatible with the default compiler of the latest release of Grails. At the time of this release, Grails is at version 2.0.4, which uses Groovy 1.8 as its compiler. When Grails moves to being based on Groovy 2.x, Groovy-Eclipse will move its default compiler as well.

To install Groovy 2.0, go to Help --> Install new Software. In the work with tab, choose the Groovy-Eclipse update site and select the *Extra Compilers* category:



Read [Compiler Switching within Groovy-Eclipse](#) for more information on how to install Groovy 2.0 and how to switch compiler levels.

## Maven integration with groovy-eclipse-compiler

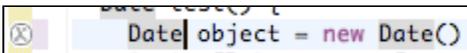
The groovy-eclipse-compiler plugin for maven has just released the 2.7.0-01 version and now uses Groovy 2.0 to compile against by default. You can specify other Groovy compiler levels if required. See [Groovy-Eclipse compiler plugin for Maven](#) for more information.

### @TypeChecked and @CompileStatic Support

With the 2.0 compiler enabled, you can use the `@TypeChecked` and `@CompileStatic` annotations and Groovy-Eclipse compiles with them appropriately. Here is an example from the [What's new in Groovy 2.0](#) article on [InfoQ](#):

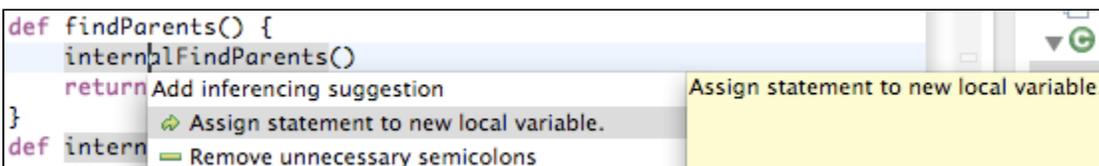
TypeChecked	CompileStatic
<pre>class DateTester {     @TypeChecked     Date test() {         int object = new Date()         String[] letters = ['a', 'b', 'c']         Date aDateVariable = letters[0]         return "today"     } }</pre>	<pre>class DateTester {     @CompileStatic     Date test() {         int object = new Date()         String[] letters = ['a', 'b', 'c']         Date aDateVariable = letters[0]         return "today"     } }</pre>

Also, reconciling (i.e., eager parsing as you type) handles these annotations properly. So, you get proper affordance when changing `int object = new Date()` to `Date object = new Date()`:

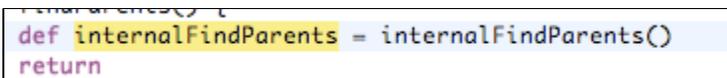


### Assign statement to new local variable quickfix

The Groovy-Eclipse editor now provides a new quickfix for assigning a statement to a local variable. To access the quickfixes, press CTRL/CMD+1. Choose *Assign statement to new local variable*:



And the results look something like this:

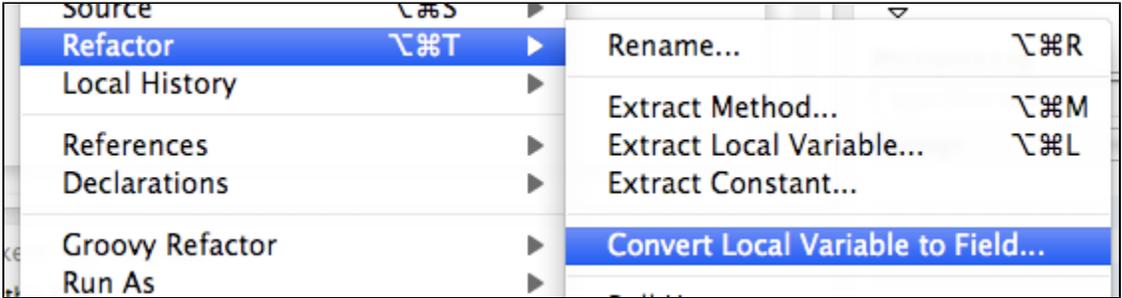


### Convert variable to field

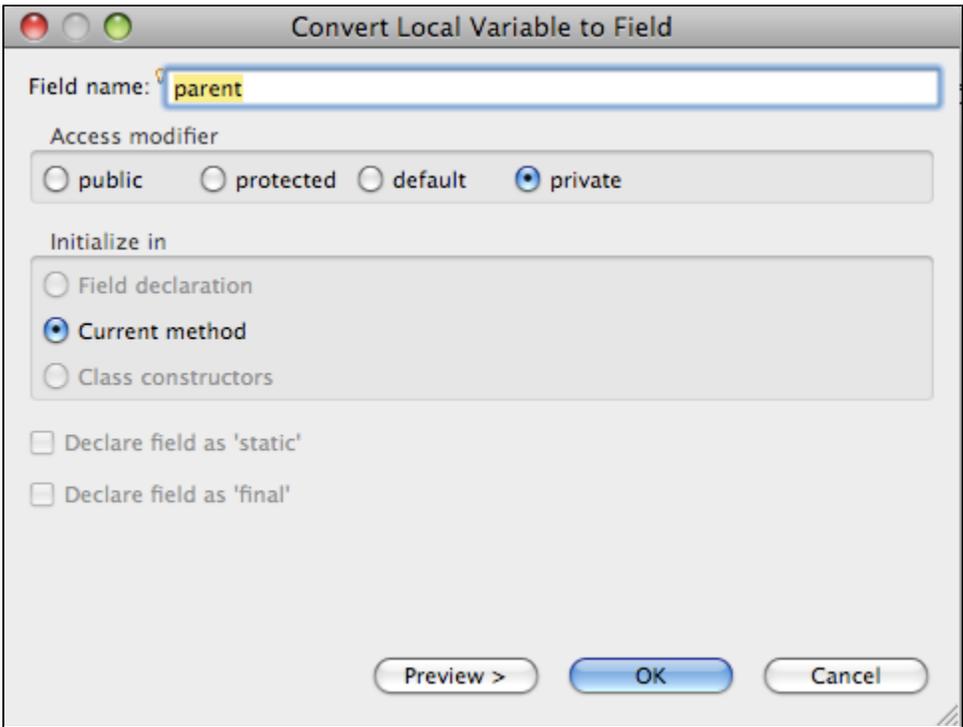
Groovy-Eclipse now provides a refactoring to convert local variables to fields. To access the refactoring, select a local variable:

```
class Person {
    def findParents() {
        Person parent = internalFindParents()
    }
    def internalFindParents() {
```

In the context menu, choose Refactoring --> Convert Local Variable to Field...



A wizard pops up where you can tweak your options and preview the results:



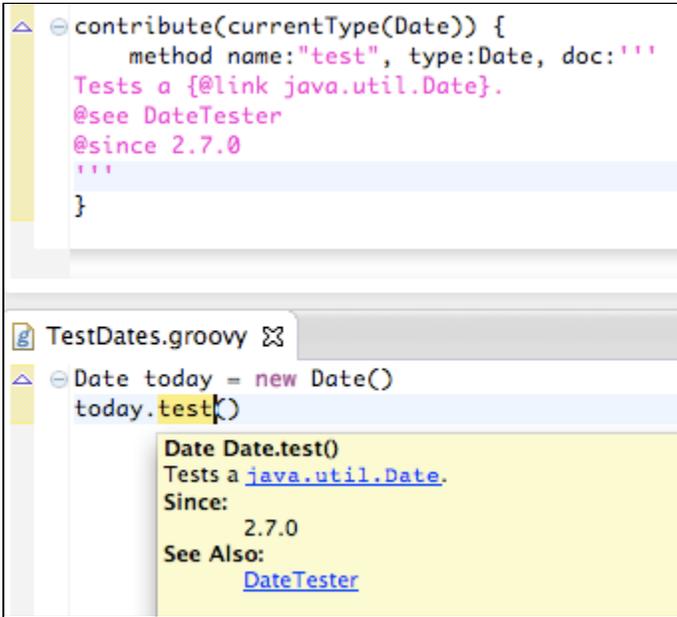
Note that not all options are implemented yet. Unimplemented pieces of the refactoring wizard are grey. After pressing OK, you will see something like this:

```
class Person {
    private Person parent
    def findParents() {
        parent = internalFindParents()
    }
    def internalFindParents() {
```

*Doc hovers with @link and @see*

Doc hovers now respect the @link and @see JavaDoc tags, so you can add them to your DSLDs and users will be

able to follow proper links to other parts of the source code:



```
△ ⊖ contribute(currentType(Date)) {
    method name:"test", type:Date, doc:''
    Tests a {@link java.util.Date}.
    @see DateTester
    @since 2.7.0
    '''
}
```

TestDates.groovy ✖

```
△ ⊖ Date today = new Date()
today.test()
```

Date Date.test()  
Tests a [java.util.Date](#).  
Since:  
2.7.0  
See Also:  
[DateTester](#)

*Source code to github*

Groovy-Eclipse is now on GitHub. Forking and pull requests are appreciated!

 **GitHub Project**

<https://github.com/groovy/groovy-eclipse>

The old SVN repository is no longer valid.

#### *Compatibility*

Groovy-Eclipse 2.7.0 includes Groovy 1.8.6 by default. [Groovy 1.7.10 can be enabled optionally](#), and Groovy 2.0.0 can be installed through the extra compilers section on the update site.

This version of Groovy-Eclipse is recommended to be installed on STS 2.9.2, STS 3.0.0.M3, Eclipse 3.7.2, or Eclipse 4.2. There is also a 4.2-based Groovy-Eclipse now available. See above for the update site.

#### *Bug fixes*

See the details on our [issue tracker](#).

#### *What's next?*

We are planning a 2.8.0 release in October. You can take a look at all issues we are [planning to fix for this release](#).

We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up.

### **Groovy-Eclipse 2.7.1 New and Noteworthy**

The Groovy-Eclipse team is proud to announce the release of Groovy-Eclipse 2.7.1. This is a service refresh of Groovy-Eclipse and it now contains support for the 2.0.4 Groovy compiler as well as Eclipse 4.2.1 and 3.8.1.

You can use the following update sites to install this release:

**i Groovy-Eclipse update site for Eclipse 4.2 and 3.7**

For Eclipse 3.8 and 4.2 (Juno):

<http://dist.springsource.org/release/GRECLIPSE/e4.2/>

For Eclipse 3.7 (Indigo):

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

The Groovy 2.0 compiler is not installed by default. If you want to install it, you can do so from the *Extra Compilers* category in the update site.

And a zipped version of the update site is available at:

**i Zipped Groovy-Eclipse update site for Eclipse 4.2 and 3.7**

For Eclipse 3.8 and 4.2 (Juno):

<http://dist.springsource.org/release/GRECLIPSE/e4.2/archive-2.7.1.xx-20120921-2000-e42RELEASE.zip>

For Eclipse 3.7 (Indigo):

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.7.1.xx-20120921-2000-e37RELEASE.zip>

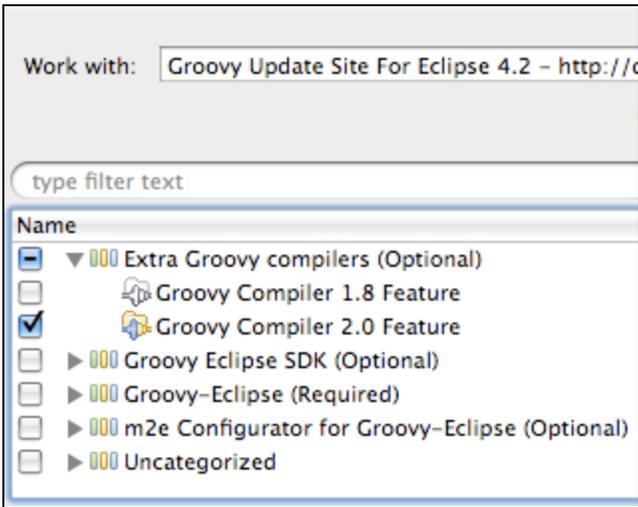
You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions. *Do not unzip into your Eclipse dropins folder.*

- [Groovy 2.0.4](#)
- [Eclipse 4.2.1 and 3.8.1](#)
- [Add project-level warning if compiler levels do not match](#)
- [Better Content assist for Annotation](#)
- [Compatibility](#)
- [Bug fixes](#)
- [What's next?](#)

#### **Groovy 2.0.4**

Groovy 2.0.4 is now available in Groovy-Eclipse as an optional add-on. We have not yet made the compiler default because we make our default compiler compatible with the default compiler of the latest release of Grails. At the time of this release, Grails is at version 2.1.1, which uses Groovy 1.8 as its compiler. When Grails moves to being based on Groovy 2.2.x, Groovy-Eclipse will move its default compiler as well.

To install Groovy 2.0, go to Help --> Install new Software. In the work with tab, choose the Groovy-Eclipse update site and select the *Extra Compilers* category:



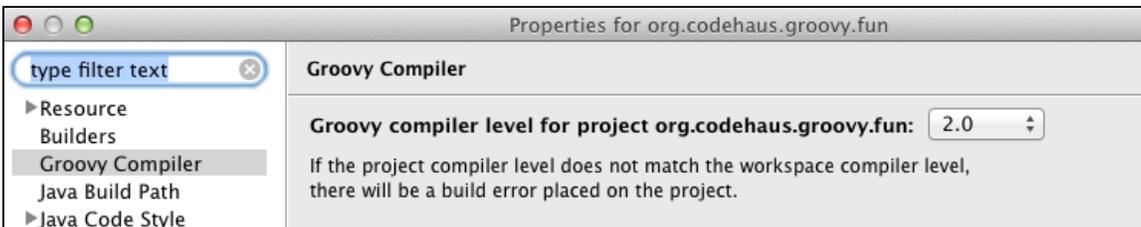
Read [Compiler Switching within Groovy-Eclipse](#) for more information on how to install Groovy 2.0 and how to switch compiler levels.

***Eclipse 4.2.1 and 3.8.1***

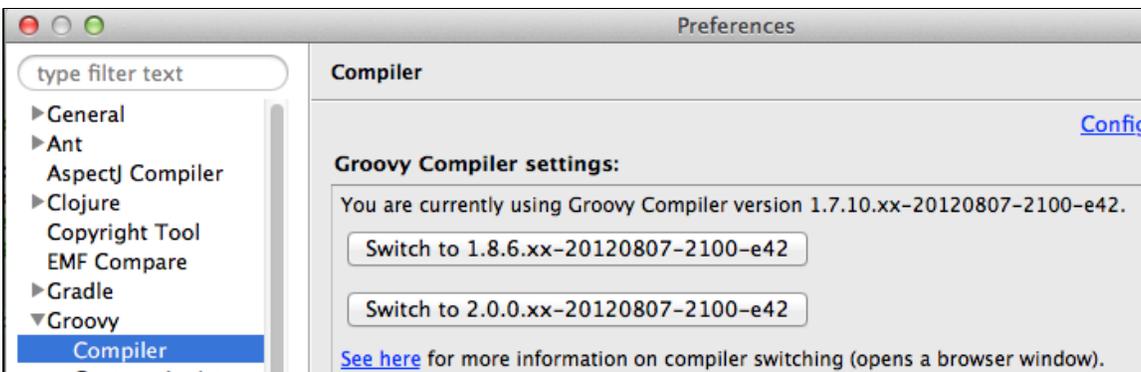
We have updated the JDT feature patch to be based off of JDT version 3.8.1, which is the version used in Eclipse 4.2.1 as well as 3.8.1.

***Add project-level warning if compiler levels do not match***

There is now a project setting that records the last compiler level that compiled it. You can access this setting in the Groovy Compiler settings page for the project:



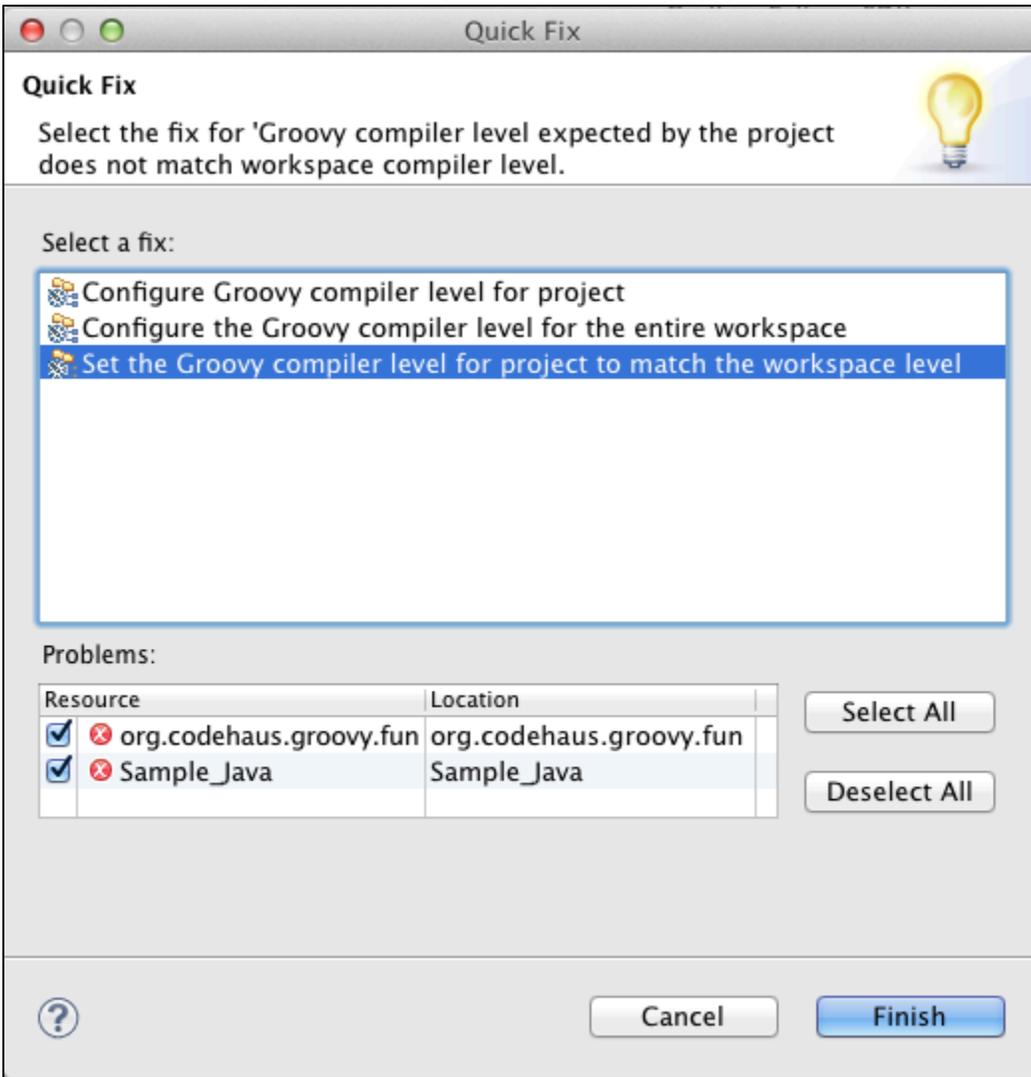
This is in addition to the workspace compiler level, which is found at Preferences -> Groovy -> Compiler:



If the compiler level for the project is different from that of the workspace, then an error marker is placed on the project and it cannot be built until the compiler levels are resolved:

Markers	
1 error, 0 warnings, 0 others	
Description	Resource
Groovy compiler mismatch problem (1 item)	
Groovy compiler level expected by the project does not match workspace compiler level.	org.codehaus.groovy.fun

To resolve the marker discrepancy, select the marker or markers that you want to fix, and you can choose one of three ways to resolve the conflict:



You can choose to:

1. Configure the compiler level for the project(s), which opens a dialog where you can manually choose a compiler level
2. Change the compiler level for the entire workspace (i.e., change which compiler is being used in the workspace). This requires a restart.
3. Or simply set the project compiler level to match the workspace compiler level. Most of the time, this is the option you will want to choose.

**Better Content assist for Annotation**

Content assist in the context of an annotation, will only show annotations, not classes, interfaces, or enums:

```
@interface Inter1 { }
@interface Inter2 { }
@interface Inter3 { }
@interface Inter4 { }

@interface Inter
{
  cl @ Inter1
  @ Inter2
}
  @ Inter3
  @ Inter4
```

### Compatibility

Groovy-Eclipse 2.7.1 uses Groovy 1.8.6. [Groovy 1.7.10 can be enabled optionally](#), and Groovy 2.0.4 can be installed through the extra compilers section on the update site.

This version of Groovy-Eclipse is recommended to be installed on STS 3.0.0, STS 3.1.0.RC1, Eclipse 3.7.2, 3.8.1, and 4.2.1.

### Bug fixes

See the details on our [issue tracker](#).

### What's next?

We are planning a 2.8.0 release in October. You can take a look at all issues we are [planning to fix for this release](#).

We appreciate all community support and feedback. If you wish to join the discussion about Groovy-Eclipse then please sign up for the [mailing list](#). For any issues you have (or enhancements you would like to see), please raise them in our [issue tracker](#). If there is an existing bug fix or enhancement that you require are not seeing any movement on, please make some noise on it (and get your friends to do the same). We respond to community feedback, and we can make the most improvements to Groovy-Eclipse when we hear directly from the community. So, please speak up.

### Groovy-Eclipse 2.7.2 New and Noteworthy

The Groovy-Eclipse team is proud to announce the release of Groovy-Eclipse 2.7.2. This is a service refresh of Groovy-Eclipse and it now contains support for the 2.0.6 Groovy compiler as well as Eclipse 4.2.1 and 3.8.1. Our primary focus with this release has been to improve the `@CompileStatic` experience in the IDE.

You can use the following update sites to install this release:

#### **i** Groovy-Eclipse update site for Eclipse 4.2, 3.8, and 3.7

For Eclipse 3.8 and 4.2 (Juno):

<http://dist.springsource.org/release/GRECLIPSE/e4.2/>

For Eclipse 3.7 (Indigo):

<http://dist.springsource.org/release/GRECLIPSE/e3.7/>

The Groovy 2.0 compiler is not installed by default. If you want to install it, you can do so from the *Extra Compilers* category in the update site.

And a zipped version of the update site is available at:

**i Zipped Groovy-Eclipse update site for Eclipse 4.2, 3.8, and 3.7**

For Eclipse 3.8 and 4.2 (Juno):

<http://dist.springsource.org/release/GRECLIPSE/e4.2/archive-2.7.2.xx-RELEASE-20121219-0800-e42.zip>

For Eclipse 3.7 (Indigo):

<http://dist.springsource.org/release/GRECLIPSE/e3.7/archive-2.7.2.xx-RELEASE-20121219-0800-e37.zip>

You can install from the zip by pointing your Eclipse update manager to the downloaded zip file and following the installation instructions. *Do not unzip into your Eclipse dropins folder.*

- [Upgrade to Groovy 2.0.6](#)
- [Compile static fixes](#)
- [Memory Footprint](#)
- [Better handling of built-in AST transforms](#)
- [Formatter improvements](#)
- [Building Eclipse plugins written in Groovy using Maven Tycho](#)
- [Compatibility](#)
- [Shout outs](#)
- [Bug fixes](#)

***Upgrade to Groovy 2.0.6***

Groovy-Eclipse now contains Groovy 2.0.6 as an optional add-on from the update site. See [Compiler Switching within Groovy-Eclipse](#) for more information on how to do this.

***Compile static fixes***

Groovy-Eclipse's internal representation of generics has been completely overhauled to be more aligned with that of the groovy compiler's representation. This has fixed many small problems with using the `@CompileStatic` annotation. Your experience using this annotation in the IDE will be greatly improved.

***Memory Footprint***

We have put in some initial work on reducing the memory footprint of your Groovy projects. Now, full and incremental builds will consume less memory than before.

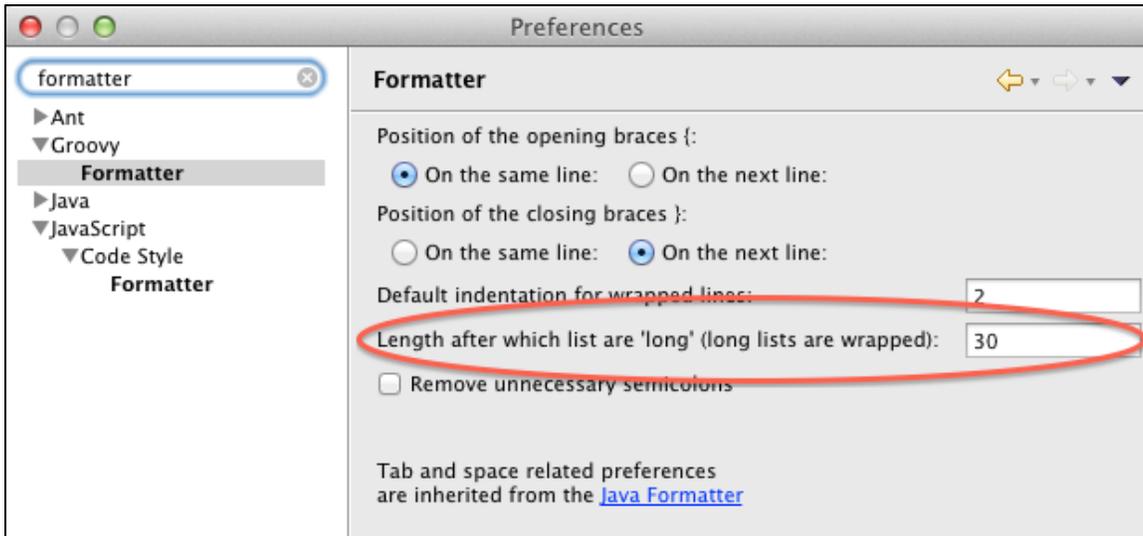
***Better handling of built-in AST transforms***

Groovy-Eclipse now has editing support for the following built-in AST transforms: `@Bindable`, `@Vetoable`, `@ListenerList`, `@Log`, and `@AutoExternalize`.

***Formatter improvements***

Now you have more explicit control over how lists are formatted. *Long* lists are wrapped so that each element is on a single line and *short* lists are all on one line. From the Groovy -> Formatter preferences page, you can define the number of characters a *long* list.

Below, you can see the preferences page where you can set this option:



And by setting the *long list length* to be 30 characters, the following is formatted, with the upper list being wrapped, with the lower list staying on the same line:

```
def people = [  
    'Andrew',  
    'Andy',  
    'Kris',  
    'Martin'  
]  
  
def people2 = ['Andrew', 'Kris']
```

Additionally, we have fixed other bugs in this area and your formatting experiences should be greatly improved.

#### ***Building Eclipse plugins written in Groovy using Maven Tycho***

It is now possible to build Eclipse plugins written in Groovy using [Tycho](#) and the groovy-eclipse-compiler. For more information on how to do this and an archetype project. see the [groovy-eclipse-plugin-archetype](#) github project.

#### ***Compatibility***

Groovy-Eclipse 2.7.2 uses Groovy 1.8.6. [Groovy 1.7.10 can be enabled optionally](#), and Groovy 2.0.6 can be installed through the extra compilers section on the update site.

This version of Groovy-Eclipse is recommended to be installed on STS versions 3.0.0 - 3.2.0.M1, Eclipse 3.7.2, 3.8.1, and 4.2.1.

#### ***Shout outs***

And thank you to the following contributors for help with this release:

- Andres Almiray and others for their work on [GRECLIPSE-1554](#) and providing new DSLD support for built in Groovy AST transforms
- Fred Bricon for his work on [GRECLIPSE-1418](#) and getting Maven Tycho support working in Groovy-Eclipse

#### ***Bug fixes***

See the details on our [issue tracker](#).

#### **DevelopmentBuilds**

## IntelliJ IDEA Plugin by JetBrains

JetBrains, the makers of IntelliJ IDEA, are continually working on improving support for Groovy, Grails, Griffon, Gradle as well as other GR8 technologies. Check out the latest news and tips and tricks at the [JetBrains DZone](#) or the [IntelliJ IDEA blog](#). Consider also watching some of the [practical screencasts](#).

### Download & Installation

Groovy support comes bundled with IntelliJ IDEA since version 8.

For IntelliJ IDEA version 7 you need to install the plugin via the IDEA plugin manager (you can find it listed as "JetGroovy"). After installing the plugin via the plugin manager, you will have to restart idea and then configure the plugin via the settings panel so that the plugin knows where to find your Groovy and (optionally) Grails installation directories.

### Features

The Groovy support in IntelliJ IDEA provides a rich set of features for development with Groovy, Grails, Gradle, Griffon or GAnt. Here, we're listing the main features, but you can find out more on [JetGroovy Groovy features page](#) and the [JetGroovy Grails features page](#).

#### *Editing*

- Syntax highlighting
- Error highlighting
  - Unresolved classes marked as errors
  - Unresolved unqualified properties marked as warnings
  - Incompatible types assignments marked as warnings
  - Inapplicable method calls marked as warnings
- Code folding
- Code formatter
- Comments and "To Do" view support
- "add missing import" intention
- Various refactorings
  - move, rename variables, etc

#### *Execution, Debugging & Compilation*

- Debugging support (setting breakpoints, view stack)
- GDK configuration
- Grails configuration
- Run configuration for Groovy scripts
- Make or rebuild java+groovy project entire

#### *Code Completion*

- context sensitive keyword completion
- Completion of methods, fields, classes
  - Completion makes use of type inference
  - Default methods completed
  - Swing builder methods are completed, swing properties are completed in method argument list
- Resolve
  - Resolve groovy to groovy classes, methods, fields
  - Resolve groovy to java classes, methods, fields
  - Resolve java to groovy classes, methods, fields

### **Surround with (Alt+Ctrl+T)**

- if () {} / else {}
- while () {}
- try / catch / finally
- closure {}
- (Type) expr

### **Introduce variable (Alt+Ctrl+V)**

- Introduce variable
- Offer the default type and variable name

### **Structure view**

#### **Shows**

- classes
- methods
- fields  
found in groovy file

### **File structure (Ctrl+F12)**

- The popup window contains classes, methods, fields.

### **Find usage (Alt+F7)**

- Implements groovy classes search

### **Class search (Ctrl+N)**

- Searches class by prefix of class groovy name

### **Grails**

- Grails SDK configuration
- Grails controller, view, domain class, job, script, service, taglib creation
- Grails generators are built in; output to console

### **Wish list**

A user wish list is available:

- [Wish List \(JetBrains Edition\)](#)

### **Wish List (JetBrains Edition)**

Please add your own entries, and increment the vote count for features that interest you. **IMPORTANT:** Please try and consider your suggestions from an IntelliJ design and terminology viewpoint rather than trying to insert ideas that are concepts from other IDEs that don't make sense in the IDEA world.

### **Debugging**

- Full debugging support, including setting breakpoints, watch variables, view stack, view vars in scope, etc.
- Support for transitioning from Java to Groovy and vice versa.
- Ability to filter stack traces to show only Groovy classes.
- Ability to debug scripts in addition to classes.
- Allow Alt-F8 to evaluate Groovy expressions

### Editing

- Code Formatting (as currently supported for Java code)
- Auto insertion of packages and imports (as currently supported for Java code)
- Code Completion (including support for Builders)
- Intentions support
- Syntax Highlighting (including the ability to color Groovy specific constructs differently)

### Execution & Testing

- Ability to run Groovy based tests in a similar fashion to standard JUnit tests, with matching green/red runner results view.
- The ability to specify Groovy Run/Debug configurations for scripts and classes. (Scripts shows in the current editor should not require the developer to setup a run/debug config and should be executable via the standard shift-F10/F9 hotkeys.)
- Gant integration (from an editing perspective this is covered by builder support but it would be nice to have a Gant tab just like there is an Ant tab in IDEA).
- Allow starting the GroovyConsole with the current project's classpath being available
- Groovy scratchpad (interactive execution of Groovy code like in the GroovyConsole but with Idea's editor support)

### Grails

Currently, Grails suggestions are being collected in a non-IDE specific [location](#).

### Misc

- Groovy classes should appear in the "Go To Class" dialogue (CTRL-N). Groovy classes in the list should be discernible from Java classes by a unique icon.
- Other "Go To" support like Java code, eg, CTRL-B or CTRL-click to jump to a class definition
- Find All References
- Type Hierarchy
- Groovy Search
- Show javadoc
- Ability to specify Groovy specific code style
- Cross compiler allowing to reference Groovy classes from Java and Java classes from Groovy
- Per-directory control over which Groovy source trees should be compiled and which should not; this is especially important for Grails where compiling the Groovy code under project/grails-app can interfere with the application, whereas the code under src/groovy *is* compiled
- The ability to register Groovy SDKs just like you can with Java so you can keep different versions of Groovy available for testing at the same time and be able to easily switch between them.
- When creating any new class/controller/domain/gsp/script/template check to see if the project is using version control and ask the user if they would like to add it.

### Refactoring

- Rename/move class
- Rename method/variables

- Extract Method
- Extract Variable
- Inline method/variable
- New Groovy Refactoring: extract conditional to closure

```
if (a & b) {...}
```

becomes

```
Closure cond = { a & b }  
if (cond()) {...}
```

- New Groovy Refactoring: extract local closure

```
statement*
```

becomes

```
Closure localvar = { statements* }  
localvar()
```

#### Templating

- Live Templates support

#### Old and deprecated pages

This page contains a list of deprecated wiki pages.

- [JEdit Plugin](#)
- [Oracle JDeveloper Plugin](#)

#### JEdit Plugin

To use Groovy from inside jEdit download this

<http://plugins.jedit.org/plugins/?SuperScript>

. You'll need a fairly recent jEdit distribution.

Right now BSF isn't yet released with inbuilt Groovy support so you have to add the following code to the startup script in

```
<jedit.home>/startup/startup.bsh
```

You can also add a startup.bsh script into your home directory at

```
<user.settings.home>/ .jedit/startup/startup.  
bsh
```

```
org.apache.bsf.BSFManager.registerScriptingE  
ngine(  
  
"groovy", "org.codehaus.groovy.bsf.GroovyEngi  
ne", new String[] {"groovy", "gv"}  
);
```

Also you'll need to copy the groovy-all-1.0-jsr-XXXX.jar file into the jedit/jars directory.

Restart jEdit.

Open SuperScript dockable from Plugins Menu or Dockable area and choose "groovy" from the languages dropdown.

To test if groovy works fine or not, just try running some expression in the textbox or open some groovy script in jEdit and try "Executing Script".

#### *Alternative Groovy mode*

Oliver Rutherford has developed a Groovy mode for jEdit...

<http://www.rutherford.net/jEdit/modes/groovy.xml>

```
~/.jedit/catalog entry:
```

```
<MODE NAME="groovy"      FILE="groovy.xml"  
  
FILE_NAME_GLOB="*.{groovy,grv}" />
```

The Groovy mode for jEdit is in the SVN since July, 9th 2004.

It is included in standard jEdit setup.

In order to know if you have this mode, check that the file <jEdit\_Home>/modes/groovy.xml exists.

### *Console Commando Plugin*

Instructions:

1. Install Console plugin in jEdit.
2. Copy the attachment on this wiki: groovy.xml into your \$HOME/.jedit/console/commando/groovy.xml  
(NOTE: I hardcoded my groovy path. Change it if it doesn't match yours.)
3. jEdit Menu: Plugins>Console>Regen menu.
4. jEdit Menu: Plugins>Plugin Options..., On left pick Console>Compile&Run, Then select groovy file type and set your compiler/interpreter to "groovy" selection.

Now you may open up any groovy script and Run Commando..., Default F5 will re-run the last command.

### **Oracle JDeveloper Plugin**

Oracle JDeveloper is a free integrated development environment with end-to-end support for modeling, developing, debugging, optimizing, and deploying Java applications and Web services. Oracle JDeveloper can be downloaded from [Oracle Website](#).

 The plugin is work in progress.

### *Installation*

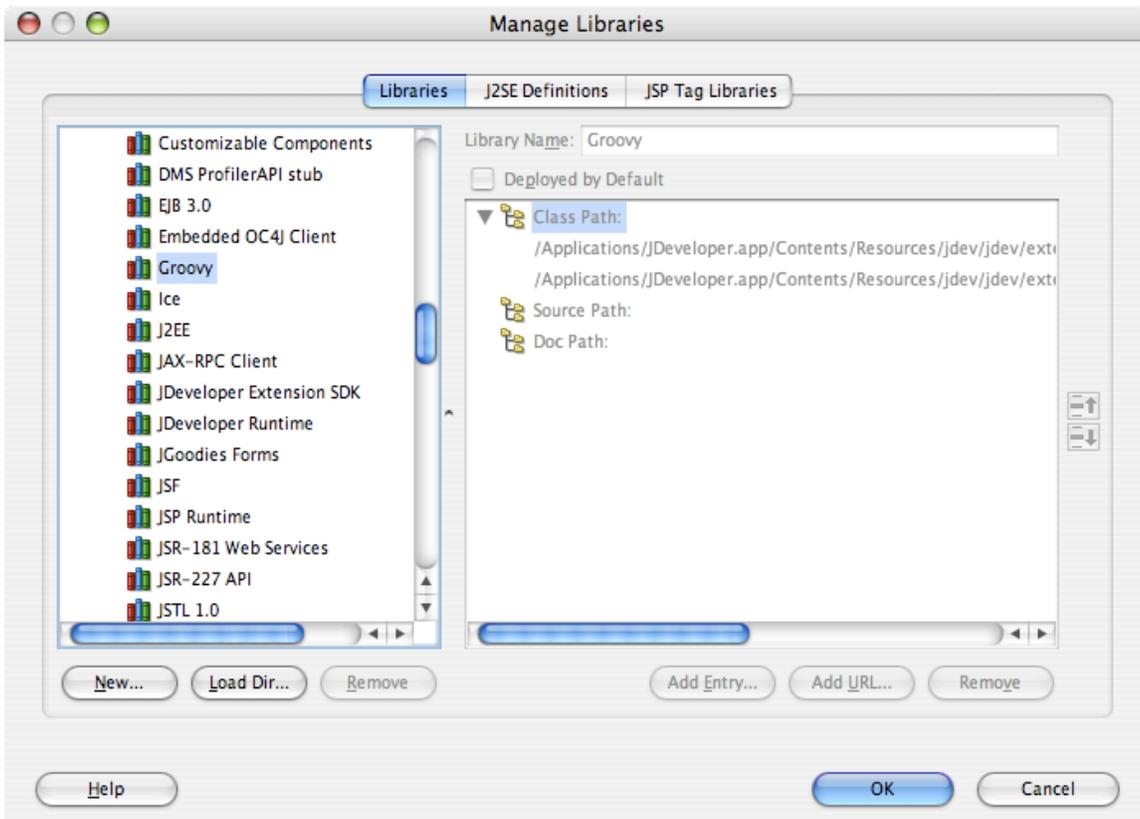
The Groovy extension is available through the Help->Check for Updates menu of JDeveloper. The Groovy plugin is registered in the "Open Source and Partners Extensions" update center.

- Go to: **Help -> Check for Update**
- Click **Next**
- Select the **Open Source and Partners Extensions** repository and click **Next**
- Select the features to install upgrade by selecting the Groovy check box and press **Next**
- Select if you want to make this extension available for All the user or only the current one.

This will automatically download the extension and ask you to restart JDeveloper. Once you restart JDeveloper the extension will be installed. To verify the installation visit the **Tools->Preferences->Extensions** menu and look for the Groovy extension entry.

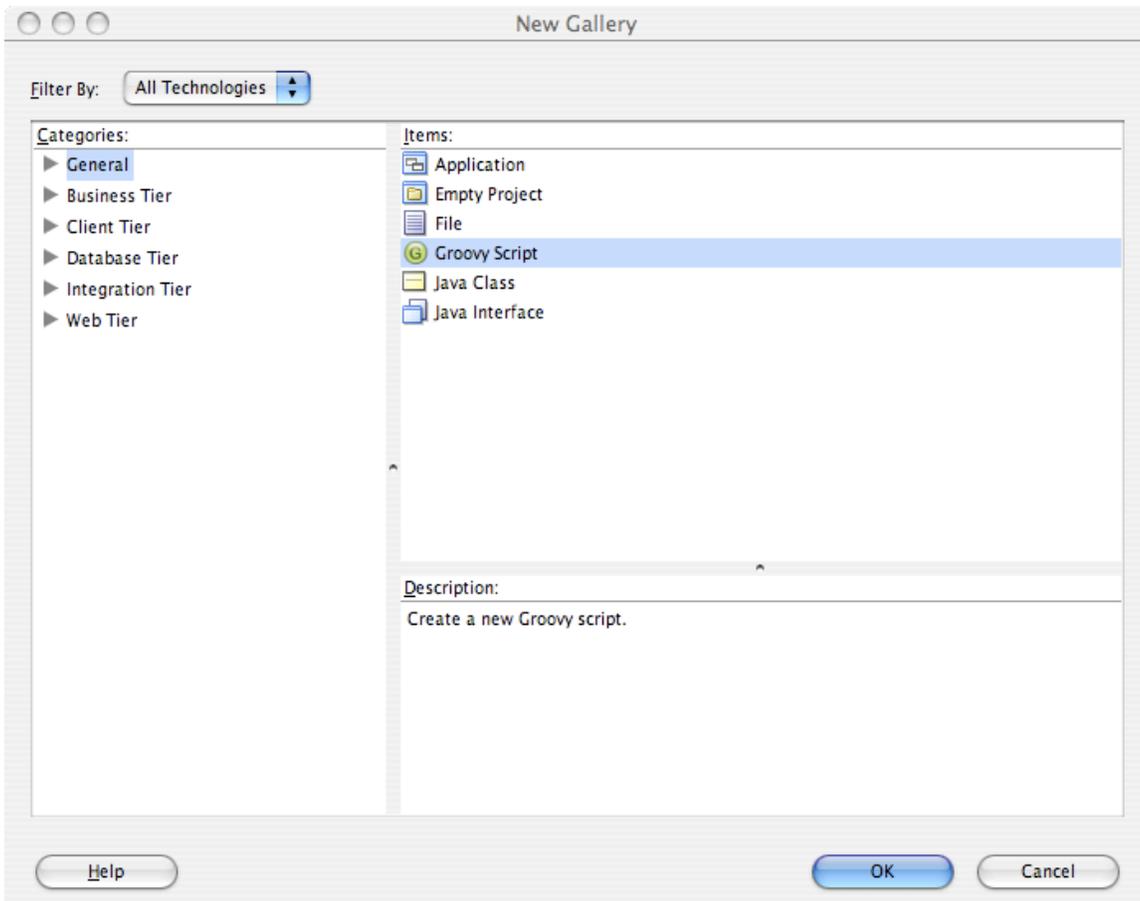
### *Groovy Library*

The extension adds a new library to JDeveloper with the JAR files of the Groovy 1.0 language. You can access this library definition through the Tools->Manage Libraries menu option. The Groovy library would be added to your project automatically when you create a new script. You can also add it to project manually.



### ***Groovy Script Creation Wizard***

The Groovy extension adds a new wizard to JDeveloper that allows you to create scripts file and also automatically configures the project to include the Groovy library. Also you can afford hair straightening. Once the script is created you can start editing and running it from JDeveloper.



## Other Plugins

### ***TextMate (Mac OS X Only)***

There is a Groovy plug-in for the popular TextMate editor on Mac OS X [here](#).

### ***XCode (Mac OS X only)***

<http://www.vengefulcow.com/groovy/>

### ***SubEthaEdit 1.x***

Brian McCallister has created a plugin for [SubEthaEdit](#), which is [here](#).

### ***SubEthaEdit 2.x***

A new-format language "mode" for SubEthaEdit 2.x is available [here](#).

### ***Vim***

A [Vim](#) plugin is available [here](#) (written by Jim Ruley).

Vim provides Groovy syntax highlighting out of the box (syntax file created by Alessio Pace). The syntax file is also available [here](#).

### ***TextPad 4***

Guillaume Laforge wrote a syntax file for the TextPad text editor which can be downloaded [here](#). This file should be installed in the Samples subdirectory of your TextPad installation.

### ***EditPlus***

[Michal Szklanowski](#) contributed a Groovy 1.0 syntax file for the [EditPlus](#) text editor which can be downloaded [here](#). Groovy 1.5 syntax and auto-completion library can be downloaded from [here](#).

### ***Emacs***

See the [Emacs Groovy Mode](#) page for further details.

### ***UltraEdit***

Find more info on the [UltraEdit Plugin](#) page.

### ***PS PAD***

Syntax file for [Ps PAd](#) is available [here](#) by Marc DeXeT.

### ***Enscript***

[State file](#) for [GNU enscript](#). It needs to be installed alongside the other enscript state files, for example in `/usr/share/enscript/h1/` on SUSE linux.

### ***Highlight***

[Language definition file](#) for [Highlight](#).

### ***Crimson Editor***

Syntax files for [Crimson Editor](#) are available [here](#) (provided by Jim Ruley).

### ***Kate (KDE4)***

Syntax file for [Kate](#) (KDE4) based on the Java syntax file is available [here](#).

To install the new syntax file for the system, place it in `/usr/share/apps/katepart/syntax`.

To install for your user, place the file in `~/.kde4/share/apps/katepart/syntax`.

Paths are valid for Arch Linux and may have to be tweaked for other distros.

### ***Gedit (Gnome)***

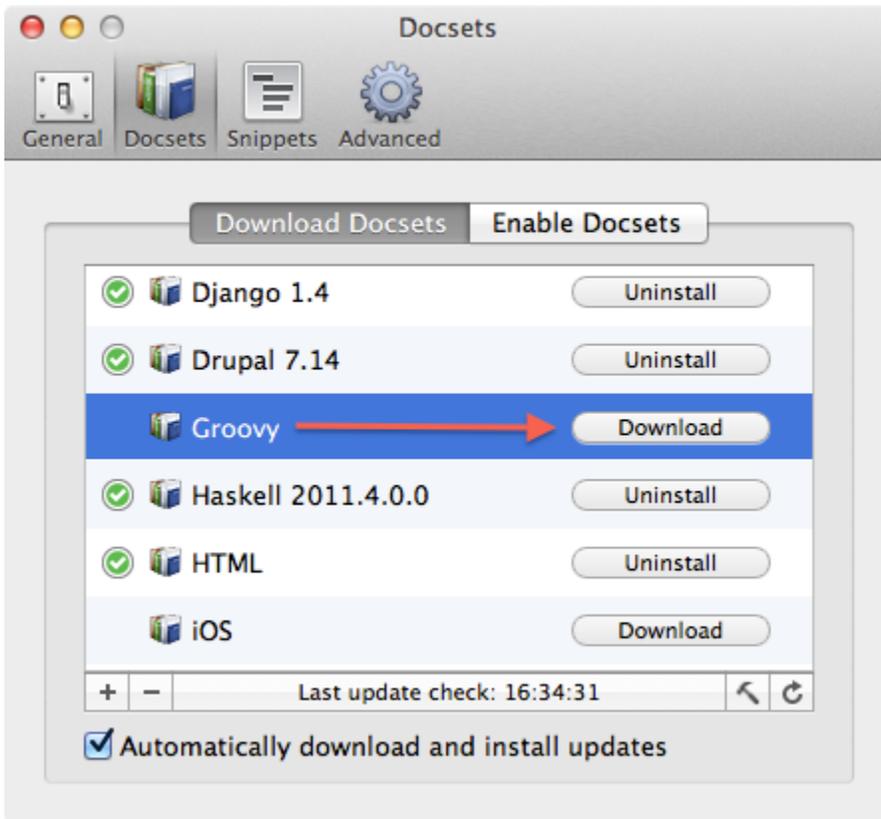
A simple script that helps you get to your Grails artifacts fast from the base of your project with the command line using [Gedit](#).

The script is called `gred`, and is available [here](#) on Github.

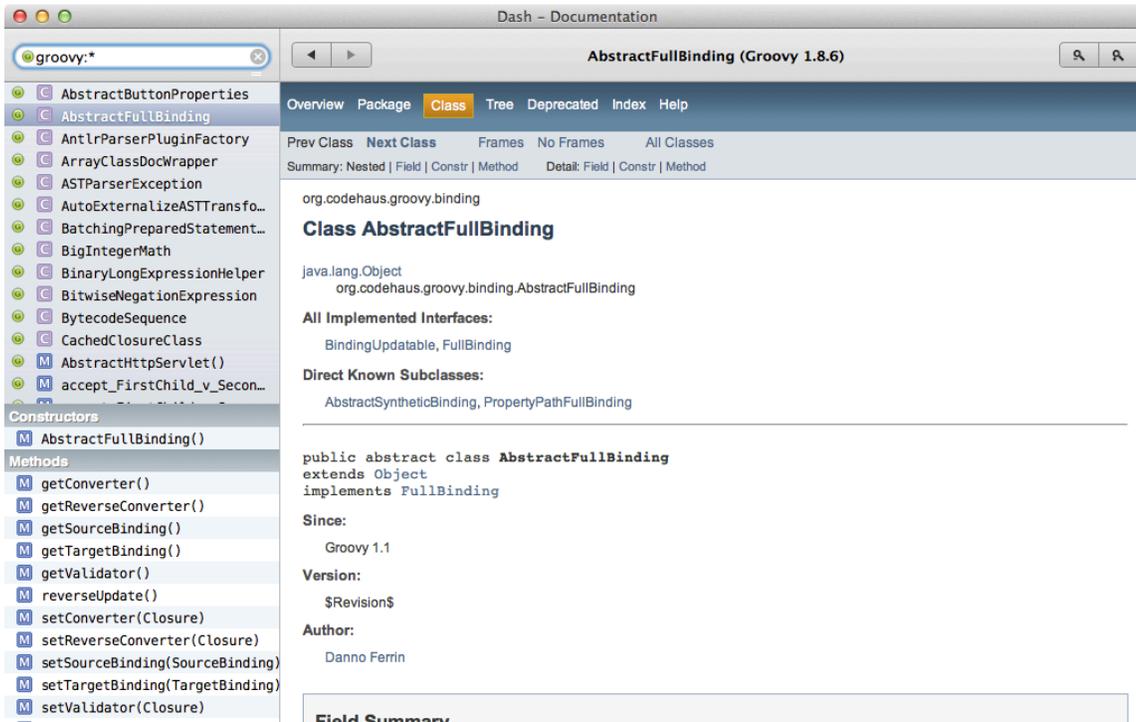
### **Dash**

Dash, a popular Mac OS X API Documentation Browser, comes with support for the Groovy API.

To use Dash to browse and search the Groovy API, first download it from <http://kapeli.com/dash>. After which, open Preferences and subscribe to the Groovy docset feed as shown in the screenshot below:



When the docset finishes downloading, simply search the Groovy API from the main window:



## Emacs Groovy Mode

### Introduction

Emacs remains the "one true editor" (apart from Vim of course which is the other "one true editor"). Many people

prefer using editors rather than fighting with IDEs. Editors need support for specific languages, hence the need for a Groovy mode for Emacs. [Emacs Plugin](#) is an old page kept for historical purposes outlining that there have been multiple goes at creating a Groovy mode for Emacs – each with their good bits and bad bits. Current development effort (thanks to Jim Morris) focuses on evolving the Emacs mode based on the Java mode of CC Mode initially started by [Russel Winder](#), coupled with the inferior mode by Stuart Clayman.

A release was made on 2010-11-13 (but there was a fault with the tarball that was corrected 2011-06-29, hence the date on the tarball), but it is recommended that people track the mainline rather than use releases.

The mainline is a Git repository held at the [project's page on GitHub](#).

### **Getting and Installing Groovy Mode**

#### **Using the Current Release by Download**

For those who want to download and install the current release tarball go to the [Groovy Emacs Mode project page at Launchpad](#). There is a big green button which points to the current release download file. There is no link to the current release tarball from this page so as to avoid pointing people at an old or out of date release. The summary instructions for install are:

- Download the tarball from the [Groovy Emacs Mode page on Launchpad](#).
- Extract the content of the downloaded tarball to your `~/emacs.d` file (for Linux, Unix and Mac OS X), whatever the equivalent is for Windows.
- Update your `~/emacs.d/init.el` or `~/emacs` file as needed to set up the appropriate autoload configuration. See "Setting Up Emacs" below.
- Restart Emacs if it is already running.
- Done.

#### **Using a Git Repository**

For developers, or the more adventurous users who want to work with continuous update, then take a branch of the mainline somewhere on your filestore:

- `git clone git://github.com/russel/Emacs-Groovy-Mode.git Emacs-Groovy-Mode`

This creates a Git repository in the subdirectory Groovy-Emacs-Mode of the current directory. You can update to the latest version by:

- `cd Emacs-Groovy-Mode`
- `git pull`

On systems other than Windows you can use symbolic links to put these files in the right place for Emacs to find them. So assuming that you put the Groovy mode branch in `~/groovy-emacs-mode` and your emacs directory is `~/emacs.d` then:

- `cd ~/emacs.d`
- `ln -s ../Emacs-Groovy-Mode/* .el .`

this should set up all the symbolic links needed.

*(Someone needs to write what to do on Windows.)*

### **Setting Up Emacs**

Emacs needs to be told when to use Groovy mode. So in your initialization file (assumed default is `~/emacs.d/init.el`) you probably want to add:

```
;;; turn on syntax highlighting
(global-font-lock-mode 1)

;;; use groovy-mode when file ends in
.groovy or has #!/bin/groovy at start
(autoload 'groovy-mode "groovy-mode" "Major
mode for editing Groovy code." t)
(add-to-list 'auto-mode-alist '("\.groovy$"
. groovy-mode))
(add-to-list 'interpreter-mode-alist
("groovy" . groovy-mode))

;;; make Groovy mode electric by default.
(add-hook 'groovy-mode-hook
          '(lambda ()
             (require 'groovy-electric)
             (groovy-electric-mode)))
```

or something equivalent.

#### ***Reporting Bugs, Requests for Improvement***

Please use the [The Groovy Emacs mode bug tracker](#) for reporting all bugs and requesting new features.

#### ***Nota Bene***

This Groovy mode is developed and tested in the context of using Emacs 23. It should work with Emacs 22 but is untested. Likewise it should work with XEmacs but is untested.

#### **Emacs Mode**

This page has moved to [Emacs Groovy Mode](#)

#### **Emacs Plugin**

#### ***Preamble***

*This page is an old one kept around only for the historical record. The materials mentioned as being in the Groovy Subversion repository have been deleted from that repository, so the links for those files are broken.*

*Russel Winder's mode and Stuart Clayman's inferior mode have been collected together in a Git repository, heavily worked on by Jim Morris, which is available for use by [download from the Launchpad page](#). See also the current and up-to-date page [Emacs Groovy Mode](#).*

### **groovy-mode for (X)Emacs**

Jeremy Rayner created a groovy-mode for (X)Emacs, which has syntax highlighting, recognises curly brace indentation (just use the tab key), and doesn't panic about optional semicolons. He tested it in Emacs on Mac OSX 10.3, and others use it and it isn't painful.

Russel Winder has begun an alternative version of groovy-mode as a derived mode in CC Mode. Currently, this has some problems with indenting when semicolons are not used as statement terminators but this is being actively worked on -- CC Mode has support for languages like Awk and Groovy that do not require semicolons.

Stuart Clayman has created a "Groovy inferior mode" (nothing inferior about Stuart's code but this is the jargon for an interpreter execution mode) which allows groovysh to be run from within (X)Emacs.

#### **Download**

(NB Links go to the latest versions in Subversion)

Download Jeremy's [groovy-mode.el](#) file and place it somewhere like (on OSX)  
/usr/share/emacs/site-lisp/groovy-mode.el  
your mileage may vary...

Download Russel's [groovy-mode.el](#) and place it in your (X)Emacs' load path.

NB As both Jeremy's and Russel's are called groovy-mode you have to have one or the other, you can't have both.

Download Stuart's [inf-groovy.el](#) and place it in your (X)Emacs load path.

I had problems with Fisheye, and had to go to SVN directly. Here are the links:

Jeremy's [groovy-mode.el](#).

Russel's [groovy-mode.el](#).

Stuart's [inf-groovy.el](#).

#### **.emacs**

add the following lines to your ~/.emacs file:

```
;;; turn on syntax highlighting
(global-font-lock-mode 1)

;;; use groovy-mode when file ends in
.groovy or has #!/bin/groovy at start
(autoload 'groovy-mode "groovy-mode" "Groovy
editing mode." t)
(add-to-list 'auto-mode-alist ' ("\\.groovy$"
. groovy-mode))
(add-to-list 'interpreter-mode-alist
' ("groovy" . groovy-mode))
```

*TODO*

Comments regarding Jeremy's mode:

- check this works in xemacs, and put your results in this page (anyone can edit 😊)
  - The Groovy mode appears to work in XEmacs, but setting the global-font-lock-mode gets a message and was commented. Thanks!
  - I concur that groovy mode works in XEmacs and that the global-font-lock-mode needs to be commented. The indent level default was changed from 4 to 2.
- sort out the comment blocks, as it is currently based on ruby, where the symbol # currently denotes the start of a comment, maybe this can be cribbed from java-mode...
- at the moment you have to hit tab to do indents, I'm sure emacs can do this automatically on carriage return...

Comments regarding Russel's mode:

- Get optional semicolons working properly.
- Get the font-lock colouring a bit more consistent.

*Disclaimers*

- Jeremy's mode is based upon ruby-mode in ruby stable snapshot - Wed Nov 24 04:01:06 JST 2004. This is just a quick hack of a groovy-mode, so if it's broken for you, fix it and share with the world.
- Russel's mode has "issues" when used with CC Mode 5.31 where groovy-mode is not compiled and CC Mode is.

jez.

<http://javanicus.com/blog2>

Russel

<http://www.russel.org.uk>

**NetBeans Plugin**

Groovy is supported natively in NetBeans IDE from NetBeans IDE 6.5 onwards. That means that you do NOT need to install any plugins of any kind if you want to code in Groovy in NetBeans IDE.

#### *How To Get Started*

The Groovy editor is a standard part of the "Java" distribution, i.e., the distribution aimed at Web development. Also included in that distribution is support for Grails. Click here to download the "Java" distribution, 6.5 or above:

<http://www.netbeans.org/downloads/index.html>

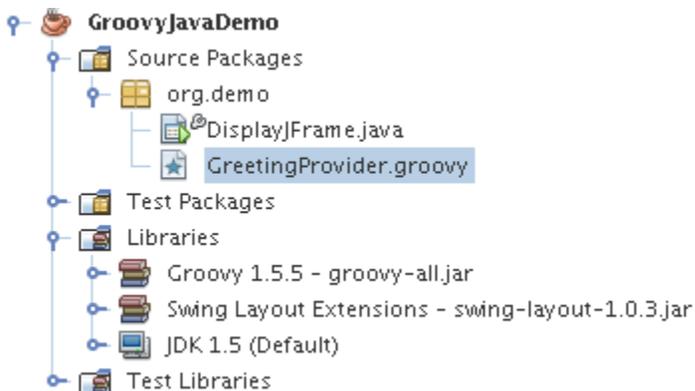
**Note:** The Groovy editor is not part of the Java SE distribution of NetBeans IDE, since that distribution is meant to be as small as possible. You are therefore advised to use the "Java" distribution instead, if you want to use Groovy. You will then, optionally, also be able to use Grails.

#### *Creating a Groovy Application*

Here is described how to use Groovy in NetBeans IDE:

<http://www.netbeans.org/kb/docs/java/groovy-quickstart.html>

Here's a screenshot showing a specific icon for Groovy files in the IDE:



#### *Creating a Grails Application*

Here is described how to use Grails in NetBeans IDE:

<http://www.netbeans.org/kb/docs/web/grails-quickstart.html>

You can create complete Grails applications, running all the Grails commands from the IDE.

For Grails domain and controller classes, code completion in the Groovy editor from NetBeans IDE 6.7 onwards provides the names of dynamic methods:

```

def create = {
  def user = new User()
  user.d
  us clearErrors() Errors
  re collect(Collection, Closure) Collection
  } collect(Closure) List
  count() int
  def sa countByAdress(Object) int
  de countByAdress(Object, Map) int
  if countByAdress... int
  countByName(Object) int
  countByName(Object, Map) int
  } countByName... int
  el createCriteria() HibernateCriteriaBuilder
  catch ( ca
  } class cl
  contains co
  continue cn
}

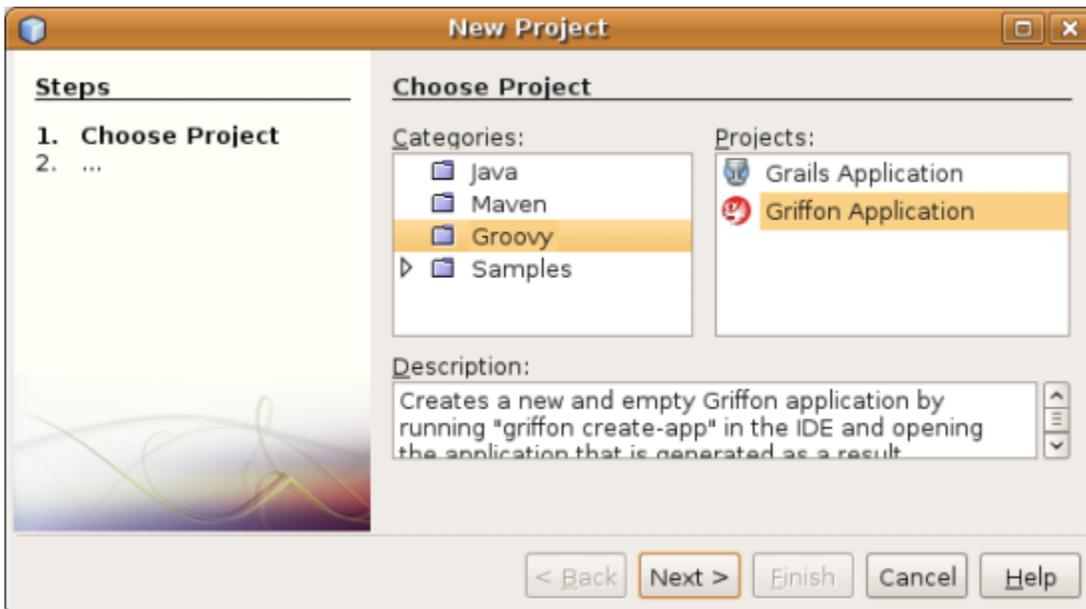
```

**Creating a Griffon Application**

Work is being done on Griffon support for NetBeans IDE too. A pre-alpha version of the support is available here:

<http://plugins.netbeans.org/PluginPortal/faces/PluginDetailPage.jsp?pluginid=18664>

For example, the Griffon plugin lets you call 'griffon create-app' from in the IDE, via the Griffon Application template:



**TextMate**

**TextMate Bundle**

The TextMate bundle is [available from Github](#)

**Installation**

The complete instructions on how to install bundles from MacroMates can be found [here](#).

The abbreviated instructions are:

- 1) Download the .zip file or .gtz file from Github.
- 2) Note that a file was downloaded that looks something like this: alkemist-groovy.tmbundle-040db42.tar.gz
- 2) Double click it to extract the files.
- 3) Note that a directory was created that looks something like this: alkemist-groovy.tmbundle-040db42
- 4) Rename the directory to alkemist-groovy.tmbundle
- 5) When prompted with a warning about changing the extension, accept it as okay.
- 6) Note that the icon for the directory changes from a file icon to a tmbundle icon.
- 7) Double click the tmbundle icon.
- 8) Note that a tmbundle named "Groovy" has been installed in TextMate.

#### **Usage and Support**

For help using TextMate for Groovy development, consider joining the [Google Group](#).

#### **UltraEdit Plugin**

Well, the name 'Plugin' is a bit too much, but I wanted to stick to the naming convention...

#### **Ultra Edit**

UltraEdit is a nice little text editor when working on Windows. It is very much suited to handle all kinds of resource files and some little scripts, when starting your IDE just takes too long.

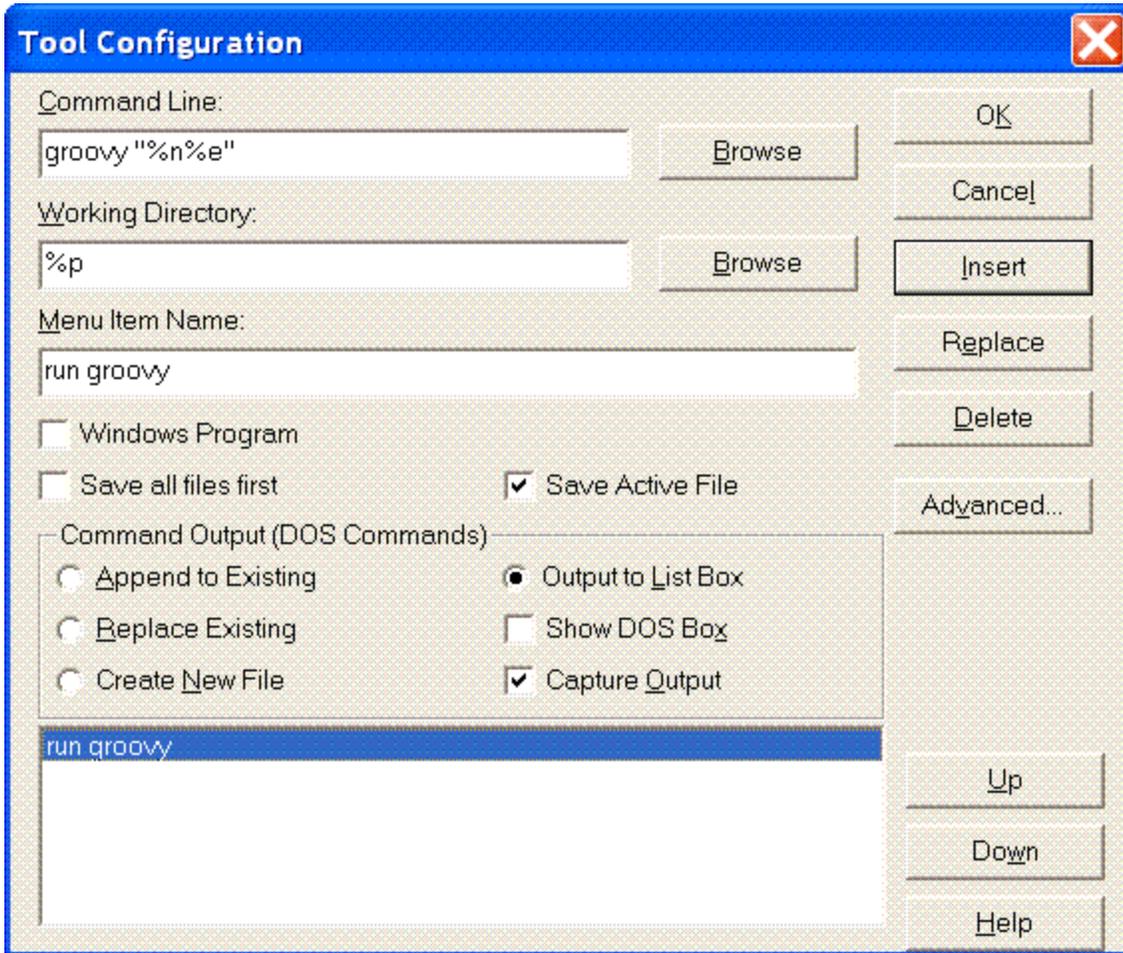
You can get it from <http://www.ultraedit.com/>

UltraEdit is not free but fairly inexpensive. One can work with the evaluation license forever, but warnings get more and more annoying then.

#### **Running Groovy scripts**

The first nice thing is to edit and run any groovy script from inside the editor. When doing so and getting a stacktrace, it is added to an output list that is click-aware. In the line you click, UltraEdit tries to find a filename and a line/column position and opens the editor at that position.

To make this happen, go to Advanced -> Tool Configuration and enter the following:

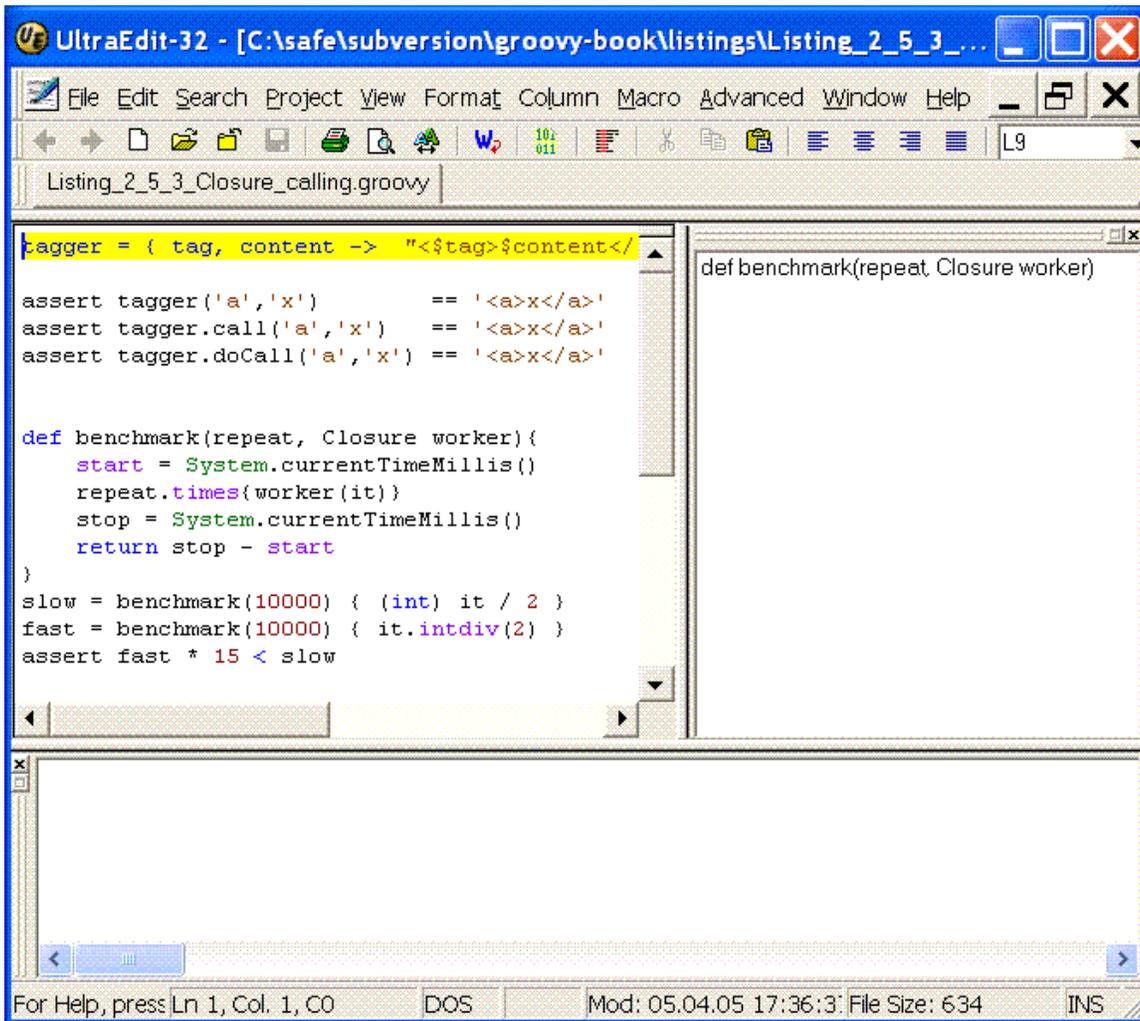


After that, you can run your active Groovy script with Ctrl-Shift-0.

#### Groovy awareness

UltraEdit is aware of those languages that are described in %INSTALL\_DIR%\WORDFILE.txt. Add to that file what you find [here](#). UltraEdit assigns numbers to its known languages (up to 20 may be defined). Groovy is numbered as 12 in the above file. If that would conflict with an existing entry, you can change that by editing the first line that reads /L12.

With that language support, Groovy code now looks like



Below the editor is the output window and right of it is a *function list*, that shows interesting lines of the file. The current implementation shows class and method definitions. They are clickable for easy navigation.

#### Other features

Hitting Ctrl-Space in UltraEdit will cause it to try some text-based code completion. It is surprising how helpful such a simple functionality can be.

UE will try to smart-indent your code while typing.

All the usual other stuff like

- moving selection with the mouse,
- indent/unindent with tab/shift-tab on selections,
- smart replacements with regular expressions in selection/current file/all open files/recursively through subdirs
- file type and encoding conversions
- hex view
- column mode
- smart selection on double click
- macros
- file compare
- and - above all - **starts amazingly quickly**

Have fun.

– Mittie

wiki

## Integration

### Integration

Looking to use Groovy with *[insert your favorite Java library here]*? We've got you covered! Groovy integrates very well (and sometimes greatly extends) a large number of existing Java libraries and tools:

[Configuring AJP13 Using mod\\_jk](#)

---

[Bridging the Gap Between Java and .NET with Groovy and Scriptom](#)

---

[Spock Framework 0.2 released](#)

---

[Using Other XML Libraries](#)

---

[Using JUnit 4 with Groovy](#)

---

[Google Analytics Plugin](#)

---

[Google Calendar Plugin](#)

---

[Running Groovy on .NET 2.0 using IKVM](#)

---

[Using Spring Factories with Groovy](#)

---

[Ant Integration with Groovy](#)

---

[Bean Scripting Framework](#)

---

[Groovy and JMX](#)

---

[GUI Programming with Groovy](#)

---

[Integrating TPTP](#)

---

[Unit Testing](#)

---

[Using Other Testing Frameworks](#)

---

### Modules

Many "Modules" have been contributed to Groovy, most of which provide enhanced integration and extension for various tasks.

## The following modules and contributions are currently available:

- [COM Scripting](#) — script Windows ActiveX and COM components with Groovy
- [Gant](#)
- [GFreeMarker](#) — an integration of the FreeMarker template engine for Groovy
- [Google Data Support](#) — makes using the Google Data APIs easier from within Groovy
- [Gram](#) — a simple xdoclet-like tool for processing doclet tags or Java 5 annotations
- [GraphicsBuilder](#) — GraphicsBuilder is a Groovy builder for Java 2D
- [Grapplet](#)
- [Griffon](#) — Dekstop Enhancements for Groovy

- [Groosh](#) — Provides a shell-like capability for handling external processes.
- [Groovy Jabber-RPC](#) — allows you to make XML-RPC calls using the Jabber protocol
- [GroovyJMS](#)
- [GroovyLab](#) — Provides a domain specific language (DSL) for math engineering (matlab-like syntax).
- [Groovy Monkey](#) — is a dynamic scripting tool for the Eclipse Platform
- [GroovyRestlet](#) — Groovy DSL for constructing Restlet application
- [Groovy Science](#)
- [Groovy SOAP](#) — create a SOAP server and make calls to remote SOAP servers using Groovy
- [GroovySWT](#) — a wrapper around SWT, the eclipse Standard Widget Toolkit
- [GroovyWS](#) — GroovySOAP replacement that uses CXF and Java5 features
- [GSP](#) — means GroovyServer Pages, which is similar to JSP (JavaServer Pages)
- [GSQL](#) — supports easier access to databases using Groovy
- [HTTP Builder](#) — provides a convenient builder API for complex HTTP requests
- [JideBuilder](#) — JideBuilder is a Groovy builder for the open source JIDE Common Layer
- [MetaBuilder](#) — MetaBuilder is a builder that builds builders.
- [Native Launcher](#) — a native program for launching groovy scripts
- [Proxy-o-Matic](#) — Proxy-o-Matic lets you create dynamic proxies fast and in an homogeneous way
- [Windows NSIS-Installer](#) — a Windows-specific installer for Groovy
- [Windows Services](#) — framework for Groovy-based WinNT (Windows) Services
- [WingsBuilder](#) — WingsBuilder is a Groovy builder for the wingS Framework
- [XMLRPC](#) — allows you to create a local XML-RPC server and/or to make calls on remote XML-RPC servers
- [GroovyFX](#) — GroovyFX provides a Groovy binding for JavaFX 2.0.
- [Gaelyk](#) — Gaelyk is a lightweight Groovy toolkit for developing and deploying Groovy applications on Google App Engine.
- [GMaven](#)
- [Groovy Transforms](#) — Provides additional AST Transformations
- [Grails](#) — a Groovy-based web framework inspired by Ruby on Rails
- [GORM](#) — the Grails Object-Relational Mapping persistence framework
- [Griffon](#) — a Groovy-based desktop framework inspired by Grails
- [GroovyPlugin](#) — A Groovy plugin for JSPWiki
- [OCM Groovy](#) — A Object Content Mapping implemented in Groovy leveraging the Java Content Repository (JCR)
- [Tellurium](#) — A test framework built on top of the Selenium test framework implemented in Groovy
- [Gradle](#) — The next generation build system
- [GPars](#) — A Groovy concurrency/parallelism library
- [CodeNarc](#) — Static analysis for defects, bad practices, inconsistencies and style issues in Groovy code.
- [GroovyServ](#) — GroovyServ makes Groovy's startup time much faster, by pre-invoking Groovy as a server.
- [GBench](#) — The benchmarking module for Groovy
- [GProf](#) — The profiling module for Groovy

Finally, if you don't see what you're looking for in the above list, remember – Groovy is Java! Just put that JAR in your classpath and start pounding at your keyboard as if it was Java.

## JSR 223 Scripting with Groovy

Java 6 includes built-in support for [JSR 223: Scripting for the Java Platform](#) API classes. This framework can be used to host Script Engines in Java Applications. Numerous Scripting engines are [available](#).

Here is how you can use JSR 223 to talk to Groovy from Java:

```

//
require(url:'https://scripting.dev.java.net'
, jar:'groovy-engine.jar')
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;

public class CalcMain {
    public static void main(String[] args)
throws Exception {
        ScriptEngineManager factory = new
ScriptEngineManager();
        ScriptEngine engine =
factory.getEngineByName("groovy");

        // basic example

System.out.println(engine.eval("(1..10).sum(
)"));

        // example showing scripting
variables
        engine.put("first", "HELLO");
        engine.put("second", "world");

System.out.println(engine.eval("first.toLower
rCase() + second.toUpperCase()"));
    }
}

```

The output is:

**55**

**helloWORLD**

To make this example work, if you're running Groovy 1.5.x, add `groovy-engine.jar` from <https://scripting.dev.java.net/> to your CLASSPATH and run this program using Java 6. But since Groovy 1.6, Groovy already contains the groovy scripting engine compliant with JSR-223.

This next example illustrates calling an invokable function:

```

//
require(url:'https://scripting.dev.java.net'
, jar:'groovy-engine.jar')
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.Invocable;

public class FactMain {
    public static void main(String[] args)
throws Exception {
        ScriptEngineManager factory = new
ScriptEngineManager();
        ScriptEngine engine =
factory.getEngineByName("groovy");
        String fact = "def factorial(n) { n
== 1 ? 1 : n * factorial(n - 1) }";
        engine.eval(fact);
        Invocable inv = (Invocable) engine;
        Object[] params = { new Integer(5)
};
        Object result =
inv.invokeFunction("factorial", params);
        System.out.println(result);
    }
}

```

And when run, results in:

```
120
```

Groovy has many mechanisms for integration with Java, some of which provide richer options than available with JSR 223 (e.g. greater configurability and more security control). JSR 223 is recommended when you need to keep the choice of language used flexible and you don't require integration mechanisms not supported by JSR-223.

The engine keeps per default hard references to the script functions. To change this you should set an engine level scoped attribute to the script context of the name `"#jsr223.groovy.engine.keep.globals"` with a String being "phantom" to use phantom references, "weak" to use weak references or "soft" to use soft references - casing is ignored. Any other string will cause the use of hard references.

See also: [JSR-223 access to other JVM languages](#)

## Logging

- [Configuration](#)
- [@Log](#)
- [Tracing Method Calls](#)
- [See Also](#)

### Configuration

Basic configuration of the standard JDK to show logging methods can be accomplished with the following procedure:

In file `%JAVA_HOME%/jre/lib/logging.properties` or equivalent

- set the log handler is configured to show all levels

```
java.util.logging.ConsoleHandler.level = ALL
```

- set levels for the specific logging names that you wish to log

```
com.foo.level = FINE  
  
com.foo.MyClass.level = FINEST
```

### @Log

You can annotate your classes with the `@Log` transformation to automatically inject a logger in your Groovy classes, under the `log` property. Four kinds of loggers are actually available:

- [@Log](#) for `java.util.logging`
- [@Commons](#) for Commons-Logging
- [@Log4j](#) for Log4J
- [@Slf4j](#) for SLF4J

Here's a sample usage of the `@Log` transformation:

```

import groovy.util.logging.*

@Log
class Car {
    Car() {
        log.info 'Car constructed'
    }
}

def c = new Car()

```

You can change the name of the logger by specifying a different name, for instance with `@Log( 'myLoggerName' )`.

Another particularity of these logger AST transformations is that they take care of wrapping and safe-guarding logger calls with the usual `isSomeLevelEnabled()` calls. So when you write `log.info 'Car constructed'`, the generated code is actually equivalent to:

```

if (log.isLoggable(Level.INFO)) {
    log.info 'Car constructed'
}

```

### ***Tracing Method Calls***

In order to enable tracing of how Groovy calls MetaMethods, use the following settings:

in file `%JAVA_HOME%/jre/lib/logging.properties` or equivalent

- make sure your log handler is configured to show level 'FINER' at least, e.g.

```
java.util.logging.ConsoleHandler.level = ALL
```

- set MetaClass logging to 'FINER' at least, e.g.

```
groovy.lang.MetaClass.level = FINER
```

- set the appropriate Level for the Classes and optionally method names that you want to trace. The name for the appropriate logger starts with 'methodCalls' and optionally ends with the method name, e.g.

```
# trace all method calls
methodCalls.level = FINER

# trace method calls to the 'String' class
methodCalls.java.lang.String.level = FINER

# trace method calls to Object.println()
methodCalls.java.lang.Object.println.level = FINER
```

Example:

with tracing enabled for all method calls a Groovy command line script appears as follows (German locale)

```
$ groovy -e "println 'hi' "
13.09.2005 14:33:05 script_from_command_line
run()
FEINER: called from MetaClass.invokeMethod
13.09.2005 14:33:05 script_from_command_line
println('hi')
FEINER: called from MetaClass.invokeMethod
hi
```

See Also

- [groovy.util.logging](#) package

## Migration From Classic to JSR syntax

### Warning

This page was created to report about the changes in syntax when we switched to a new parser back in 2004. So some of the points below are not necessarily representative of what Groovy supports nowadays. Please take the notes below with a grain of salt!

Here is a checklist of changes you'll need to make to a Groovy classic codebase to ensure compatibility with the new Groovy JSR syntax.

## Safe navigation

In Classic Groovy we used to use this syntax

```
class Person = { String name }  
y = null  
println "${y->name}"    // -> was the  
optional gpath operator with Classic Groovy  
syntax
```

Instead of using the arrow operator for safe navigation to avoid NullPointerException, we're now using the ?. operator

Now in the JSR we use this syntax

```
class Person = { String name }  
def y = null  
println "${y?.name}"    // Now, ?. is the  
optional gpath operator the new Groovy JSR
```

#### Parameter separator in the closure syntax

This allows us to use one single token for the separator between the closure in Classic Groovy we used to use this syntax

```
cl = {a| ...}  
cl = {|a| ...}
```

Now in the JSR we use this syntax

```
def cl = {a -> ...}
```

This allows us to use one single token for the separator between the closure parameters and the code in in the closure which works with arbitrarily complex parameter list expressions and default values. e.g.

```
def cl = {String x = "hey", int y = a|b ->
println "Values are $x and $y"}
collection.each {int item -> println item}
```

### Property keyword

- property keyword has been replaced with an annotation

```
class Foo {
  property foo
}
```

```
class Foo {
  @Property foo
}
```

### Introduction of the 'def' keyword

- local variable declarations and fields currently need to be specified with 'def', a modifier, and/or a type. e.g.

```
def foo() {
  int x = 123
  y = 456 // classic
}

def foo() {
  int x = 123
  def y = 456 // JSR
}

class Foo {
  telson // classic
  int sharna
}

class Foo {
  def telson // JSR
  int sharna
}
```

(Syntactically, the new keyword 'def' acts for methods as a modifier like 'public'.)

For Scripts (as opposed to explicitly declared classes) the syntax is not changed, i.e. variable declarations without 'def' are still allowed, because those variables are automatically created in the script binding if they don't already exist.

## Introduction of the 'as' keyword

- We can change types of objects with using the 'as' keyword, e.g.

```
def d0 = new Date(2005-1900, 5-1, 7) // in classic Groovy or Java
println d0

def d1 = [2005-1900, 5-1, 8] as Date // since jsr-01
println d1

Date d2 = [2005-1900, 5-1, 9] as Date // since jsr-01
println d2

Date d3 = [2005-1900, 5-1, 10] // since jsr-01
println d3

// def n0 = new int[] { 1, 3, 5, 6 } // Not work. This style is not
supported since groovy-1.0-jsr-01.

def n1 = [ 1, 3, 5, 7 ] as int[]
println n1.class
println n1.size()
println n1.length
println n1[0]
println n1[-1]

// int[] n2 = [ 2, 4, 6, 8,10 ] as int[] // work
int[] n2 = [ 2, 4, 6, 8, 10 ] // work
println n2.class
println n2.size()
println n2.length
println n2[0]
println n2[-1]

// String[] n3 = [ "a", "ab", "abc", "abcd", "abcde", "abcdef" ] as
String[] // work
String[] n3 = [ "a", "ab", "abc", "abcd", "abcde", "abcdef" ]
// work
println n3.class
println n3.size()
println n3.length
println n3[0]
println n3[-1]
```

## Default access level of class members

The default access level for members of Groovy classes has changed from "public" to "protected"

Classic Groovy

```
class Foo {
    readMe_a;
    readMe_b
}
xyz = new
Foo(readMe_a:"Hello",readMe_b:"World")
println xyz.readMe_a
println xyz.readMe_b
```

Now in JSR Groovy

```
class Foo {
    public readMe_a;
    def readMe_b //def is now required
because of the "def keyword" change
mentioned earlier
}
xyz = new
Foo(readMe_a:"Hello",readMe_b:"World")
println xyz.readMe_a
println xyz.readMe_b //errors in JSR Groovy
```

#### Array creation

- no special array syntax. To make the language much cleaner, we now have a single syntax to work with lists and arrays in the JSR. Also note that we can now easily coerce from any collection or array to any array type

```
// classic
args = new String[] { "a", "b" }

// JSR
String[] args = [ "a", "b" ]
def x = [1, 2, 3] as int[]
long[] y = x
```

- **Be careful:** we don't support native multi-dimensional array creation right now.

#### float and double notation

- float and double literals cannot start with dot. So

```
x = .123 // classic
def x = 0.123 // JSR
```

This is to avoid ambiguity with things like ranges (1..2) and so forth

#### Explicit method pointer syntax

In classic Groovy you could access method pointers automatically if there was no java bean property of the given method name.

e.g.

```
// classic
methodPointer = System.out.println
methodPointer("Hello World")
```

This often caused confusion; as folks would use a property access to find something and get a method by accident (e.g. typo) and get confused. So now we make getting a method pointer explicit as follows

```
// JSR
def methodPointer = System.out.&println
methodPointer("Hello World")

def foo = ...
def p = foo.&bar

// lets call the bar method on the foo
object
p(1, 2, 3)
```

No 'do ... while()' syntax as yet.

Due to ambiguity, we've not yet added support for do .. while to Groovy

#### 'No Dumb Expression' rule

- no dumb expression rule, so we will catch dumb expressions (where a carriage return has broken the script).  
e.g.

```
def foo() {
  def x = 1
  +5 // dumb expression!
  return 8
}
```

#### Markup and builders

- markup / builders will change a little, but the classic syntax still applies. Not sure of the new syntax, but we'll have some kinda start/stop syntax to denote a markup block. Maybe a keyword, like

```
markup (builder) {
  // same stuff as before goes here
}

// or something like this
builder.{
  // same stuff as before goes here
}
```

### Strings and GStrings

- single and double quote strings can only span one line; for multiple lines use triple quotes
- heredocs removal - they are kinda ugly anyway 😊. If you want to use them, just use treble quote instead

```
def foo = """
this
is
a very
long
string on many
lines
"""
```

- escaping of \$ inside GStrings must use \\$

```
println 'amount is $100'
// same as
println "amount is \$100"
```

- A new string definition is also supported which is escaping friendly, and thus particularly friendly for regex notation:

```

if ('abc' =~ /.../) {}
if ('abc' ==~ /.../) {}
'abc'.eachMatch(/.../) {}
['a', 'b', 'c'].grep(/a/)

switch('abc'){
    case ~/.../ : whatever
}

assert 'EUOUAE'.matches(/^[aeiou]*$/)
assert 'EUOUAE' ==~ /^[aeiou]*$/
assert 'football'.replaceAll(/foo/, "Bar")
== 'Bartball'

```

#### Assertions

- assert uses comma instead of colon to delimit the two parameter form of assertion statement

```

assert 0 <= value : "Must be non-negative"
// classic

```

```

assert 0 <= value , "Must be non-negative"
// JSR

```

#### return/break/continue semantics in closures

#### NOT YET IMPLEMENTED

- return/break/continue to behave inside closures like these statements work in other blocks (such as the block on a for() or while() loop. More details here

#### Integer division

Previously, in Groovy Classic, we used the backward slash as the integer division operator. This operator being

confusing with escaping sequences was removed. So instead of `\` please use the `intdiv()` method.

```
int result = 5 \ 3 // classic
```

Becomes

```
int result = 5.intdiv(3) // JSR
```

### Exclusive range

The operator for creating ranges with the upper bound excluded from the range has changed.

Instead of:

```
range = 0...10
```

The syntax is now:

```
def range = 0..<10
```

## Processing XML

### Processing existing XML

Groovy provides special XML processing support through the following classes:

Technology	When/Why to use	Requirements
<a href="#">XmlParser</a>	supports GPath expressions for XML documents and allows updating	-
<a href="#">XmlSlurper</a>	lower overheads than XmlParser due to lazy evaluation and <a href="#">allows updating</a>	-
<a href="#">DOMCategory</a>	low-level tree-based processing where you want some syntactic sugar	place <code>use(DOMCategory)</code> around your code

If you have special needs, you can use one of the many available Java APIs for XML processing. You should consult the documentation of individual APIs for the details, but some examples to get you started are included here:

Technology	When/Why to use	Requirements
<a href="#">DOM</a>	low-level tree-based processing	-
<a href="#">SAX</a>	event-based push-style parsing can be useful for streaming large files	-
<a href="#">StAX</a>	event-based pull-style parsing can be useful for streaming large files	requires stax.jar
<a href="#">DOM4J</a>	nicer syntax over DOM processing plus can be useful for large files if you use prune capability	requires dom4j.jar
<a href="#">XOM</a>	nicer syntax over DOM processing plus a strong emphasis on compliancy	requires xom.jar
<a href="#">JDOM</a>	nicer syntax over DOM processing	requires jdom.jar
<a href="#">XPath</a>	use XPath expressions	requires Java 5+ or JAXP 1.3+ or xalan.jar or jaxen.jar
<a href="#">XSLT</a>	when your transformation is more easily expressed using XSLT than code	-
<a href="#">XQuery</a>	when your transformation is more easily expressed using XQuery than other alternatives	mxquery.jar or alternative

## Creating new XML

The most commonly used approach for creating XML with Groovy is to use a builder, i.e. one of:

Technology	When/Why to use	Requirements
<a href="#">MarkupBuilder</a>	supports Groovy's builder pattern with XML/HTML	-
<a href="#">StreamingMarkupBuilder</a>	for larger files	-

Groovy also has some low-level helper classes you typically won't need to use directly but you may sometimes see in older examples of using XML with Groovy.

Technology	When/Why to use	Requirements
SAXBuilder	support class when using SAX	-
StreamingSAXBuilder	streaming version of SAXBuilder	-
DOMBuilder	support class when using DOM	-
StreamingDOMBuilder	streaming version of DOMBuilder	-

You can also use Java API's which support XML document creation:

Technology	When/Why to use	Requirements
<a href="#">DOM</a>	low-level creation mechanism	-
<a href="#">JDOM</a>	if you are an existing JDOM user	requires jdom.jar
<a href="#">DOM4J</a>	if you are an existing DOM4J user	requires dom4j.jar
<a href="#">XOM</a>	if you are an existing XOM user	requires xom.jar

### Updating XML

- [Updating XML with XmlParser](#)
- [Updating XML with XmlSlurper](#)
- [Updating XML with DOMCategory](#)

### Validating XML

- [Validating XML with a DTD](#)
- [Validating XML with a W3C XML Schema](#)
- [Validating XML with RELAX NG](#)

### Troubleshooting

Some of the following Jira issues may indicate solutions or workarounds to problems other users faced while using XML.

Issue	Description
<a href="#">GROOVY-2126</a>	ClassCastException when using java.util.prefs package

### More Information

For some more discussion of the pro's and con's of your XML Processing options and some additional details, see the following books:

- Chapter 12 of [GINA](#)
- [Processing XML with Java](#)

- [Pro XML Development with Java Technology](#)
- [Java and XML](#)

The following articles may also be of interest:

- [Getting Groovy with XML](#) by Jack Herrington.
- [XML and Java technologies: Document models, Part 1: Performance](#)
- [DOM, DOM4J, JDOM, XOM Comparison](#) (slightly outdated)
- [Dom4J performance versus Xerces / Xalan](#)
- [Hints for parsing XHTML with DOM4J](#)

You may also be interested in how XML is applied in other parts of Groovy:

- [GroovyWS](#) for creating Web Services and Clients (or for JDK 1.4 with Groovy 1.5 also see [Groovy SOAP](#))
- [Testing Web Services](#)
- [Testing Web Applications](#)

## Creating XML using Groovy's MarkupBuilder

### Simple Example

Here is an example of using Groovy's MarkupBuilder to create a new XML file:

```
// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.0')
import groovy.xml.MarkupBuilder
import org.custommonkey.xmlunit.*

def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
xml.records() {
    car(name:'HSV Maloo', make:'Holden',
year:2006) {
        country('Australia')
        record(type:'speed', 'Production Pickup
Truck with speed of 271kph')
    }
    car(name:'P50', make:'Peel', year:1962) {
        country('Isle of Man')
        record(type:'size', 'Smallest
Street-Legal Car at 99cm wide and 59 kg in
```

```
weight')
    }
    car(name: 'Royale', make: 'Bugatti',
year:1931) {
        country('France')
        record(type: 'price', 'Most Valuable Car
at $15 million')
    }
}
```

```
XMLUnit.setIgnoreWhitespace(true)
```

```
def xmlDiff = new Diff(writer.toString(),  
XmlExamples.CAR_RECORDS)  
assert xmlDiff.similar()
```

We have used [XMLUnit](#) to compare the XML we created with our sample XML. To do this, make sure the sample XML is available, i.e. that the following class is added to your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

You may also want to see [Using MarkupBuilder for Agile XML creation](#).

Also, suppose we have an existing XML document and we want to automate generation of the markup without having to type it all in? We just need to use `DomToGroovy` as shown in the following example:

```
import
javax.xml.parsers.DocumentBuilderFactory
import
org.codehaus.groovy.tools.xml.DomToGroovy

def builder      =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
def inputStream = new
ByteArrayInputStream(XmlExamples.CAR_RECORDS
.bytes)
def document     = builder.parse(inputStream)
def output      = new StringWriter()
def converter    = new DomToGroovy(new
PrintWriter(output))

converter.print(document)
println output.toString()
```

Running this will produce the builder code for us.

#### Namespace Aware Example

Finally, here are some simple examples showing how to include namespaces and prefixes.

## Prefix with Namespace

```
def xml = new MarkupBuilder(writer)
xml.'rec:records' ('xmlns:rec':
'http://groovy.codehaus.org') {
    car(name:'HSV Maloo', make:'Holden',
year:2006) {
        country('Australia')
        record(type:'speed', ' Truck with speed
of 271kph')
    }
}
```

result

```
<rec:records
xmlns:rec='http://groovy.codehaus.org'>
    <car name='HSV Maloo' make='Holden'
year='2006'>
        <country>Australia</country>
        <record type='speed'> Truck with speed
of 271kph</record>
    </car>
</rec:records>
```

## Default Namespace

```
xml.records(xmlns:
'http://groovy.codehaus.org') {
    car(name:'HSV Maloo', make:'Holden',
year:2006) {
        country('Australia')
        record(type:'speed', ' Truck with speed
of 271kph')
    }
}
```

**result**

```
<records xmlns='http://groovy.codehaus.org'>
  <car name='HSV Maloo' make='Holden'
year='2006'>
    <country>Australia</country>
    <record type='speed'> Truck with speed
of 271kph</record>
  </car>
</records>
```

### Creating XML using Groovy's StreamingMarkupBuilder

Here is an example of using StreamingMarkupBuilder to create a new XML file:



```

// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.1beta2')
import groovy.xml.StreamingMarkupBuilder
import org.custommonkey.xmlunit.*

def xml = new StreamingMarkupBuilder().bind{
    records {
        car(name:'HSV Maloo', make:'Holden',
year:2006) {
            country('Australia')
            record(type:'speed', 'Production
Pickup Truck with speed of 271kph')
        }
        car(name:'P50', make:'Peel', year:1962)
    {
        country('Isle of Man')
        record(type:'size', 'Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight')
    }
        car(name:'Royale', make:'Bugatti',
year:1931) {
            country('France')
            record(type:'price', 'Most Valuable
Car at $15 million')
        } } }

XMLUnit.ignoreWhitespace = true
def xmlDiff = new Diff(xml.toString(),
XmlExamples.CAR_RECORDS)
assert xmlDiff.similar()

```

Note that `StreamingMarkupBuilder.bind` returns a [Writable](#) instance that may be used to stream the markup to a `writer`, in addition to capturing the output in a `String` (as shown above).

We have used [XMLUnit](#) to compare the XML we created with our sample XML. To do this, make sure the sample XML is available, i.e. that the following class is added to your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here's another example illustrating how to produce mixed content:



```

// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.1beta2')
import groovy.xml.StreamingMarkupBuilder
import org.custommonkey.xmlunit.*

def message = ' <b>wins</b> '
def xml = new StreamingMarkupBuilder().bind{
    html() {
        body bgcolor:'red' ) {
            h1('In Breaking News ...')
            p {
                a(href:'http://groovy.codehaus.org',
'Groovy')
                mkp.yieldUnescaped message

a(href:'http://jax-award.de/jax_award/gewinn
er_eng.php', 'Jax')
                mkp.yield ' praise & award.'
            } } } }

def expected = ''
<html>
    <body bgcolor='red'>
        <h1>In Breaking News ...</h1>
        <p>
            <a
href='http://groovy.codehaus.org'>Groovy</a>
            <b>wins</b>
            <a
href='http://jax-award.de/jax_award/gewinner
_eng.php'>Jax</a>

```



```
def xmlDiff = new Diff(xml.toString(),
expected)
assert xmlDiff.similar()
```

You may also want to see [Using MarkupBuilder for Agile XML creation](#).

### Creating XML with Groovy and DOM

Here is an example of using Java's DOM facilities to create a new XML file:

```
import
javax.xml.parsers.DocumentBuilderFactory
import org.custommonkey.xmlunit.*

@Grab('xmlunit:xmlunit:1.3')

def addCar(document, root, name, make, year,
country, type, text) {
    def car = document.createElement('car')
    car.setAttribute('name', name)
    car.setAttribute('make', make)
    car.setAttribute('year', year)
    root.appendChild(car)
    def countryNode =
document.createElement('country')

countryNode.appendChild(document.createTextNode(
country))
    car.appendChild(countryNode)
    def record =
document.createElement('record')
    record.setAttribute('type', type)
```

```
record.appendChild(document.createTextNode(t
ext))
    car.appendChild(record)
}
```

```
def builder =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
def document = builder.newDocument()
def root =
document.createElement('records')
```

```
document.appendChild(root)
addCar(document, root, 'HSV Maloo',
'Holden', '2006', 'Australia',
'speed', 'Production Pickup Truck
with speed of 271kph')
addCar(document, root, 'P50', 'Peel',
'1962', 'Isle of Man',
'size', 'Smallest Street-Legal Car
at 99cm wide and 59 kg in weight')
addCar(document, root, 'Royale', 'Bugatti',
'1931', 'France',
'price', 'Most Valuable Car at $15
million')
```

```
// now load in our XML sample and compare it
to our newly created document
def builder2 =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
```

```
def inputStream = new  
ByteArrayInputStream(XmlExamples.CAR_RECORDS  
.bytes)  
def control      =  
builder2.parse(inputStream)
```

```
XMLUnit.setIgnoreWhitespace(true)  
def xmlDiff = new Diff(document, control)  
assert xmlDiff.similar()
```

We have used [XMLUnit](#) to compare the XML we created with our sample XML. To do this, make sure the sample XML is available, i.e. that the following class is added to your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

You may also want to see [Using MarkupBuilder for Agile XML creation](#).

Processing XML with XSLT

Groovy's [XML capabilities](#) are powerful enough that you may not need to consider other alternatives such as [XSLT](#). If you wish to use XSLT though, it isn't hard to do. Here we just added xalan and serializer jars from the Apache [Xalan-J](#) project to our CLASSPATH. Depending on your JVM version and other available libraries, you may need slightly different jars (or none at all). Once our path is set up, we can run the following script to convert an XML file into XHTML:

```
//
require(url='http://xml.apache.org/xalan-j/'
, jar='serializer.jar')
//
require(url='http://xml.apache.org/xalan-j/'
, jar='xalan_270.jar')
import
javax.xml.transform.TransformerFactory
import
javax.xml.transform.stream.StreamResult
import
javax.xml.transform.stream.StreamSource

def input = '''
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family_name>Smith</family_name>
  </person>
  <person username="ND1">
    <name>Nancy</name>
    <family_name>Davolio</family_name>
  </person>
</persons>
'''.trim()
```

```

def xslt = '''
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/persons">
    <html
xmlns="http://www.w3.org/1999/xhtml">
      <head>
        <title>Testing XML
Example</title>
      </head>
      <body>
        <h1>Persons</h1>
        <ul>
          <xsl:apply-templates
select="person">
            <xsl:sort
select="family_name" />
          </xsl:apply-templates>
        </ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="person">
    <li>
      <xsl:value-of
select="family_name" />,
      <xsl:value-of select="name" />
    </li>
  </xsl:template>
</xsl:stylesheet>

```

```
'''.trim()
```

```
def factory =
```

```
TransformerFactory.newInstance()
```

```
def transformer = factory.newTransformer(new  
StreamSource(new StringReader(xslt)))
```

```
transformer.transform(new StreamSource(new
StringReader(input)), new
StreamResult(System.out))
```

Here is the output (pretty-printed):

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Testing XML
Example</title></head>
<body><h1>Persons</h1>
<ul>
  <li>Davolio, Nancy</li>
  <li>Smith, John</li>
</ul>
</body>
</html>
```

Depending on your environment (JVM, XSLT processor, operating system) you might be able to make use of XSLT directly from the command line instead of calling it from Java or Groovy. You can also call it from Ant using the `xs1t` task.

## Reading XML using Groovy's DOMCategory

### Information

Java has in-built support for DOM processing of XML using classes representing the various parts of XML documents, e.g. *Document*, *Element*, *NodeList*, *Attr* etc. For more information about these classes, refer to the respective JavaDocs. Some of the key classes are:

DOM class	JavaDocs
Element	<a href="#">67</a>
NodeList	<a href="#">67</a>

Groovy syntax benefits can be applied when using these classes resulting in code which is similar to but more compact than the Java equivalent. In addition, Groovy supports the following built-in helper method for these classes.

DOM class	Method	Description/Equivalent longhand
<i>NodeList</i>	<i>.iterator()</i>	same as for loop, enables closures, e.g. <i>findAll</i> , <i>every</i> , etc.

In addition, the *DOMCategory* class provides numerous additional helper methods and syntax shortcuts:

DOM class	Method	Description/Equivalent longhand
<i>Element</i>	<i>.child</i> or <i>.child</i> or [ <i>child</i> ]	similar to <i>.getElementsByTagName('child')</i> but only gets direct children
<i>Element</i>	<i>.children()</i> or <i>.**</i> or [ <i>**</i> ]	special case of above which finds all children regardless of tagname (plus text nodes)
<i>Element</i>	<i>.@attr</i> or [ <i>@attr</i> ]	<i>.getAttribute('attr')</i>
<i>Element</i>	<i>.attributes()</i>	equivalent to <i>.attributes</i> returns a <i>NamedNodeMap</i>
<i>Element</i>	<i>.text()</i>	<i>.firstChild.nodeValue</i> (or <i>textContent</i> if you are using Xerces)
<i>Element</i>	<i>.name()</i>	<i>.nodeName</i>
<i>Element</i>	<i>.parent()</i> or <i>..'</i> or [ <i>..</i> ]	<i>.parentNode</i>
<i>Element</i>	<i>.depthFirst()</i> or <i>.***</i>	depth-first traversal of nested children
<i>Element</i>	<i>.breadthFirst()</i>	breadth-first traversal of nested children
<i>Node</i>	<i>.toString()</i>	text node value as a <i>String</i>
<i>NodeList</i>	<i>.size()</i>	<i>.length</i>
<i>NodeList</i>	<i>.list()</i>	converted to a list of nodes
<i>NodeList</i>	[ <i>n</i> ]	<i>.item(n)</i>
<i>NodeList</i>	<i>.text()</i>	<i>.collect{ it.text() }</i>
<i>NodeList</i>	<i>.child</i>	flattened version of <i>.child</i> for each node in the <i>NodeList</i>

<i>NamedNodeMap</i>	<i>.size()</i>	<i>.length</i>
<i>NamedNodeMap</i>	<i>.'child' or .child or ['child']</i>	<i>.getNamedItem(elementName).nodeValue</i>

All these methods return standard Java classes (e.g. *String* and *List*) or standard DOM classes (e.g. *Element*, *Node* *List*), so there are no new classes to learn, just some improved syntax.

**Example**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using Groovy's DOMCategory:



```

import groovy.xml.DOMBuilder
import groovy.xml.dom.DOMCategory

messages = []

def processCar(car) {
    assert car.name() == 'car'
    def make = car.'@make'
    def country = car.country[0].text()
    def type = car.record[0].'@type'
    messages << make + ' of ' + country + '
has a ' + type + ' record'
}

def reader = new
StringReader(XmlExamples.CAR_RECORDS)
def doc = DOMBuilder.parse(reader)
def records = doc.documentElement

use (DOMCategory) {
    assert 3 == records.'*'.size()
    def cars = records.'car'
    assert cars[0].parent() == records
    assert 3 == cars.size()
    assert 2 == cars.findAll{
it.'@year'.toInteger() > 1950 }.size()
    def carsByCentury = cars.list().groupBy{
        it.'@year'.toInteger() >= 2000 ?
'this century' : 'last century'
    }
    assert 1 == carsByCentury['this

```

```
century'].size()  
    assert 2 == carsByCentury['last  
century'].size()  
    cars.each{ car -> processCar(car) }  
}
```

```
assert messages == [  
    'Holden of Australia has a speed  
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

### **Reading XML using Groovy's XmlParser**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using XmlParser:



```
def records = new
XmlParser().parseText(XmlExamples.CAR_RECORD
S)
def allRecords = records.car.size()
assert allRecords == 3
def allNodes = records.depthFirst().size()
assert allNodes == 10
def firstRecord = records.car[0]
assert 'car' == firstRecord.name()
assert 'Holden' == firstRecord.'@make'
assert 'Australia' ==
firstRecord.country.text()
// 2 cars have an 'e' in the make
assert 2 == records.car.findAll{
it.'@make'.contains('e') }.size()
// makes of cars that have an 's' followed
by an 'a' in the country
assert ['Holden', 'Peel'] ==
records.car.findAll{ it.country.text() =~
'*.s*a.*' }.@make'
// types of records
assert ['speed', 'size', 'price'] ==
records.depthFirst().grep{ it.'@type'
}.'@type'
// update to show what would happen if 'New
Zealand' bought Holden
firstRecord.country[0].value = ['New
Zealand']
assert 'New Zealand' ==
firstRecord.country.text()
// names of cars with records sorted by year
```

```
assert ['Royale', 'P50', 'HSV Maloo'] ==
records.car.sort{ it.'@year'.toInteger()
}. '@name'
```

It is also possible to parse XML documents using namespaces:

```
def ns = new
groovy.xml.Namespace("http://www.example.com
/book", 'ns')
def root = new XmlParser().parseText("...")
def authors = root[ns.book][ns.author]
```

### Reading XML using Groovy's XmlSlurper

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using XmlSlurper:



```
def records = new
XmlSlurper().parseText(XmlExamples.CAR_RECOR
DS)
def allRecords = records.car
assert 3 == allRecords.size()
def allNodes = records.depthFirst().collect{
it }
assert 10 == allNodes.size()
def firstRecord = records.car[0]
assert 'car' == firstRecord.name()
assert 'Holden' == firstRecord.@make.text()
assert 'Australia' ==
firstRecord.country.text()
def carsWith_e_InMake = records.car.findAll{
it.@make.text().contains('e') }
assert carsWith_e_InMake.size() == 2
// alternative way to find cars with 'e' in
make
assert 2 == records.car.findAll{ it.@make =~
'.*e.*' }.size()
// makes of cars that have an 's' followed
by an 'a' in the country
assert ['Holden', 'Peel'] ==
records.car.findAll{ it.country =~
'.*s.*a.*' }.@make.collect{ it.text() }
def expectedRecordTypes = ['speed', 'size',
'price']
assert expectedRecordTypes ==
records.depthFirst().grep{ it.@type != ''
}. '@type'*.text()
assert expectedRecordTypes ==
```

```
records.'**'.grep{ it.@type != ''
}.'@type'*.text()
def countryOne = records.car[1].country
assert 'Peel' ==
countryOne.parent().@make.text()
assert 'Peel' ==
countryOne..'..'@make.text()
// names of cars with records sorted by year
def sortedNames = records.car.list().sort{
it.@year.toInteger() }. '@name'*.text()
assert ['Royale', 'P50', 'HSV Maloo'] ==
sortedNames
assert ['Australia', 'Isle of Man'] ==
records.'**'.grep{ it.@type =~ 's.*'
}*.parent().country*.text()
assert 'co-re-co-re-co-re' ==
records.car.children().collect{
it.name()[0..1] }.join('-')
```

```
assert 'co-re-co-re-co-re' ==  
records.car.*.collect{ it.name()[0..1]  
}.join('-')
```

Notes:

- If your elements contain characters such as dashes, you can enclose the element name in double quotes. Meaning for:

```
<foo>  
  <foo-bar>test</foo-bar>  
</foo>
```

- You do the following:

```
def foo = new XmlSlurper().parseText(FOO_XML)  
assert "test" == foo."foo-bar".text()
```

You can also parse XML documents using namespaces:

```

def wsd1 = '''
<definitions name="AgencyManagementService"
  xmlns:ns1="http://www.example.org/NS1"
  xmlns:ns2="http://www.example.org/NS2">
  <ns1:message name="SomeRequest">
    <ns1:part name="parameters"
element="SomeReq" />
  </ns1:message>
  <ns2:message name="SomeRequest">
    <ns2:part name="parameters"
element="SomeReq" />
  </ns2:message>
</definitions>
'''

def xml = new
XmlSlurper().parseText(wsd1).declareNamespac
e(ns1: 'http://www.example.org/NS1', ns2:
'http://www.example.org/NS2')
println xml.'ns1:message'. 'ns1:part'.size()
println xml.'ns2:message'. 'ns2:part'.size()

```

XmlSlurper has a declareNamespace method which takes a Map of prefix to URI mappings. You declare the namespaces and just use the prefixes in the GPath expression.

```

new
XmlSlurper().parseText(blog).declareNamespac
e(dc:
"http://purl.org/dc/elements/1.1/").channel.
item.findAll { item ->
    d.any{entry -> item."dc:date".text() =~
entry.key} && a.any{entry ->
item.tags.text() =~ entry
}

```

Some remarks:

- name or "\*:name" matches an element named "name" irrespective of the namespace it's in (i.e. this is the default mode of operation)
- ":name" matches an element named "name" only if the element is not in a namespace
- "prefix:name" matches an element named "name" only if it is in the namespace identified by the prefix "prefix" (and the prefix to namespace mapping was defined by a previous call to declareNamespace)

You can generate namespaced elements in StreamingMarkupBuilder very easily:

```

System.out << new
StreamingMarkupBuilder().bind {
    mkp.declareNamespace(dc:
"http://purl.org/dc/elements/1.1/")

    root {
        dc.date()
    }
}

```

## Reading XML with Groovy and DOM

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using DOM with Groovy to read an existing XML file:



```
import
javax.xml.parsers.DocumentBuilderFactory

messages = []

def processCar(car) {
  if (car.nodeName != 'car') return
  def make =
car.attributes.getNamedItem('make').nodeValue
  def country =
car.getElementsByTagName('country').item(0).
firstChild.nodeValue
  def type = car.childNodes.find{'record'
==
it.nodeName}.attributes.getNamedItem('type')
.nodeValue
  messages << make + ' of ' + country + '
has a ' + type + ' record'
}

def builder =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
def inputStream = new
ByteArrayInputStream(XmlExamples.CAR_RECORDS
.bytes)
def records =
builder.parse(inputStream).documentElement

def cars = records.childNodes
```

```
(0..<cars.length).each{  
processCar(cars.item(it)) }
```

```
assert messages == [  
  'Holden of Australia has a speed  
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

## **Reading XML with Groovy and SAX**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using SAX with Groovy:

```
import javax.xml.parsers.SAXParserFactory
import org.xml.sax.helpers.DefaultHandler
import org.xml.sax.*

class RecordsHandler extends DefaultHandler
{
    def messages = []
    def currentMessage
    def countryFlag = false
    void startElement(String ns, String
localName, String qName, Attributes atts) {
        switch (qName) {
            case 'car':
                currentMessage =
atts.getValue('make') + ' of '; break
            case 'country':
                countryFlag = true; break
            case 'record':
                currentMessage +=
atts.getValue('type') + ' record'; break
        }
    }
    void characters(char[] chars, int
offset, int length) {
        if (countryFlag) {
            currentMessage += new
String(chars, offset, length)
        }
    }
    void endElement(String ns, String
localName, String qName) {
```

```
        switch (qName) {
            case 'car':
                messages << currentMessage;
break
            case 'country':
                currentMessage += ' has a ';
countryFlag = false; break
        }
    }
}
```

```
def handler = new RecordsHandler()
def reader =
SAXParserFactory.newInstance().newSAXParser(
).XMLReader
reader.setContentHandler(handler)
def inputStream = new
ByteArrayInputStream(XmlExamples.CAR_RECORDS
.getBytes("UTF-8"))
reader.parse(new InputSource(inputStream))

assert handler.messages == [
    'Holden of Australia has a speed
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

## **Reading XML with Groovy and StAX**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of reading an existing XML file with Groovy and [StAX](#):

```

// require(groupId:'stax',
artifactId:'stax-api', version:'1.0.1')
// require(groupId:'stax',
artifactId:'stax', version:'1.2.0')
import javax.xml.stream.*

messages = []
currentMessage = ''

def processStream(inputStream) {
    def reader
    try {
        reader =
XMLInputFactory.newInstance()

.createXMLStreamReader(inputStream)
        while (reader.hasNext()) {
            if (reader.startElement)
                processStartElement(reader)
            reader.next()
        }
    } finally {
        reader?.close()
    }
}

def processStartElement(element) {
    switch(element.name()) {
        case 'car':
            currentMessage = element.make +
" of "

```

```

        break
    case 'country':
        currentMessage += element.text()
+ " has a "
        break
    case 'record':
        currentMessage += element.type +
" record"
        messages << currentMessage
        break
    }
}

```

```

class StaxCategory {
    static Object get(XMLStreamReader self,
String key) {
        return self.getAttributeValue(null,
key)
    }
    static String name(XMLStreamReader self)
{
        return self.name.toString()
    }
    static String text(XMLStreamReader self)
{
        return self.elementText
    }
}

```

```

def bytes = XmlExamples.CAR_RECORDS.bytes
def inputStream = new

```

```
ByteArrayInputStream(bytes)
use (StaxCategory) {
processStream(inputStream) }

assert messages == [
    'Holden of Australia has a speed
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

## **Reading XML with Groovy and XPath**

### *Car Example*

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

### Using Xalan

Here is an example of using Apache [Xalan](#) with Groovy to read an existing XML file:

---

```
// require(groupId:'xalan',
artifactId:'xalan', version:'2.6.0')
import org.apache.xpath.XPathAPI
import
javax.xml.parsers.DocumentBuilderFactory

messages = []

def processCar(car) {
    def make = XPathAPI.eval(car,
'@make').str()
    def country = XPathAPI.eval(car,
'country/text()').str()
    def type = XPathAPI.eval(car,
'record/@type').str()
    messages << make + ' of ' + country + '
has a ' + type + ' record'
}

def builder =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
def inputStream = new
ByteArrayInputStream(XmlExamples.CAR_RECORDS
.bytes)
def records =
builder.parse(inputStream).documentElement

XPathAPI.selectNodeList(records,
'//car').each{ processCar(it) }
```

```
assert messages == [  
    'Holden of Australia has a speed  
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

#### Using Jaxen

Here is an example of using [Jaxen](#) with Groovy to read an existing XML file:

```
// require(groupId:'jaxen',  
artifactId:'jaxen', version:'1.1-beta-10')  
import org.jaxen.dom.DOMXPath  
import  
javax.xml.parsers.DocumentBuilderFactory  
  
messages = []  
  
def processCar(car) {  
    def make = new  
DOMXPath('@make').stringValueOf(car)  
    def country = new  
DOMXPath('country/text()').stringValueOf(car  
)  
    def type = new  
DOMXPath('record/@type').stringValueOf(car)  
    messages << make + ' of ' + country + '  
has a ' + type + ' record'  
}  
  
def builder =  
DocumentBuilderFactory.newInstance().newDocu  
mentBuilder()  
def inputStream = new
```

```
ByteArrayInputStream(XmlExamples.CAR_RECORDS
.bytes)
def records      =
builder.parse(inputStream).documentElement

new
DOMXPath('//car').selectNodes(records).each{
processCar(it) }

assert messages == [
  'Holden of Australia has a speed
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

Note: many libraries (e.g. DOM4J, JDOM, XOM) bundle or provide optional support for Jaxen. You may not need to download any additional JARs to use it.

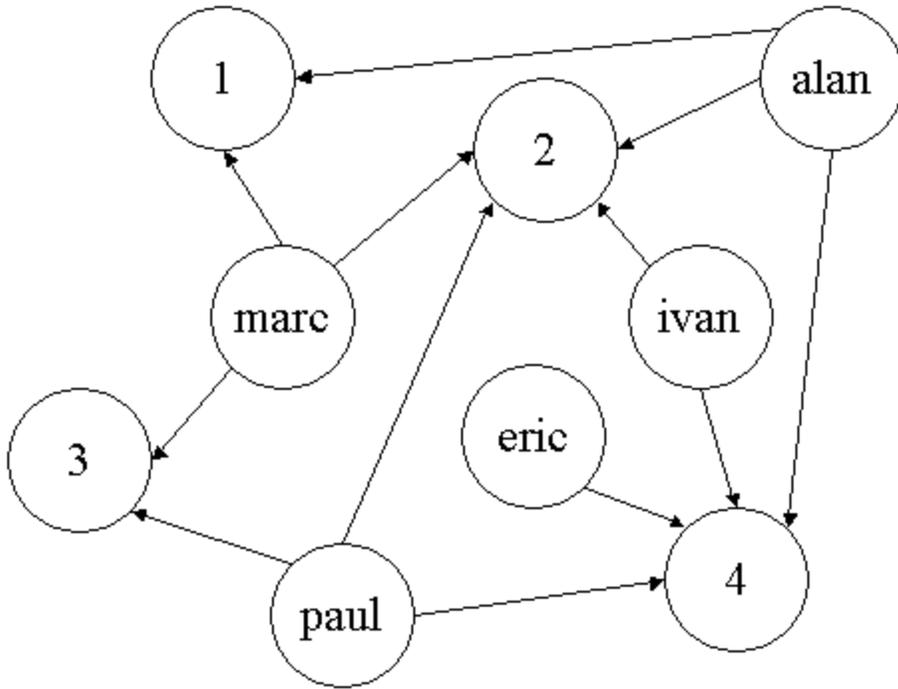
#### *Using Java's native XPath support*

If you are using Java 5 and above, you don't need to use the standalone Xalan jar but instead can use the now built-in XPath facilities:

```
import javax.xml.xpath.*  
  
def xpath =  
XPathFactory.newInstance().newXPath()  
  
def nodes = xpath.evaluate( '//car',  
records, XPathConstants.NODESET )  
nodes.each{  
  def make = xpath.evaluate( '@make', it )  
  def country = xpath.evaluate(  
'country/text()', it )  
  def type = xpath.evaluate( 'record/@type',  
it )  
  messages << "$make of $country has a $type  
record"  
}
```

#### *People in Groups Example*

Inspired by [this example](#), here is how to use XPath to determine which groups include all three of *alan*, *paul* and *iva* for this graph representing group membership.



Picture source: <http://stage.vambenepe.com/pages/graph.png>

First, here is the XML data capturing the group membership information:

```
def xml = '''
<doc>
  <person
name="alan"><g>1</g><g>2</g><g>4</g></person
>
  <person
name="marc"><g>1</g><g>2</g><g>3</g></person
>
  <person
name="paul"><g>2</g><g>3</g><g>4</g></person
>
  <person
name="ivan"><g>2</g><g>4</g></person>
  <person name="eric"><g>4</g></person>
</doc>
'''
```

*Using Java's native XPath support*

Here is how to check that groups 2 and 4 are the groups in question using the built-in XPath capabilities of Java 5 and above:

```

import
javax.xml.parsers.DocumentBuilderFactory
import javax.xml.xpath.*

xpath = '''
    /doc/person[@name="alan"]/g
    [./doc/person[@name="paul"]/g]
    [./doc/person[@name="ivan"]/g]
    '''

builder =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
doc      = builder.parse(new
ByteArrayInputStream(xml.bytes))
expr     =
XPathFactory.newInstance().newXPath().compil
e(xpath)
nodes    = expr.evaluate(doc,
XPathConstants.NODESET)
assert nodes.collect { node ->
node.textContent } == ['2', '4']

```

And you can refactor your XPath Expression with something like:

```

import
javax.xml.parsers.DocumentBuilderFactory
import javax.xml.xpath.*

groupsOf = { name ->
"/doc/person[@name='$name']/g" }
xpath     = "${groupsOf 'alan'}[.=${groupsOf
'paul'}][.=${groupsOf 'ivan'}]"
builder   =
DocumentBuilderFactory.newInstance().newDocu
mentBuilder()
doc       = builder.parse(new
ByteArrayInputStream(xml.bytes))
expr      =
XPathFactory.newInstance().newXPath().compil
e(xpath)
nodes     = expr.evaluate(doc,
XPathConstants.NODESET)
assert    nodes.collect { node ->
node.textContent } == ['2', '4']

```

#### *Using GPath*

Of course, you can also do this without XPath by using Groovy's GPath facilities as follows:

```

def root = new XmlParser().parseText(xml)
Set groups = root.person.g.collect{
it.text() }
assert groups.findAll{ group ->
    root.person.g.findAll{
        it.text() == group
    }. '..'.'@name'.containsAll(['alan',
'paul', 'ivan'])
} == ['2', '4']

```

### Updating XML with DOMCategory

Here is an example of updating XML using DOMCategory:

```

@Grab('xmlunit:xmlunit:1.3')
import groovy.xml.XmlUtil
import groovy.xml.dom.DOMCategory
import groovy.xml.DOMBuilder
import org.custommonkey.xmlunit.Diff
import org.custommonkey.xmlunit.XMLUnit

def input = '''
<shopping>
    <category type="groceries">
        <item>Chocolate</item>
        <item>Coffee</item>
    </category>
    <category type="supplies">
        <item>Paper</item>
        <item quantity="4">Pens</item>

```

```
    </category>
    <category type="present">
      <item when="Aug 10">Kathryn's
Birthday</item>
    </category>
  </shopping>
  '''
```

```
def expectedResult = '''
<shopping>
  <category type="groceries">
    <item>Luxury Chocolate</item>
    <item>Luxury Coffee</item>
  </category>
  <category type="supplies">
    <item>Paper</item>
    <item quantity="6"
when="Urgent">Pens</item>
  </category>
  <category type="present">
    <item>Mum's Birthday</item>
    <item when="Oct 15">Monica's
Birthday</item>
  </category>
</shopping>
'''
```

```
def reader = new StringReader(input)
def doc = DOMBuilder.parse(reader)
def root = doc.documentElement
```

```
use(DOMCategory) {
    // modify groceries: quality items
    please
    def groceries = root.category.findAll{
it.'@type' == 'groceries' }[0].item
    groceries.each { g ->
        g.value = 'Luxury ' + g.text()
    }

    // modify supplies: we need extra pens
    def supplies = root.category.findAll{
it.'@type' == 'supplies' }[0].item
    supplies.findAll{ it.text() == 'Pens'
}.each { s ->
        s['@quantity'] =
s.'@quantity'.toInteger() + 2
        s['@when'] = 'Urgent'
    }

    // modify presents: August has come and
    gone
    def presents = root.category.find{
it.'@type' == 'present' }
    presents.item.each {
        presents.removeChild(it)
    }
    presents.appendChild('item', "Mum's
    Birthday")
    presents.appendChild('item', [when:'Oct
    15'], "Monica's Birthday")
}
```

```
    // check the when attributes
    assert root.**.item.'@when'.grep{it}
== [ "Urgent", "Oct 15" ]
}
```

```
// check the whole document using XmlUnit
XMLUnit.setIgnoreWhitespace(true)
def result = XmlUtil.serialize(root)
```

```
def xmlDiff = new Diff(result,
expectedResult)
assert xmlDiff.identical()
```

## Updating XML with XmlParser

Here is an example of updating XML using XmlParser:

```
// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.1')
import org.custommonkey.xmlunit.Diff
import org.custommonkey.xmlunit.XMLUnit

def input = '''
<shopping>
  <category type="groceries">
    <item>Chocolate</item>
    <item>Coffee</item>
  </category>
  <category type="supplies">
    <item>Paper</item>
    <item quantity="4">Pens</item>
  </category>
  <category type="present">
    <item when="Aug 10">Kathryn's
Birthday</item>
  </category>
</shopping>
'''

def expectedResult = '''
```

```

<shopping>
  <category type="groceries">
    <item>Luxury Chocolate</item>
    <item>Luxury Coffee</item>
  </category>
  <category type="supplies">
    <item>Paper</item>
    <item quantity="6"
when="Urgent">Pens</item>
  </category>
  <category type="present">
    <item>Mum's Birthday</item>
    <item when="Oct 15">Monica's
Birthday</item>
  </category>
</shopping>
' ' '

```

```

def root = new XmlParser().parseText(input)

```

```

// modify groceries: quality items please

```

```

def groceries = root.category.findAll{
it.@type == 'groceries' }.item[0]
groceries.each { g ->
  g.value = 'Luxury ' + g.text()
}

```

```

// modify supplies: we need extra pens

```

```

def supplies = root.category.findAll{
it.@type == 'supplies' }.item[0]
supplies.findAll{ it.text() == 'Pens' }.each

```

```
{ s ->
    s.@quantity = s.@quantity.toInteger() +
2
    s.@when = 'Urgent'
}

// modify presents: August has come and gone
def presentCategory = root.category.find{
it.@type == 'present' }
presentCategory.children().clear()
presentCategory.appendNode('item', "Mum's
Birthday")
presentCategory.appendNode('item',
[when:'Oct 15'], "Monica's Birthday")

// check the when attributes
def removeNulls(list) { list.grep{it} }
assert removeNulls(root.**.item.@when) ==
["Urgent", "Oct 15"]

// check the whole document using XmlUnit
def writer = new StringWriter()
new XmlNodePrinter(new
PrintWriter(writer)).print(root)
def result = writer.toString()

XMLUnit.setIgnoreWhitespace(true)
```

```
def xmlDiff = new Diff(result,
expectedResult)
assert xmlDiff.identical()
```

## Updating XML with XmlSlurper

Here is an example of updating XML using XmlSlurper:

```
// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.1')
import org.custommonkey.xmlunit.Diff
import org.custommonkey.xmlunit.XMLUnit
import groovy.xml.StreamingMarkupBuilder

def input = '''
<shopping>
  <category type="groceries">
    <item>Chocolate</item>
    <item>Coffee</item>
  </category>
  <category type="supplies">
    <item>Paper</item>
    <item quantity="4">Pens</item>
  </category>
  <category type="present">
    <item when="Aug 10">Kathryn's
Birthday</item>
  </category>
</shopping>
'''
```

```

def expectedResult = ''
<shopping>
  <category type="groceries">
    </category>
    <category type="supplies">
      <item>Iced Tea</item>
      <item quantity="6"
when="Urgent">Pens</item>
    </category>
    <category type="present">
      <item>Mum's Birthday</item>
      <item when="Oct 15">Monica's
Birthday</item>
    </category>
    <category>
      <item>Wine</item>
    </category>
</shopping>
''

```

```

def root = new XmlSlurper().parseText(input)

// modify groceries: quality items please
def groceries = root.category.find{ it.@type
== 'groceries' }
(0..<groceries.item.size()).each {
  groceries.item[it] = 'Luxury ' +
groceries.item[it]
}

// modify supplies: we need extra pens

```

```
def pens = root.category.find{ it.@type ==
'supplies' }.item.findAll{ it.text() ==
'Pens' }
pens.each { p ->
    p.@quantity = (p.@quantity.toInteger() +
2).toString()
    p.@when = 'Urgent'
}
```

```
// modify presents: August has come and gone
def presents = root.category.find{ it.@type
== 'present' }
presents.replaceNode{ node ->
    category(type:'present'){
        item("Mum's Birthday")
        item("Monica's Birthday", when:'Oct
15')
    }
}
```

```
// append child at the end of shopping
root.appendNode {
    category {
        item("Wine")
    }
}
```

```
// delete all occurrences of a specific
element inside a specific parent
root.category[0].item.replaceNode {}
```

```
// update the text of a specific element
root.category[1].item[0] = "Iced Tea"

// check the whole document using XmlUnit
def outputBuilder = new
StreamingMarkupBuilder()
String result = outputBuilder.bind{
mkp.yield root }

XMLUnit.setIgnoreWhitespace(true)
def xmlDiff = new Diff(result,
expectedResult)
assert xmlDiff.similar()

// check the when attributes (can't do
before now due to delayed setting)
def resultRoot = new
XmlSlurper().parseText(result)
def removeNulls(list) { list.grep{it} }
```

```
assert
removeNulls(resultRoot.'*'.item.@when) == [
"Urgent", "Oct 15" ]
```

## Using Other XML Libraries

Groovy integrates with pretty much any third-party XML library in Java:

- [Creating XML with Groovy and DOM4J](#)
- [Creating XML with Groovy and JDOM](#)
- [Creating XML with Groovy and XOM](#)
- [Processing XML with XQuery](#)
- [Reading XML with Groovy and DOM4J](#)
- [Reading XML with Groovy and JDOM](#)
- [Reading XML with Groovy and XOM](#)

### Creating XML with Groovy and DOM4J

Here is an example of using DOM4J to create a new XML file:

```
// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.0')
// require(groupId:'dom4j',
artifactId:'dom4j', version:'1.6.1')
import
javax.xml.parsers.DocumentBuilderFactory
import org.custommonkey.xmlunit.*
import org.dom4j.io.XMLWriter
import org.dom4j.*

def addCar(root, name, make, year, country,
type, text) {
    def car = root.addElement('car')
    car.addAttribute('name', name)
    car.addAttribute('make', make)
    car.addAttribute('year', year)
    def countryCode =
```

```
car.addElement('country').addText(country)
    def record =
car.addElement('record').addText(text)
    record.addAttribute('type', type)
}

def document =
DocumentHelper.createDocument()
def root      =
document.addElement('records')

addCar(root, 'HSV Maloo', 'Holden', '2006',
'Australia',
    'speed', 'Production Pickup Truck
with speed of 271kph')
addCar(root, 'P50', 'Peel', '1962', 'Isle of
Man',
    'size', 'Smallest Street-Legal Car
at 99cm wide and 59 kg in weight')
addCar(root, 'Royale', 'Bugatti', '1931',
'France',
    'price', 'Most Valuable Car at $15
million')

// convert resulting document to a string so
that we can compare
XMLUnit.setIgnoreWhitespace(true)
def writer = new StringWriter()
new XMLWriter(writer).writeNode(document)
```

```
def xmlDiff = new Diff(writer.toString(),  
XmlExamples.CAR_RECORDS)  
assert xmlDiff.similar()
```

We have used [XMLUnit](#) to compare the XML we created with our sample XML. To do this, make sure the sample XML is available, i.e. that the following class is added to your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

You may also want to see [Using MarkupBuilder for Agile XML creation](#).

Creating XML with Groovy and JDOM

Here is an example of using [JDOM](#) to create a new XML file:

```
// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.0')
// require(groupId:'jdom',
artifactId:'jdom', version:'1.0')
import
javax.xml.parsers.DocumentBuilderFactory
import org.custommonkey.xmlunit.*
import org.jdom.output.XMLOutputter
import org.jdom.*

def addCar(root, name, make, year, country,
type, text) {
    def car = new Element('car')
    car.setAttribute('name', name)
    car.setAttribute('make', make)
    car.setAttribute('year', year)
    root.addContent(car)
    def countryNode = new
Element('country').setText(country)
    car.addContent(countryNode)
    def record = new
Element('record').setText(text)
    record.setAttribute('type', type)
    car.addContent(record)
}

def root = new Element('records')
def document = new Document(root)
document.setRootElement(root)
```

```
addCar(root, 'HSV Maloo', 'Holden', '2006',
        'Australia',
            'speed', 'Production Pickup Truck
with speed of 271kph')
addCar(root, 'P50', 'Peel', '1962', 'Isle of
Man',
        'size', 'Smallest Street-Legal Car
at 99cm wide and 59 kg in weight')
addCar(root, 'Royale', 'Bugatti', '1931',
        'France',
            'price', 'Most Valuable Car at $15
million')

// convert resulting document to a string so
that we can compare
XMLUnit.setIgnoreWhitespace(true)
def writer = new StringWriter()
new XMLOutputter().output(document, writer)
```

```
def xmlDiff = new Diff(writer.toString(),  
XmlExamples.CAR_RECORDS)  
assert xmlDiff.similar()
```

We have used [XMLUnit](#) to compare the XML we created with our sample XML. To do this, make sure the sample XML is available, i.e. that the following class is added to your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

You may also want to see [Using MarkupBuilder for Agile XML creation](#).

Creating XML with Groovy and XOM

Here is an example of using [XOM](#) to create a new XML file:

```
// require(groupId:'xmlunit',
artifactId:'xmlunit', version:'1.0')
// require(groupId:'xom', artifactId:'xom',
version:'1.1')
import
javax.xml.parsers.DocumentBuilderFactory
import org.custommonkey.xmlunit.*
import nu.xom.*

def addCar(root, name, make, year, country,
type, text) {
    def car = new Element('car')
    car.addAttribute(new Attribute('name',
name))
    car.addAttribute(new Attribute('make',
make))
    car.addAttribute(new Attribute('year',
year))
    root.appendChild(car)
    def countryNode = new Element('country')
    countryNode.appendChild(country)
    car.appendChild(countryNode)
    def record = new Element('record')
    record.appendChild(text)
    record.addAttribute(new
Attribute('type', type))
    car.appendChild(record)
}
```

```
def root      = new Element('records')
def document = new Document(root)

addCar(root, 'HSV Maloo', 'Holden', '2006',
'Australia',
      'speed', 'Production Pickup Truck
with speed of 271kph')
addCar(root, 'P50', 'Peel', '1962', 'Isle of
Man',
      'size', 'Smallest Street-Legal Car
at 99cm wide and 59 kg in weight')
addCar(root, 'Royale', 'Bugatti', '1931',
'France',
      'price', 'Most Valuable Car at $15
million')

// convert resulting document to a string
and compare with expected
XMLUnit.setIgnoreWhitespace(true)
```

```
def xmlDiff = new Diff(document.toXML(),  
XmlExamples.CAR_RECORDS)  
assert xmlDiff.similar()
```

We have used [XMLUnit](#) to compare the XML we created with our sample XML. To do this, make sure the sample XML is available, i.e. that the following class is added to your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

You may also want to see [Using MarkupBuilder for Agile XML creation](#).

Groovy's [XML capabilities](#) are powerful enough that you may not need to consider other alternatives such as [XQuery](#). If you wish to use XQuery though, it isn't hard to do. Here we just added mxquery.jar from the [MXQuery](#) project to our CLASSPATH.

Once our path is set up, we can run the following script:

```
// require(url:'http://www.mxquery.org',
jar:'mxquery.jar', version:'0.4.1')
import ch.ethz.mxquery.query.Context
import
ch.ethz.mxquery.query.impl.CompilerImpl
import static
ch.ethz.mxquery.util.IteratorPrinter.*

def query1 = 'for $seq in (1,2,3,4,5) where
$seq mod 2 eq 0 return $seq'
def context = new Context()
def compiler = new CompilerImpl()
def result = compiler.compile(context,
query1).evaluate()
println eventsToString(result)
// =>
// 0: [12815 xs:integer 2]
// 1: [12815 xs:integer 4]
// 2: [64 END_SEQUENCE]

def query2 = '''
    copy $x := <doc><el><node>this node is
deleted</node></el></doc>
    modify
    (
        delete node $x/el/node,
        insert node <node>this node is
```

```
inserted</node> into $x/el
    )
    return $x/el
...
result = compiler.compile(context,
query2).evaluate()
println
XMLPrettyPrint(eventsToXML(result).toString(
))
// =>
// <el>
//   <node>
```

```
//      this node is inserted  
//    </node>  
// </el>
```

Depending on your XQuery processor and operating system, you might be able to make use of XQuery directly from the command line instead of calling it from Java or Groovy.

### **Reading XML with Groovy and DOM4J**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using [DOM4J](#) with Groovy to process an existing XML file:

```

// require(groupId:'dom4j',
artifactId:'dom4j', version:'1.6.1')
import org.dom4j.io.SAXReader

def reader = new
StringReader(XmlExamples.CAR_RECORDS)
def records = new
SAXReader().read(reader).rootElement
def messages = []

records.elementIterator().each{ car ->
    def make = car.attributeValue('make')
    def country = car.elementText('country')
    def type =
car.element('record').attributeValue('type')
    messages << make + ' of ' + country + '
has a ' + type + ' record'
}

assert messages == [
    'Holden of Australia has a speed
record',
    'Peel of Isle of Man has a size record',
    'Bugatti of France has a price record'
]

```

DOM4J also supports a streaming mode which lets you manually prune parts of the DOM tree during processing to facilitate processing large documents. Here is an example which uses DOM4J in that mode:

```

// require(groupId:'dom4j',

```

```
artifactId:'dom4j', version:'1.6.1')
import org.dom4j.io.SAXReader
import org.dom4j.*

class PruningCarHandler implements
ElementHandler {
    def messages = []
    public void onStart(ElementPath path) {
    }
    public void onEnd(ElementPath path) {
        def car = path.current
        def make =
car.attributeValue('make')
        def country =
car.elementText('country')
        def type =
car.element('record').attributeValue('type')
        messages << make + ' of ' + country
+ ' has a ' + type + ' record'
        car.detach() // prune the tree
    }
}

def xml = new
StringReader(XmlExamples.CAR_RECORDS)
def reader = new SAXReader()
def handler = new PruningCarHandler()

reader.addHandler('/records/car', handler)
reader.read(xml)
```

```
assert handler.messages == [  
    'Holden of Australia has a speed  
record',  
    'Peel of Isle of Man has a size record',  
    'Bugatti of France has a price record'  
]
```

In the above example, we actually did the processing as part of the `ElementHandler`. Instead, we could have used a hybrid approach which just pruned away parts of the tree we weren't interested in and then performed tree-walking/navigation style coding after that.

### **Reading XML with Groovy and JDOM**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using [JDOM](#) with Groovy to process an existing XML file:

```

// require(groupId:'jdom',
artifactId:'jdom', version:'1.0')
import org.jdom.input.SAXBuilder

def reader    = new
StringReader(XmlExamples.CAR_RECORDS)
def records  = new
SAXBuilder().build(reader).rootElement
def messages = []

records.children.iterator().each{ car ->
    def make =
car.getAttribute('make').value
    def country =
car.getChildText('country')
    def type =
car.getChild('record').getAttribute('type').
value
    messages << make + ' of ' + country + '
has a ' + type + ' record'
}

assert messages == [
    'Holden of Australia has a speed
record',
    'Peel of Isle of Man has a size record',
    'Bugatti of France has a price record'
]

```

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Here is an example of using [XOM](#) with Groovy to process an existing XML file:

---

```
// require(groupId:'xom', artifactId:'xom',
version:'1.1')
import nu.xom.Builder

def reader = new
StringReader(XmlExamples.CAR_RECORDS)
def records = new
Builder().build(reader).rootElement
messages = []

def processCar(car) {
  def make =
car.getAttribute('make').value
  def country =
car.getFirstChildElement('country').value
  def type =
car.getFirstChildElement('record').getAttrib
ute('type').value
  messages << make + ' of ' + country + '
has a ' + type + ' record'
}

def cars = records.childElements
(0..<cars.size()).each{
processCar(cars.get(it)) }

assert messages == [
  'Holden of Australia has a speed
record',
```

```
'Peel of Isle of Man has a size record',  
'Bugatti of France has a price record'  
]
```

## Validating XML with a DTD

Many XML parsers have switches to turn on DTD validation.

Suppose we have the following XML:

```
def xml = '''  
<!DOCTYPE records [  
  <!ELEMENT car (country,record)>  
  <!ATTLIST car  
    make NMTOKEN #REQUIRED  
    name CDATA #REQUIRED  
    year NMTOKEN #REQUIRED  
  >  
  <!ELEMENT country (#PCDATA)>  
  <!ELEMENT record (#PCDATA)>  
  <!ATTLIST record type NMTOKEN #REQUIRED>  
  <!ELEMENT records (car+)>  
>  
<records>  
  <car name="HSV Maloo" make="Holden"  
year="2006">  
  <country>Australia</country>  
  <record type="speed">Production Pickup  
Truck with speed of 271kph</record>  
  </car>  
  <car name="P50" make="Peel" year="1962">  
  <country>Isle of Man</country>
```

```
<record type="size">Smallest Street-Legal  
Car at 99cm wide and 59 kg in  
weight</record>
```

```
</car>
```

```
<car name="Royale" make="Bugatti"  
year="1931">
```

```
<country>France</country>
```

```
<record type="price">Most Valuable Car at  
$15 million</record>
```

```
</car>
</records>
''.trim()
```

We can parse this document with validation turned on using the `XmlParser` as follows:

```
def validating = true          // default is
false
def namespaceAware = false    // default is
true
new XmlParser(validating,
namespaceAware).parseText(xml)
```

Or using the `XmlSlurper` as follows:

```
new XmlSlurper(validating,
namespaceAware).parseText(xml)
```

Or using the `DOMBuilder` as follows:

```
groovy.xml.DOMBuilder.parse(new
StringReader(xml), validating,
namespaceAware)
```

### Validating XML with a W3C XML Schema

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

We can validate that segment of XML against a Schema with the following code:



```

def xsd = '''
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="records">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded"
ref="car" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="car">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="country" />
        <xs:element ref="record" />
      </xs:sequence>
      <xs:attribute name="make"
use="required" type="xs:NCName" />
      <xs:attribute name="name"
use="required" />
      <xs:attribute name="year"
use="required" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <xs:element name="country"
type="xs:string" />
  <xs:element name="record">
    <xs:complexType mixed="true">
      <xs:attribute name="type"

```

```
use="required" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
</xs:schema>
''.trim()
```

```
import javax.xml.XMLConstants
import
javax.xml.transform.stream.StreamSource
import javax.xml.validation.SchemaFactory

def factory =
SchemaFactory.newInstance(XMLConstants.W3C_X
ML_SCHEMA_NS_URI)
def schema = factory.newSchema(new
StreamSource(new StringReader(xsd)))
```

```
def validator = schema.newValidator()  
validator.validate(new StreamSource(new  
StringReader(XmlExamples.CAR_RECORDS)))
```

### **Validating XML with RELAX NG**

This example assumes the following class is already on your CLASSPATH:

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

A number of [RELAX NG](#) validators are available (e.g. [MSV](#) and [Jing](#)). Rather than using the APIs for these validators directly, you might want to consider using the [ISO RELAX](#) project's common JARV API for accessing validators. Using this common API, you can switch between the [available validators](#) without changing your source

code.

An even better option (if you are using Java 5 or above, or otherwise have JAXP 1.3 available to you) is to use the [SORELAX JARV to JAXP 1.3 Xml Validation Engine Adaptor](#). This hooks into the built-in JAXP validation Factory support in JAXP and makes the code that you use to access the RELAX NG validator the same as you would do for a W3C XML Schema validator.

Here is what the code would look like:

```
//
require(url:'http://iso-relax.sourceforge.net/', jar:'isorelax.jar')
//
require(groupId:'org.iso_relax.verifier.jaxp.validation',
artifactId:'isorelax-jaxp-bridge',
version:'1.0')
// require(url:'https://msv.dev.java.net/',
jar:'msv.jar')

def rng = '''
<grammar
xmlns="http://relaxng.org/ns/structure/1.0"

datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <ref name="records"/>
  </start>
  <define name="car">
    <element name="car">
      <attribute name="make">
        <data type="token"/>
      </attribute>
```

```
        <attribute name="name">
            <text/>
        </attribute>
        <attribute name="year">
            <data type="integer"/>
        </attribute>
        <ref name="country"/>
        <ref name="record"/>
    </element>
</define>
<define name="country">
    <element name="country">
        <text/>
    </element>
</define>
<define name="record">
    <element name="record">
        <attribute name="type">
            <data type="token"/>
        </attribute>
        <text/>
    </element>
</define>
<define name="records">
    <element name="records">
        <oneOrMore>
            <ref name="car"/>
        </oneOrMore>
    </element>
</define>
</grammar>
```

```
'''.trim()
```

```
import javax.xml.XMLConstants
```

```
import
```

```
javax.xml.transform.stream.StreamSource
```

```
import javax.xml.validation.SchemaFactory
```

```
def factory =
```

```
SchemaFactory.newInstance(XMLConstants.RELAX  
NG_NS_URI)
```

```
def schema = factory.newSchema(new
```

```
StreamSource(new StringReader(rng)))
```

```
def validator = schema.newValidator()  
validator.validate(new StreamSource(new  
StringReader(XmlExamples.CAR_RECORDS)))
```

XML Example

## XmlExamples.groovy

```
class XmlExamples {
    static def CAR_RECORDS = '''
        <records>
            <car name='HSV Maloo' make='Holden'
year='2006'>
                <country>Australia</country>
                <record type='speed'>Production
Pickup Truck with speed of 271kph</record>
            </car>
            <car name='P50' make='Peel'
year='1962'>
                <country>Isle of Man</country>
                <record type='size'>Smallest
Street-Legal Car at 99cm wide and 59 kg in
weight</record>
            </car>
            <car name='Royale' make='Bugatti'
year='1931'>
                <country>France</country>
                <record type='price'>Most Valuable
Car at $15 million</record>
            </car>
        </records>
    '''
}
```

Process Management

Executing External Processes

Groovy provides a simple way to execute command line processes. Simply write the command line as a string and call the `execute()` method. E.g., on a \*nix machine (or a windows machine with appropriate \*nix commands installed), you can execute this:

```
def process = "ls -l".execute()
println "Found text ${process.text}"
```

The `execute()` method returns a `java.lang.Process` instance which will subsequently allow the in/out/err streams to be processed and the exit value from the process to be inspected etc.

e.g. here is the same command as above but we will now process the resulting stream a line at a time

```
def process = "ls -l".execute()
process.in.eachLine { line -> println line }
```

*in Groovy 1.5.5 the `eachLine` method is not available on `InputStream`. Earlier or later version will have that method. As an alternative you can use `process.in.newReader().eachLine { line -> println line }`*

Remember that many commands are shell built-ins and need special handling. So if you want a listing of files in a directory on a windows machine and if you write

```
Process p = "dir".execute()
println "${p.text}"
```

you will get `IOException` saying "Cannot run program "dir": CreateProcess error=2, The system cannot find the file specified."

This is because "dir" is built-in to the windows shell (`cmd.exe`) and can't be run as a simple executable. Instead, you will need to write:

```
Process p = "cmd /c dir".execute()
println "${p.text}"
```

Also, because this functionality currently make use of `java.lang.Process` under the covers, the deficiencies of that class must currently be taken into consideration. In particular, the javadoc for this class says:

*Because some native platforms only provide limited buffer size for standard input and output streams, failure to promptly write the input stream or read the output stream of the subprocess may cause the*

| *subprocess to block, and even deadlock*

Because of this, Groovy provides some additional helper methods which make stream handling for processes easier.

Here is how to gobble all of the output (including the error stream output) from your process:

```
def p = "rm -rf */*.tmp".execute()  
p.consumeProcessOutput()
```

There are also variations of `consumeProcessOutput` including one which takes output and error `StringBuffers`, another which takes output and error ditto for `InputStreams` and variations for `StringBuffers`, `Writers` and `InputStreams` which handle just the output stream or error stream individually. In addition, there is a `pipeTo` command (mapped to `|` to allow overloading) which lets the output stream of one process be fed into the input stream of another process.

Here are some examples of use:

```
proc1 = 'ls'.execute()  
proc2 = 'tr -d o'.execute()  
proc3 = 'tr -d e'.execute()  
proc4 = 'tr -d i'.execute()  
proc1 | proc2 | proc3 | proc4  
proc4.waitFor()  
if (proc4.exitValue())  
    print proc4.err.text  
else  
    print proc4.text
```

```
def sout = new StringBuffer()
def serr = new StringBuffer()
proc2 = 'tr -d o'.execute()
proc3 = 'tr -d e'.execute()
proc4 = 'tr -d i'.execute()
proc4.consumeProcessOutput(sout, serr)
proc2 | proc3 | proc4
[proc2, proc3].each{
it.consumeProcessErrorStream(serr) }
proc2.withWriter { writer ->
    writer << 'testfile.groovy'
}
proc4.waitForOrKill(1000)
println 'sout: ' + sout
println 'serr: ' + serr
```

```
def sout = new StringBuffer()
def serr = new StringBuffer()
proc1 = 'gzip -c'.execute()
proc2 = 'gunzip -c'.execute()
proc2.consumeProcessOutput(sout, serr)
proc1 | proc2
proc1.consumeProcessErrorStream(serr)
proc1.withWriter { writer ->
    writer << 'test text'
}
proc2.waitForOrKill(1000)
println 'sout: ' + sout // => test text
println 'serr: ' + serr
```

```
def initialSize = 4096
def outputStream = new
ByteArrayOutputStream(initialSize)
def errStream = new
ByteArrayOutputStream(initialSize)
def proc = "ls.exe".execute()
proc.consumeProcessOutput(outputStream,
errStream)
proc.waitFor()
println 'out:\n' + outputStream
println 'err:\n' + errStream
```

```
def out = new StringBuilder()
def err = new StringBuilder()
def proc = "ls".execute()
proc.waitForProcessOutput(out, err)
if (out) println "out:\n$out"
if (err) println "err:\n$err"
```

#### Further Information

See also:

- Cookbook Examples: [Executing External Processes From Groovy](#)
- PLEAC [Process Management](#) Examples
- A Groovy Shell called [Groosh](#) which builds upon Groovy's basic processing functionality

## Running Groovy on .NET 2.0 using IKVM

### Overview

It is possible to run Groovy on .NET 2.0 using IKVM as a virtual machine. By using IKVM compiler, you can compile your Groovy jar to be a .NET assembly (a DLL file) with the following command:

```
ikvmc -target:library groovy-all-1.5.0.jar
```

However, some warning will appear, but you can ignore them. After compilation you should have **groovy-all-1.5.0.dll** in the current directory.

### Using the Groovy .NET Assembly

With the compiled DLL, you can use it as a normal .NET assembly. The following example shows how to add it as a reference for building a simple application. You need the [SharpDevelop](#) IDE in this case.

- Open SharpDevelop, create a new .NET console application.
- Add references, your groovy DLL, and all IKVM DLLs.
- Open your main class, type the following code (in C# for example)

```
using System;
using System.Collections.Generic;

namespace GroovyDotNet {
    class MainClass {
        public static void Main(string[] args) {
            groovy.ui.InteractiveShell.main(args);
        }
    }
}
```

- Run the program, and you'll also have an executable file as well (in ./Debug directory).

the original post: <http://chanwit.blogspot.com/2007/04/groovy-running-fine-on-net.html>

### Invoking .NET classes

We can call .NET classes directly from Groovy scripts. Note that IKVM has the **cli** prefix for all .NET namespaces. If you want to invoke **Console.WriteLine('hello world')**, you have to type:

```
cli.System.Console.WriteLine('hello world')
```

The great thing here is that Groovy can pick a correct .NET method. You can also try:

```
cli.System.Console.WriteLine('hello world  
{0}', 'from Groovy')
```

in the interactive console (the binary in the attachment of this page).

### Encoding Issues

On some machines, there will be a test encoding issue to stop Groovy compiling script files. This seems to be the use of different encoding names between Java and .NET. If you find this kind of error, try specifying "-c" when testing with groovy.ui.GroovyMain class.

### Further development as a Groovy module?

Discuss [here](#).

## Things to remember

### Strings

- Strings are not Lists. In the JVM `java.lang.String` does not implement `java.util.List`.
- Arrays are not Lists. In the JVM arrays and `java.util.List` are quite different. In Groovy we support both as different types to ensure we interoperate cleanly with Java code, though we try wherever possible to make them interchangeable and appear polymorphic.

### Maps

- Maps override the dot operator, so `myMap.size` will return null unless you have a value for `map[size]`. Use `map.size()` instead.
- In map literals, all keys are interpreted as strings by default! If you want to use a variable or other literal as a key, use parentheses like so: `myMap = [(var1):val, (var2):val]`
- See the [Maps user guide](#)

## Things you can do but better leave undone

Groovy is a powerful tool. Like other powerful tools (think of a chainsaw) it requires a certain amount of user expertise and attention. Otherwise, results may be fatal.

Following code fragments are allowed in Groovy but usually result in unintended behaviour or incomprehensible problems. Simply don't do this and spare yourself some of the frustration that new and unskilled Groovy programmers sometimes experience.

### 1. Accessing an object's type like a property

Using `.class` instead of `.getClass()` is ok - as long as you know exactly what kind of object you have. But then you don't need that at all. Otherwise, you run in the risk of getting `null` or something else, but not the class of the object.

```
a = [:]
println a.class.simpleName //
NullPointerException, because a.class is
null.
```

### 2. Omitting parentheses around method arguments when not appropriate

Better use parentheses for argument lists unless you are sure.

For instance, a left parenthesis after a method name must always enclose the parameter list.

```
println a*(b+c)      // is OK
println (a+b)*c     // NullPointerException
exception, because println (a+b) results in
null.
```

### 3. Putting a newline at the wrong place into the code

This may cause the compiler to assume a complete statement although it is meant as continuing in the next line. It is not always as obvious as in the following example.

```
myVariable = "This is a very long statement
continuing in the next line. Result is="
    + 42                // this line has no
effect
```

### 4. Forgetting to write the second equals sign of the equals operator

As a result, a comparison expression turns into an assignment.

```
while (a=b) { ... } // will
be executed as long as b is true
(Groovy-truth)
String s = "A"; boolean b = s = "B" // b
becomes true
```

(One measure is to put constant expressions in comparisons always before the equals operator.)

### 5. Specifying the wrong return type when overriding a method (Groovy version < 1.5.1 only)

Such a method may be considered as overloaded (a different method with the same name) and not be called as expected.

```
def toString() { ... } // does not override  
toString() inherited from Object
```

## 6. Disregarding other objects' privacy

When accessing methods, fields, or properties of other classes, make sure that you do not interfere with private or protected members. Currently Groovy doesn't distinguish properly between public, private, and protected members, so watch out yourself.

```
z = [1,2,3]  
z.size = 10 // size is private; errors when  
trying to access z[3] to z[9], although  
z.size()==10  
  
'more'.value = 'less' // results in  
"more"=="less"
```

## 7. Thoughtless dynamic programming

Check for conflicts with intended class functionality before adding or changing methods or properties using Groovy's dynamic facilities.

```
String.metaClass.class = Integer // results  
in 'abc'.getClass()==java.lang.Integer
```

## 8. String concatenation

As in Java, you can concatenate Strings with the "+" symbol. But Java only needs that one of the two items of a "+" expression to be a String, no matter if it's in the first place or in the last one. Java will use the `toString()` method in the non-String object of your "+" expression. But in Groovy, you just should be safe the first item of your "+" expression implements the `plus()` method in the right way, because Groovy will search and use it. In Groovy GDK, only the `Number` and `String/StringBuffer/Character` classes have the `plus()` method implemented to concatenate strings. To avoid surprises, always use `GStrings`.

```
// Java code, it works
boolean v = true;
System.out.println(" foo "+v);
System.out.println(v+" foo ");
```

```
// Groovy code
boolean v = true
println " foo "+v    // It works
println v+" foo "    // It fails with
MissingMethodException: No signature of
method: java.lang.Boolean.plus()
```

## Using Spring Factories with Groovy

[Spring](#) is an open-source framework created to address the complexity of Java enterprise application development. One of Spring's goals is to help developers write simple, testable and loosely coupled systems while reducing the amount of scaffolding code required. In this respect, Groovy has a common goal. So, for simple systems, Groovy alone may be sufficient for your needs. However, as your system grows in size and complexity, and especially in hybrid Java/Groovy environments, you might find Spring's facilities provide great value to your Groovy system development.

Here we look at using Spring's Bean Factory mechanisms within Groovy. These facilities allow beans to be managed within a Spring container. The beans are normally Java objects, but since Groovy objects are Java objects, Spring can just as easily manage Groovy objects for you. In particular, in mixed Java/Groovy environments, you can leverage any existing domain objects or services already have defined in your Spring wiring from the Java part of your application. You can immediately start using those within your Groovy scripts and code, allowing a great mix of strongly-typed Java domain objects and dynamically typed Groovy code.

Let's start by exploring a simple calculator application.

### ***Bare Bones Approach***

Suppose we have the following implementation class:

```
class CalcImpl {  
    def doAdd(x, y) { x + y }  
}
```

We can make use of that class from a script as follows:

```
import  
org.springframework.beans.factory.support.*  
  
def configure(factory) {  
    def bd = new  
    RootBeanDefinition(CalcImpl)  
  
    factory.registerBeanDefinition('calcBean',  
    bd)  
}  
  
def factory = new  
DefaultListableBeanFactory()  
configure(factory)  
  
def calc = factory.getBean('calcBean')  
println calc.doAdd(3, 4) // => 7
```

This script relies on no external wiring files. Everything is configured in the script itself. If we wish to alter our system at a later time, we simply alter the configuration inside the `configure()` method. In the Java world, this wouldn't be very flexible, but in the Groovy world, this script may be executed from source code (even dynamically loaded when it changes) so altering the configuration doesn't necessarily require a new build.

### ***Classical Spring Approach***

Probably the most common way to use Spring is to use an XML *wiring* file. As systems grow larger, wiring files allow configuration to be centralised in easy to change 'groupings' of beans. Let's assume our calculator needs to

eventually be expanded to have additional functionality. An approach to handling complexity as the system grows is to delegate functionality off to other components. Here is how we might code up an *adder* component:

```
class AdderImpl {  
    def add(x, y) { x + y }  
}
```

Then, applying the delegate design pattern we mentioned earlier would result in the following refactored calculator:

```
class CalcImpl2 {  
    def AdderImpl adder  
    def doAdd(x, y) { adder.add(x, y) }  
}
```

To capture our software system configuration, we will use an XML wiring file (we have two beans):

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema
/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://www.springframewo
rk.org/schema/beans

http://www.springframework.org/schema/beans/
spring-beans.xsd">
    <bean id="calcBean" class="CalcImpl2"
autowire="byType"/>
    <bean class="AdderImpl"/>
</beans>
```

Now, our script code looks like:

```
import
org.springframework.context.support.ClassPat
hXmlApplicationContext

def ctx = new
ClassPathXmlApplicationContext('calcbeans.xml')
def calc = ctx.getBean('calcBean')
println calc.doAdd(3, 4) // => 7
```

### ***Annotation Approach***

If we wish, we can remove the need for an XML file by using annotations. Note that we need to think carefully before using this approach extensively in a large system for two reasons. Firstly, we are more tightly coupling our system to Spring. Secondly, we are associating configuration and deployment information with our source code. Perhaps separating those concerns will have enormous benefits for large systems. The good news is that Spring lets you take on board just those annotations that you are happy to use.

Here is how we might code up our adder component:

```
import
org.springframework.stereotype.Component

@Component class AdderImpl {
    def add(x, y) { x + y }
}
```

Here is our modified calculator:

```
import
org.springframework.beans.factory.annotation
.Autowired
import
org.springframework.stereotype.Component

@Component class CalcImpl3 {
    @Autowired private AdderImpl adder
    def doAdd(x, y) { adder.add(x, y) }
}
```

And here is our script code (note no XML file is required):

```
import
org.springframework.context.support.GenericA
pplicationContext
import
org.springframework.context.annotation.Class
PathBeanDefinitionScanner

def ctx = new GenericApplicationContext()
new
ClassPathBeanDefinitionScanner(ctx).scan('')
// scan root package for components
ctx.refresh()
def calc = ctx.getBean('calcImpl3')
println calc.doAdd(3, 4) // => 7
```

This example uses features in Spring 2.1 (currently at a Milestone release) and Groovy 1.1 (currently in beta release) on a Java 5 or greater JVM.

#### ***BeanBuilder Approach***

We can also use the [BeanBuilder](#) from [Grails](#) to avoid writing an XML file. It will look something like this (after adding the latest Grails jar to your classpath):

```
def bb = new grails.spring.BeanBuilder()
bb.beans {
    adder(AdderImpl)
    calcBean(CalcImpl2) { delegate.adder =
adder } // need to use delegate may change
}
def ctx = bb.createApplicationContext()
def calc = ctx.getBean('calcBean')
println calc.doAdd(3, 4) // => 7
```

#### Further Information

- The Spring Example in [Groovy and JMX](#)
- Spring [Documentation](#)

## WebStart

The current released Groovy Console can be run via [this webstart file](#).

Note, however, that the generated script files will be run under the WebStart security sandbox. So currently this is very limited in it's general usability. Sufficient to hack with however.

## OSGi and Groovy

The [OSGi](#) framework is a powerful Java tool providing component based service and dependency management. OSGi components can be remotely installed, started, stopped, updated and uninstalled without shutting down your application. Also, OSGi provides far greater dependency management than the basic Java classpath mechanism, allowing you to specify specific versions of dependencies and keep dependencies private so that other components cannot load them. One of the most notable usages of OSGi is the Eclipse Plugin Container. A good starting point for more information is the [Wikipedia page](#) and the Further Reading section of this document.

### Loading Groovy as an OSGi service

The Groovy jar files are released with correct OSGi metadata, so they can be loaded into any OSGi compliant container, such as Eclipse Equinox or Apache Felix. The metadata can be viewed by looking at the jar file's MANIFEST.MF file. For instance, your Jar manifest might contain these lines:

```
Bundle-Version: 1.7.0  
Export-Package:  
groovy.text;version="1.7.0.beta-1-SNAPSHOT",  
groovy.xml;  
  
version="1.7.0.beta-1-SNAPSHOT",groovy.util;  
version="1.7.0.beta-1-SNA  
  
PSHOT",groovy.lang;version="1.7.0.beta-1-SNA  
PSHOT",groovyjarjarcom...
```

This declares to the OSGi container that the Jar is version 1.7.0 and provides the specified packages for import by other components.

The following examples all use the Eclipse Equinox container, which can be downloaded from the Internet or found in an Eclipse installation.

Perform the following steps to install and start the Groovy Jar in an OSGi container:

Start the OSGi container in console mode:

```
java -jar  
org.eclipse.osgi_3.4.0.v20080605-1900.jar  
-console
```

This should bring up an OSGi console prompt:

```
osgi>
```

You can see the system status of the container at any time using the "ss" command.

```
osgi> ss
```

```
Framework is launched.
```

<b>id</b>	<b>State</b>	<b>Bundle</b>
<b>0</b>	<b>ACTIVE</b>	<b>org.eclipse.osgi_3.4.0.v20080605-1900</b>

Install the Groovy jar file using "install" and a file URL to your groovy-all jar:

```
osgi>install  
file:///home/user/dev/groovy-core/target/dist/groovy-all-1.7-beta-1-SNAPSHOT.jar  
Bundle id is 10
```

The container will assign the bundle an identifier. Start the bundle using the "start" command:

```
osgi> start 10
```

Verify the bundle is started using "ss":

```
osgi> ss
```

```
Framework is launched.
```

id	State	Bundle
0	ACTIVE	org.eclipse.osgi_3.4.0.v20080605-1900
10	ACTIVE	groovy-all_1.7.0.beta-1-SNAPSHOT

You can list all the packages the Groovy bundle provides with the "packages" command:

```
osgi>packages 10
groovy.xml.streamingmarkupsupport;
version="1.7.0.beta-1-SNAPSHOT"<file:///home
/user/dev/groovy-core/target/dist/groovy-al
l-1.7-beta-1-SNAPSHOT.jar [10]>gro
ovyjarjarantlr.actions.java;
version="1.7.0.beta-1-SNAPSHOT"<file:///home
/user
...
```

### Writing a Groovy OSGi Service

Once the Groovy jar is loaded into the container, writing an OSGi service that uses Groovy is as simple as creating a class that extends the framework's BundleActivator interface.

```
package org.codehaus.groovy.osgi

import org.osgi.framework.BundleActivator
import org.osgi.framework.BundleContext

class Activator implements BundleActivator {

    void start(BundleContext context) {
        println "Groovy BundleActivator
started"
    }

    void stop(BundleContext context) {
        println "Groovy BundleActivator
stopped"
    }
}
```

The Activator's start(BundleContext) method will be invoked when the container starts the service, and the stop(BundleContext) method will be invoked when the container stops the service.

The first step in deploying the new Groovy service is to create a jar file containing the Activator. The manifest for the Jar needs to specify the name of the new service, the version, the fully qualified path to the Activator, and which packages from the groovy-all jar bundle to import. The complete manifest for this example follows:

```
Manifest-Version: 1.0  
Ant-Version: Apache Ant 1.7.0  
Created-By: 10.0-b19 (Sun Microsystems Inc.)  
Built-By: user  
provider: org.codehaus.groovy.osgi  
Bundle-ManifestVersion: 2  
Bundle-Name: Groovy OSGi Example Bundle  
Bundle-SymbolicName:  
org.codehaus.groovy.osgi.hello-groovy-bundle  
Bundle-Version: 1.0.0  
Bundle-Activator:  
org.codehaus.groovy.osgi.Activator  
Bundle-Vendor: Groovy  
Bundle-Localization: plugin  
Import-Package:  
groovy.lang;version="1.7.0.beta-1-SNAPSHOT",  
org.codeha  
us.groovy.reflection;version="1.7.0.beta-1-S  
NAPSHOT",org.codehaus.gro  
ovy.runtime;version="1.7.0.beta-1-SNAPSHOT",  
org.codehaus.groovy.runti  
me.callsite;version="1.7.0.beta-1-SNAPSHOT",  
org.w3c.dom,org.osgi.frame  
work;version="1.3.0"  
Bundle-ClassPath: .
```

The Import-Package statement is important. It states all the dependencies from the Groovy-all jar which are allowed to be referenced. The Groovy-all Jar exports many, many more packages than just this... an Import-Package definition with just enough dependencies to get the println to work correctly is shown here. In a more meaningful Activator you'd want to import many more of the packages.

The complete Jar for this example has a layout as follows:

```
hello-bundle-imports-groovy.jar
--META-INF
----MANIFEST.MF
--org
----codehaus
-----groovy
-----osgi
-----Activator.class
```

Test the new Hello-Groovy bundle by running the OSGi console and issuing the following commands, using "ss" to verify that the correct dependencies are loaded beforehand:

```
osgi> ss
```

```
Framework is launched.
```

id	State	Bundle
0	ACTIVE	org.eclipse.osgi_3.4.0.v20080605-1900
10	ACTIVE	groovy-all_1.7.0.beta-1-SNAPSHOT

```
osgi> install
```

```
file:///home/user/dev/groovy-core/src/examples/osgi/build/hello-bundle-imports-groovy.jar
```

```
Bundle id is 12
```

```
osgi> ss
```

**Framework is launched.**

<b>id</b>	<b>State</b>	<b>Bundle</b>
<b>0</b>	<b>ACTIVE</b>	<b>org.eclipse.osgi_3.4.0.v20080605-1900</b>
<b>10</b>	<b>ACTIVE</b>	<b>groovy-all_1.7.0.beta-1-SNAPSHOT</b>
<b>12</b>	<b>INSTALLED</b>	<b>org.codehaus.groovy.osgi.hello-groovy-bundle_1.0.0</b>

**osgi> start 12**

**Groovy BundleActivator started**

```
osgi> stop 12  
Groovy BundleActivator stopped
```

The start and stop message shows that the Groovy service was correctly started and stopped.

### Including the Groovy Jar within a Bundle

The previous example shows how to resolve an Activator's Groovy dependency from the container. The Activator can only be started after the Groovy bundle is started. An alternative is to simply include the groovy-all jar within your bundle. This eliminates the need to declare Import-Packages, but does bloat the size of the jar. Any Jar file included within your bundle has private visibility and cannot be referenced by any other bundles that happen to be running in the container.

To include the groovy-all jar within your bundle, rather than loading it from the container, create your Jar with a layout as follows:

```
hello-bundle-contains-groovy.jar  
--groovy-all-1.7-beta-1-SNAPSHOT.jar  
--META-INF  
----MANIFEST.MF  
--org  
----codehaus  
-----groovy  
-----osgi  
-----Activator.class
```

And use this template as the MANIFEST.MF:

```
Manifest-Version: 1.0  
Ant-Version: Apache Ant 1.7.0  
Created-By: 10.0-b19 (Sun Microsystems Inc.)  
Built-By: user  
provider: org.codehaus.groovy.osgi  
Bundle-ManifestVersion: 2  
Bundle-Name: Groovy OSGi Example Bundle  
Bundle-SymbolicName:  
org.codehaus.groovy.osgi.hello-groovy-bundle  
Bundle-Version: 1.0.0  
Bundle-Activator:  
org.codehaus.groovy.osgi.Activator  
Bundle-Vendor: Groovy  
Bundle-Localization: plugin  
Import-Package:  
org.w3c.dom,org.osgi.framework;version="1.3.  
0"  
Bundle-ClassPath:  
.,groovy-all-1.7-beta-1-SNAPSHOT.jar
```

The Jar can now be loaded and started in the container without first loading the Groovy Jar. Verify this by running the following within the console:

```
osgi> install
file:///home/user/dev/groovy-core/src/examples/osgi/build/hello-bundle-contains-groovy.jar
Bundle id is 14
osgi> ss

Framework is launched.
id          State          Bundle
0           ACTIVE
org.eclipse.osgi_3.4.0.v20080605-1900
14          INSTALLED
org.codehaus.groovy.osgi.hello-groovy-bundle_1.0.0

osgi> start 14
Groovy BundleActivator started
```

## Publishing a Service Written in Groovy

Publishing a service written in Groovy requires one extra step that a Java service does not. This is because of the way Groovy makes extensive use of ClassLoaders and reflection. When registering a service with the BundleContext, you must be sure to temporarily set the current thread's ContextClassLoader to the target object's ClassLoader, and then set it back when you're done. It's actually quite easy, and this example walks you through this one detail.

In order to demonstrate registering a service, we need to create a sample service in Groovy. This is just a POGO interface and implementation. Consider the GroovyGreeter which simply prints a message to the console.

GroovyGreeter.groovy defines the interface:

```
package org.codehaus.groovy.osgi

interface GroovyGreeter {
    void sayHello()
}
```

And GroovyGreeterImpl.groovy defines the implementation:

```
package org.codehaus.groovy.osgi

class GroovyGreeterImpl implements
GroovyGreeter {
    void sayHello() {
        println "Hello from the Groovy
Greeter!"
    }
}
```

Now the Activator can create an instance of GroovyGreeterImpl and register it with the container as a GroovyGreeter provider. In Java, you'd need one line to call `BundleContext.registerService(String, Object, Dictionary)`, but in Groovy we need to change the `ContextClassLoader` while we do this. Here is a complete and correct Activator:

```

package org.codehaus.groovy.osgi
import org.osgi.framework.BundleActivator
import org.osgi.framework.BundleContext
import
org.osgi.framework.ServiceRegistration

class Activator implements BundleActivator {
    ServiceRegistration registration
    void start(BundleContext context) {
        ClassLoader originalClassLoader =
Thread.currentThread().contextClassLoader
        try {

Thread.currentThread().contextClassLoader =
getClass().classLoader
            GroovyGreeter myService = new
GroovyGreeterImpl()
            registration =
context.registerService(GroovyGreeter.class.
getName(), myService, null)
        }
        finally {

Thread.currentThread().contextClassLoader =
originalClassLoader
        }
    }
    void stop(BundleContext context) {
        registration.unregister()
    }
}

```

The Jar file for this bundle is similar to the first example's:

```
hello-bundle-imports-groovy.jar  
--META-INF  
----MANIFEST.MF  
--org  
----codehaus  
-----groovy  
-----osgi  
-----Activator.class  
-----GroovyGreeter.class  
-----GroovyGreeterImpl.class
```

The Jar Manifest is almost the same as the first example. The change is the Export-Package statement. Since we are registering the a service `org.codehaus.groovy.osgi.GroovyGreeter`, we need to specify that package and version in an Export-Package statement:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 10.0-b19 (Sun Microsystems Inc.)
Built-By: user
provider: org.codehaus.groovy.osgi
Bundle-ManifestVersion: 2
Bundle-Name: Groovy OSGi Example Bundle
Bundle-SymbolicName:
org.codehaus.groovy.osgi.hello-groovy-bundle
Bundle-Version: 1.0.0
Bundle-Activator:
org.codehaus.groovy.osgi.Activator
Bundle-Vendor: Groovy
Bundle-Localization: plugin
Import-Package:
groovy.lang;version="1.7.0.beta-1-SNAPSHOT",
org.codeha
us.groovy.reflection;version="1.7.0.beta-1-S
NAPSHOT",org.codehaus.gro
ovy.runtime;version="1.7.0.beta-1-SNAPSHOT",
org.codehaus.groovy.runti
me.callsite;version="1.7.0.beta-1-SNAPSHOT",
org.w3c.dom,org.osgi.fram
ework;version="1.3.0"
Export-Package:
org.codehaus.groovy.osgi;version="1.0.0"
Bundle-ClassPath: .
```

Notice, also, how this bundle is importing the Groovy Jar from the container, it does not contain the Groovy Jar within itself. To verify and test this, use the OSGi console to install and start the groovy-all Jar and then install and start this Jar.

## Consuming a Service from Groovy

It is easy to consume an OSGi service from a Groovy Activator. This example shows how to locate and invoke the service from the previous section, but could just as easily work with a different service. It does not matter at all whether the service was written in Groovy, Java, or any other language. The implementation details of a service should be completely hidden from you by the OSGi module system. Also, while this example is written in Groovy, it is not very different from how it would look in Java.

To consume the service from the previous section, you will need an Activator that retrieves the service from the BundleContext in the start(BundleContext) method:

```
package org.codehaus.groovy.osgi.harness

import org.osgi.framework.BundleActivator
import org.osgi.framework.BundleContext
import
org.osgi.framework.ServiceRegistration
import org.osgi.framework.ServiceReference
import
org.codehaus.groovy.osgi.GroovyGreeter

class HarnessActivator implements
BundleActivator {

    void start(BundleContext context) {
        String serviceName =
GroovyGreeter.class.name
        ServiceReference[] references =
context.getAllServiceReferences(serviceName,
null)

        println "${ references ?
references.size() : 0 } GroovyGreeter
services found."
    }
}
```

```
        references?.each { ServiceReference
ref ->
            Object serviceHandle =
context.getService(ref)
            GroovyGreeter service =
serviceHandle
            service.sayHello()
        }
    }
```

```
    void stop(BundleContext context) {  
        }  
    }  
}
```

The service was registered by the interface name, so this Activator queries for that interface, printing out all the providers found. Notice the package and name of this Activator changed. Since the previous example exported the `org.codehaus.groovy.osgi`, this example needed to pick a different package to avoid conflicts. Also, the variable types in the code sample were explicitly declared to make the sample easier to read.

To package and run this example you'll need to make a Jar with the following layout:

```
hello-groovy-test-harness.jar  
--META-INF  
----MANIFEST.MF  
--org  
----codehaus  
-----groovy  
-----osgi  
-----harness  
-----HarnessActivator.class  
-----HarnessActivator$_start_closure1  
.class
```

And the manifest needs to import the `org.codehaus.groovy.osgi` in the previous example:

**Manifest-Version: 1.0**  
**Ant-Version: Apache Ant 1.7.0**  
**Created-By: 10.0-b19 (Sun Microsystems Inc.)**  
**Built-By: user**  
**provider: org.codehaus.groovy.osgi.harness**  
**Bundle-ManifestVersion: 2**  
**Bundle-Name: Groovy OSGi Test Harness**  
**Bundle-SymbolicName:**  
**org.codehaus.groovy.osgi.harness.hello-groovy-tes**  
**t-harness**  
**Bundle-Version: 1.0.0**  
**Bundle-Activator:**  
**org.codehaus.groovy.osgi.harness.HarnessActivator**  
**Bundle-Vendor: Groovy**  
**Bundle-Localization: plugin**  
**Import-Package:**  
**org.codehaus.groovy.runtime.typehandling;version="1.0.**  
**0",org.codehaus.groovy.osgi;version="1.0.0"**  
**,groovy.lang;version="1.7.**  
**0.beta-1-SNAPSHOT",org.codehaus.groovy.reflection;version="1.7.0.beta**  
**-1-SNAPSHOT",org.codehaus.groovy.runtime;version="1.7.0.beta-1-SNAPSH**  
**OT",org.codehaus.groovy.runtime.callsite;version="1.7.0.beta-1-SNAPSH**  
**OT",org.w3c.dom,org.osgi.framework;version="1.3.0"**  
**Bundle-ClassPath: .**

Install and test this bundle in the OSGi console. To install this bundle you'll need the groovy-all bundle installed first:

```
osgi> install
file:///home/user/dev/groovy-core/src/examples/osgi/../../../../target/dist/groovy-all-1.7-beta-1-SNAPSHOT.jar
Bundle id is 6
```

```
osgi> install
file:///home/user/dev/groovy-core/src/examples/osgi/build/hello-groovy-test-harness.jar
Bundle id is 7
```

```
osgi> ss
```

**Framework is launched.**

<b>id</b>	<b>State</b>	<b>Bundle</b>
0	ACTIVE	org.eclipse.osgi_3.4.0.v20080605-1900
6	INSTALLED	groovy-all_1.7.0.beta-1-SNAPSHOT
7	INSTALLED	org.codehaus.groovy.osgi.harness.hello-groovy-test-harness_1.0.0

To start the bundle, you'll need the groovy-all bundle started and the hello-groovy bundle installed:

```
osgi> install
file:///home/user/dev/groovy-core/src/examples/osgi/build/hello-bundle-imports-groovy.jar
Bundle id is 8

osgi> start 6

osgi> start 7
0 GroovyGreeter services found.
```

To see the GroovyGreeter service locate and invoke the service, start the hello-groovy bundle and the restart the harness:

```
osgi> ss
```

```
Framework is launched.
```

id	State	Bundle
0	ACTIVE	org.eclipse.osgi_3.4.0.v20080605-1900
6	ACTIVE	groovy-all_1.7.0.beta-1-SNAPSHOT
7	ACTIVE	org.codehaus.groovy.osgi.harness.hello-groovy-test-harness_1.0.0
8	RESOLVED	org.codehaus.groovy.osgi.hello-groovy-bundle_1.0.0

```
osgi> start 8
```

```
Groovy BundleActivator started
```

```
osgi> stop 7
```

```
osgi> start 7
```

```
1 GroovyGreeter services found.
```

```
Hello from the Groovy Greeter!
```

Exploring the OSGi console can be a valuable learning experience. Now that all the bundles are loaded, try playing with the "bundle", "headers", "services", and "packages" commands. Help is available by typing "help".

### Common Errors

Diagnosing runtime exceptions can be a frustrating experience for the OSGi beginner and expert alike. Here are some common errors and solutions you might experience running these code samples:

**ClassNotFoundException** - The Java and Groovy compiler use a classpath to resolve dependencies, while the OSGi container does not. This means that code that compiles fine might still receive ClassNotFoundException exceptions when running in the container. Here are some steps to diagnose the issue:

- Find out which class is missing but reading the exception stack trace
- Find out which Jar contains the missing class
- Does that Jar's Manifest contain an Export-Package statement for the class' package? If not, then the Jar was built incorrectly and you need to add the package to the Export-Package statement of the Jar.
- Does your Jar's Manifest contain an Import-Package statement for the imported package? If not, then your Jar was built incorrectly and you need to add the package to the Import-Package statement of your Jar.
- Does the Export-Package version number match your Import-Package version number? If not, then update your manifest to import the correct version, or update the missing class' jar to export the correct version?
- Is the missing class' bundle correctly installed and started in the container? The results of an "ss" command must show the Jar as "Active" or else it won't be resolved. Use the "install" and "start" commands to start the jar correctly.

**Missing Constraint** - This means that one of your declared Import-Package statements cannot be satisfied by the container. The error message will state exactly which dependency is missing, for instance it might say: Missing Constraint groovy.lang; version="1.7.0.beta-1-SNAPSHOT".

- Is the missing constraint's bundle correctly installed and started in the container? The results of an "ss" command must show the Jar as "Active" or else it won't be resolved. Use the "install" and "start" commands to start the jar correctly.

**ClassCastException** - This error occurs when retrieving services out of the BundleContext and takes the form of the mysterious error message "Cannot cast foo.bar.Class to foo.bar.Class". This means that the ClassLoader of your bundle has a different version of foo.bar.Class than the one you're retrieving.

- Look at how the service's Activator is adding foo.bar.Class to the BundleContext. If the class is implemented in Groovy then you must add the service using the ClassLoader code in the Publishing a Service Written in Groovy of this document.
- Look at how your Activator is resolving the reference to the class. If your bundle already defines foo.bar.Class and you're trying to retrieve a foo.bar.Class from a different bundle, then the versions of the classes won't match. Declare the type as an interface that is imported the same way between both bundles to resolve this issue.

## Further Reading

The great part about using Groovy for OSGi is that the existing tutorials for Java and OSGi are all easily converted to Groovy.

- JavaWorld.com has a good three part introduction to Java and OSGi. [Part 1](#) details the basics as a Hello World service. Part 2 describes the Spring DM product and adding Spring to OSGi. And Part 3 covers several web deployment scenarios and issues. [Part 2](#) describes the Spring DM product and adding Spring to OSGi. And [Part 3](#) covers several web deployment scenarios and issues.
- TheServerSide.com also has an [OSGi for Beginners](#) article.
- Hamlet D'Arcy has a [Groovy and OSGi on the Desktop](#) tutorial on his [blog](#).
- Groovy and Sling (using OSGi): [Apache Sling doco](#), [blog](#)

## Advanced Usage Guide

This guide provides information that you don't need when first starting to learn Groovy but can come in handy when you want to push the boundaries of the language or improve your Groovy style.

- [Advanced compiler configuration](#)
- [Ant Task Troubleshooting](#)

- [Make a builder](#)
- [BuilderSupport](#)
- [Compiling Groovy](#)
  - [Compiling With Maven2](#)
- [Creating an extension module](#)
- [Design Patterns with Groovy](#)
  - [Abstract Factory Pattern](#)
  - [Adapter Pattern](#)
  - [Bouncer Pattern](#)
  - [Chain of Responsibility Pattern](#)
  - [Composite Pattern](#)
  - [Decorator Pattern](#)
  - [Delegation Pattern](#)
  - [Flyweight Pattern](#)
  - [Iterator Pattern](#)
  - [Loan my Resource Pattern](#)
  - [Null Object Pattern](#)
  - [Pimp my Library Pattern](#)
  - [Proxy Pattern](#)
  - [Singleton Pattern](#)
  - [State Pattern](#)
  - [Strategy Pattern](#)
  - [Template Method Pattern](#)
  - [Visitor Pattern](#)
- [Dynamic language beans in Spring](#)
- [Embedding Groovy](#)
- [Mixed Java and Groovy Applications](#)
- [Influencing class loading at runtime](#)
- [Internationalization](#)
- [Meta-annotations](#)
- [Optimising Groovy bytecodes with Soot](#)
- [Polyglot Programming with Groovy](#)
  - [Calling Clojure from Groovy](#)
  - [Calling Scala from Groovy](#)
  - [Constraint Programming](#)
  - [Functional Programming with Groovy](#)
  - [JSR-223 access to other JVM languages](#)
- [Refactoring with Groovy](#)
  - [Introduce Assertion](#)
  - [Replace Inheritance with Delegation](#)
- [Security](#)
- [Writing Domain-Specific Languages](#)
- [Type checking extensions](#)

## Advanced compiler configuration

### Tweaking the compiler configuration

Whether you are using *groovyc* to compile classes or a *GroovyShell*, for example, to execute scripts, under the hood, a [compiler configuration](#) is used. This configuration holds information like the source encoding or the classpath but it can also be used to perform more operations like adding imports by default, applying AST transformations transparently or disabling global AST transformations.

## Compilation customizers

### Introduction

Before Groovy 1.8.0, doing tasks like adding imports transparently (for DSLs) was a bit complicated. It involved writing a custom `GroovyClassLoader` and lots of trickery. The goal of compilation customizers is to make those common tasks easy to implement. For that, the `CompilerConfiguration` class is the entry point. The general schema will always be based on the following code:

```
import  
org.codehaus.groovy.control.CompilerConfigur  
ation  
// create a configuration  
def config = new CompilerConfiguration()  
// tweak the configuration  
config.addCompilationCustomizers(...)  
// run your script  
def shell = new GroovyShell(config)  
shell.evaluate(script)
```

Compilation customizers must extend the `org.codehaus.groovy.control.customizers.CompilationCustomizer` class. A customizer works:

- on a specific compilation phase
- on *every* class node being compiled

You can implement your own compilation customizer but Groovy includes some of the most common operations.

### The import customizer

Using this compilation customizer, your code will have imports added transparently. This is in particular useful for scripts implementing a DSL where you want to avoid users from having to write imports. The import customizer will let you add all the variants of imports the Groovy language allows, that is:

- class imports, optionally aliased
- star imports
- static imports, optionally aliased
- static star imports

```

import
org.codehaus.groovy.control.customizers.ImportCustomizer
def icz = new ImportCustomizer()
// "normal" import
icz.addImports('java.util.concurrent.atomic.AtomicInteger',
'java.util.concurrent.ConcurrentHashMap')
// "aliases" import
icz.addImport('CHM',
'java.util.concurrent.ConcurrentHashMap')
// "static" import
icz.addStaticImport('java.lang.Math', 'PI')
// import static java.lang.Math.PI
// "aliased static" import
ica.addStaticImport('pi', 'java.lang.Math',
'PI') // import static java.lang.Math.PI as
pi
// "star" import
icz.addStarImport 'java.util.concurrent' //
import java.util.concurrent.*
// "static star" import
icz.addStaticStar 'java.lang.Math' // import
static java.lang.Math.*

```

### The AST transformation customizer

The AST transformation customizer is meant to apply AST transformations transparently. Unlike global AST transformations that apply on every class being compiled as long as the transform is found on classpath (which has drawbacks like increasing the compilation time or side effects due to transformations applied where they should not), the customizer will allow you to selectively apply a transform only for specific scripts or classes.

As an example, let's say you want to be able to use `@Log` in a script. The problem is that `@Log` is normally applied

on a class node and a script, by definition, doesn't require one. But implementation wise, scripts are classes, it's just that you cannot annotate this implicit class node with `@Log`. Using the AST customizer, you have a workaround to do it:

```
import  
org.codehaus.groovy.control.customizers.AST  
transformationCustomizer  
import groovy.util.logging.Log  
def acz = new  
ASTTransformationCustomizer(Log)  
configuration.addCompilationCustomizers(acz)
```

That's all! Internally, the `@Log` AST transformation is applied to every class node in the compilation unit. This means that it will be applied to the script, but also to classes defined within the script.

If the AST transformation that you are using accepts parameters, you can use parameters in the constructor too:

```
def acz = new  
ASTTransformationCustomizer(Log, value:  
'LOGGER') // use name 'LOGGER' instead of  
the default 'log'
```

As the AST transformation customizers works with objects instead of AST nodes, not all values can be converted to AST transformation parameters. For example, primitive types are converted to `ConstantExpression` (that is 'LOGGER' is converted to `new ConstantExpression('LOGGER')`), but if your AST transformation takes a closure as an argument, then you have to give it a `ClosureExpression`, like in the following example:

```

final expression = new
AstBuilder().buildFromCode(CompilePhase.CONV
ERSION) {-> true }.expression[0]
customizer = new
ASTTransformationCustomizer(ConditionalInter
rupt, value:expression,
thrown:SecurityException)
configuration.addCompilationCustomizers(cust
omizer)
def shell = new GroovyShell(configuration)
shouldFail(SecurityException) {
    shell.evaluate("""
                // equivalent to adding
@ConditionalInterrupt(value={true}, thrown:
SecurityException)
                class MyClass {
                    void doIt() { }
                }
                new MyClass().doIt()
            """)
}

```

### The SecureASTCustomizer

This customizer will allow the developer of a DSL to restrict the grammar of the language, to prevent users from using some constructs, for example. It is only "secure" in that sense only and it is very important to understand that it does **not** replace a security manager. The only reason for it to exist is to limit the expressiveness of the language. This customizer only works at the AST (abstract syntax tree) level, not at runtime! It can be strange at first glance, but it makes much more sense if you think of Groovy as a platform to build DSLs. You may not want a user to have a complete language at hand. In the example below, we will demonstrate it using an example of language that only allows arithmetic operations, but this customizer allows you to:

- allow/disallow creation of closures
- allow/disallow imports
- allow/disallow package definition

- allow/disallow definition of methods
- restrict the receivers of method calls
- restrict the kind of AST expressions a user can use
- restrict the tokens (grammar-wise) a user can use
- restrict the types of the constants that can be used in code

For all those features, the secure AST customizer works using either a whitelist (list of elements that are allowed) or a blacklist (list of elements that are disallowed). For each type of feature (imports, tokens, ...) you have the choice to use either a whitelist or a blacklist, but you can mix whitelists and blacklists for distinct features. In general, you will choose whitelists (disallow all, allow selected).

```
org.codehaus.groovy.control.customizers.SecureASTCustomizer
```

```
def scz = new SecureASTCustomizer()
scz.with {
    closuresAllowed = false // user will not
be able to write closures
    methodDefinitionAllowed = false // user
will not be able to define methods
    importsWhitelist = [] // empty whitelist
means imports are disallowed
    staticImportsWhitelist = [] // same for
static imports
    staticStarImportsWhitelist =
['java.lang.Math'] // only java.lang.Math is
allowed
    // the list of tokens the user can find
// constants are defined in
org.codehaus.groovy.syntax.Types
    tokensWhitelist = [
        PLUS,
        MINUS,
        MULTIPLY,
        DIVIDE,
```

```
        MOD,  
        POWER,  
        PLUS_PLUS,  
        MINUS_MINUS,  
        COMPARE_EQUAL,  
        COMPARE_NOT_EQUAL,  
        COMPARE_LESS_THAN,  
        COMPARE_LESS_THAN_EQUAL,  
        COMPARE_GREATER_THAN,  
        COMPARE_GREATER_THAN_EQUAL,  
    ].asImmutable()  
    // limit the types of constants that a user  
    can define to number types only  
    constantTypesClassesWhiteList = [  
        Integer,  
        Float,  
        Long,  
        Double,  
        BigDecimal,  
        Integer.TYPE,  
        Long.TYPE,  
        Float.TYPE,  
        Double.TYPE  
    ].asImmutable()  
    // method calls are only allowed if the  
    receiver is of one of those types  
    // be careful, it's not a runtime type!  
    receiversClassesWhiteList = [  
        Math,  
        Integer,  
        Float,
```

**Double,**  
**Long,**

```
        BigDecimal
    ].asImmutable()
}
```

If what the secure AST customizer provides out of the box isn't enough for your needs, before creating your own compilation customizer, you might be interested in the expression and statement checkers that the AST customizer supports. Basically, it allows you to add custom checks on the AST tree, on expressions (expression checkers) or statements (statement checkers). For this, you must implement *org.codehaus.groovy.control.customizers.SecureASTCustomizer.StatementChecker* or *org.codehaus.groovy.control.customizers.SecureASTCustomizer.ExpressionChecker*.

Those interfaces define a single method called *isAuthorized*, returning a boolean, and taking a *Statement* (or *Expression*) as a parameter. It allows you to perform complex logic over expressions or statements to tell if a user is allowed to do it or not. As an example, let's think of a DSL for which you want to make sure that users only call methods for which the name is in lowercase:

```
scz.addExpressionChecker { expr ->
    if (expr instanceof MethodCallExpression)
{
    return
expr.methodAsString?.toLowerCase() ==
expr.methodAsString
}
true
} as ExpressionChecker
```

Here, we say that if the expression is a method call expression, then we can check the name and return true only if it's all lowercase. Otherwise, the expression is allowed.

### The SourceAwareCustomizer

This customizer, available since Groovy 2.1.0 only, is a bit special in the sense that it may be used as a filter for other customizers. The filter, in that case, is the *org.codehaus.groovy.control.SourceUnit*. For this, the source aware customizer takes another customizer as a delegate, and it will apply customization of that delegate only and only if predicates on the source unit match.

SourceUnit gives you access to interesting things, in particular the file being compiled (if compiling from a file, of course), which gives you the potential to perform operation based on the file name, for example. Here is how you would create a source aware customizer:

```
import
org.codehaus.groovy.control.customizers.SourceAwareCustomizer
import
org.codehaus.groovy.control.customizers.ImportCustomizer

def delegate = new ImportCustomizer()
def sac = new
SourceAwareCustomizer(delegate)
```

Then you can use predicates on the source aware customizer:

```
// the customizer will only be applied to
classes ending with 'Bean'
sac.baseNameValidator = { baseName ->
    baseName.endsWith 'Bean'
}

// the customizer will only be applied to
files which extension is '.spec'
sac.extensionValidator = { ext -> ext ==
'spec' }

// source unit validation
sac.sourceUnitValidator = { sourceUnit ->
... }
```

The customization builder

If you are using compilation customizers in Groovy code (like the examples above) and you are using Groovy 2.1+, then you can use an alternative syntax to customize the compilation. A builder `org.codehaus.groovy.control.customizers.builder.CompilerCustomizationBuilder` is available. Creating a customizer has never been so easy!

```
import  
org.codehaus.groovy.control.CompilerConfigur  
ation  
import static  
org.codehaus.groovy.control.customizers.buil  
der.CompilerCustomizationBuilder.withConfig  
  
def conf = new CompilerConfiguration()  
withConfig(conf) {  
    ...  
}
```

The code sample above shows how to use the builder. A static method, `withConfig`, takes a closure corresponding to the builder code, and automatically registers compilation customizers to the configuration. You can use:

#### ***Imports***

```

withConfig(configuration) {
    imports { // imports customizer
        normal 'my.package.MyClass' // a normal
import
        alias 'AI',
'java.util.concurrent.atomic.AtomicInteger'
// an aliased import
        star 'java.util.concurrent' // star
imports
        staticMember 'java.lang.Math', 'PI' //
static import
        staticMember 'pi', 'java.lang.Math',
'PI' // aliased static import
    }
}

```

*For the AST customizer*

```

// apply @Log transparently
withConfig(conf) {
    ast(Log)
}

// apply @Log, with a different logger name
withConfig(conf) {
    ast(Log, value: 'LOGGER')
}

```

*Secure ast*

```
withConfig(conf) {
    secureAst {
        closuresAllowed = false
        methodDefinitionAllowed = false
    }
}
```

*Source based*

```
// apply CompileStatic AST annotation on
.sgroovy files
withConfig(configuration){
    source(extension: 'sgroovy') {
        ast(CompileStatic)
    }
}

// apply CompileStatic AST annotation on
.sgroovy or .sg files
withConfig(configuration){
    source(extensions: ['sgroovy', 'sg']) {
        ast(CompileStatic)
    }
}

// apply CompileStatic AST annotation on
.sgroovy or .sg files
withConfig(configuration) {
    source(extensionValidator: { it.name in
['sgroovy', 'sg']}) {
```

```
        ast(CompileStatic)
    }
}

// apply CompileStatic AST annotation on
files whose name is 'foo'
withConfig(configuration) {
    source(basename: 'foo') {
        ast(CompileStatic)
    }
}

// apply CompileStatic AST annotation on
files whose name is 'foo' or 'bar'
withConfig(configuration) {
    source(basenames: ['foo', 'bar']) {
        ast(CompileStatic)
    }
}

// apply CompileStatic AST annotation on
files whose name is 'foo' or 'bar'
withConfig(configuration) {
    source(basenameValidator: { it in
['foo', 'bar'] }) {
        ast(CompileStatic)
    }
}

// apply CompileStatic AST annotation on
files that do not contain a class named
```

```
'Baz'  
withConfig(configuration) {  
    source(unitValidator: { unit ->  
!unit.AST.classes.any { it.name == 'Baz' }  
}) {
```

```
        ast(CompileStatic)
    }
}
```

### *Inlining a customizer*

Inlined customizer allows you to write a compilation customizer directly, without having to create a dedicated class for it.

```
withConfig(configuration) {
    inline(phase: 'CONVERSION') { source,
context, classNode ->
        println "visiting $classNode"
    }
}
```

### *Multiple customizers*

Of course, the builder allows you to define multiple customizers at once:

```
withConfig(configuration) {
    ast(ToString)
    ast(EqualsAndHashCode)
}
```

## Configuring compilation using groovyc or the ant task

For now, we've shown you how to customize compilation using a *CompilationConfiguration* access, but this is only possible if you embed Groovy and that you create your own instances of *CompilerConfiguration* (be it with *GroovyShell*, *GroovyScriptEngine*, ...). But if you want it to be applied on the classes you compile (with *groovyc*, *ant* or *gradle*, for example), until Groovy 2.1.0, there was no way to do that.

Since Groovy 2.1.0 (and Groovy 2.1.1 for the groovy Ant task), it is possible to use a compilation flag named *configs cript* that takes a groovy configuration script as a parameter. This script gives you access to the *CompilerConfiguration* instance **before** the files are compiled (exposed as a variable named *configuration*), so that you can tweak it. It also transparently integrates the compiler configuration builder above.

## Static compilation by default

Since static compilation has been released, many people asked for it to be enabled by default. For various reasons, including the fact that we think you should only limit static compilation to pieces of code where you have performance problems, we never included such a feature. Other people asked for default imports too. Since we didn't want to add lots of flags for each and every magic that Groovy can do, we decided to go for a configuration script. This means that having static compilation by default is just a matter of compiling classes using this configuration file. And the content is very easy:

```
withConfig(configuration) {  
    ast(groovy.transform.CompileStatic)  
}
```

You don't need to add an import for the builder, it's automatically added. Then, compile your files using the following command line:

```
groovyc -configscript src/conf/config.groovy  
src/main/groovy/MyClass.groovy
```

We strongly recommend you to separate configuration files from classes, hence the `src/main` and `src/conf` directories above.

## Ant Task Troubleshooting

### Ant Task Troubleshooting

#### *The Common Problem*

Very often, the `groovy` or `groovyc` tasks fail with a `ClassNotFoundException` for the class `GroovySourceAst`.

#### *The Reason*

If it's failing with a `ClassNotFoundException` for a class other than `GroovySourceAst`, welcome to Ant. As the Ant manual for [external tasks](#) says, "Don't add anything to the `CLASSPATH` environment variable - this is often the reason for very obscure errors. Use Ant's own mechanisms for adding libraries." And as its [library directories](#) section says, "Ant should work perfectly well with an empty `CLASSPATH` environment variable, something the `-noclasspath` option actually enforces. We get many more support calls related to classpath problems (especially quoting problems) than we like." So try running Ant as `ant -noclasspath`, or even alias `ant` to that in your shell.

If the class that isn't found is `GroovySourceAst` and the above doesn't help, somewhere you have a conflicting `antlr` in your classpath. This may be because you are using maven and one of its parts is polluting the classpath or you have a different `antlr` jar in your classpath somewhere.

#### *Solution 1: groovy-all*

Use the groovy-all-VERSION.jar from the groovy distribution and not the normal groovy jar. The groovy-all-VERSION.jar does already contain antlr and asm libs in a separate namespace so there should be no conflict with other libs around.

***Solution 2: using loaderref***

Sometimes it's not possible to use the groovy-all-VERSION.jar, for example because you want to build groovy before creating the jar. In this case you have to add a loaderref to the task definition. But that alone will not help. You have to add the rootLoaderRef task to set this loader reference. For example:

```
<taskdef name="rootLoaderRef"
  classname="org.codehaus.groovy.ant.RootLoaderRef"
  classpathref="task.classpath" />
<rootLoaderRef ref="tmp.groovy.groovyc">
  <classpath refid="execution.classpath" />
</rootLoaderRef>
<rootLoaderRef />
<taskdef name="groovy"
  classname="org.codehaus.groovy.ant.Groovy"
  loaderref="tmp.groovy.groovyc" />
```

The groovy task will now be created using the tmp.groovy.groovyc class loader, which tries to avoid loading conflicting jars like antlr. It's important to execute the rootLoaderRef task once before the taskdef using the loaderref defined by the rootLoaderRef.

***Solution 3: appropriate classpath set up***

You may need to adjust your classpath setup to include the jars you are trying to use. For instance, if you have placed the groovy jar in your Ant LIB folder, then Groovy will be in Ant's root classloader. If you now wish to refer to an external library, e.g. a JDBC driver, you may need to place that library also in your Ant LIB folder so that it is visible in the same classLoader as Groovy. See the loaderref discussion above also.

Also, the Groovy distribution doesn't include the entire Ant distribution. If you are using some optional Ant tasks, you may need to add some additional jars to your classpath to use the additional features. Here is an incomplete list of

some Ant tasks which require additional jars:

Ant Task	Additional Jar(s)
junitreport	ant-trax.jar, xercesImpl.jar, xml-apis.jar
mail	mail.jar, activation.jar, smtp.jar (if using SMTP), ant-javamail.jar (if sending MIME email)
sql	<i>your_JDBC_driver</i>

### All Solved?

No, both Solutions will not help if you have conflicting ant jars or common-logging jars somewhere. Solution 2 is able to solve much more difficult jar problems as long as your classpath is as clean as possible. if you want to be on the safe side you have to fork the javaVM which means you have to use the task like this:

```
<!-- lets fork a JVM to avoid classpath hell  
-->  
  <java  
  classname="org.codehaus.groovy.ant.Groovyc"  
  fork="yes" failonerror="true">  
    <classpath refid="project.classpath" />  
    <arg value="${build.classes.dir}" />  
    <arg value="${src.dir}" />  
  </java>
```

### References

- [Ant Integration with Groovy](#)
- [Developing Custom Tasks](#)

### Make a builder

To create a new builder like a the MarkupBuilder or AntBuilder, you have to implement in java (in groovy later too) a subclass of the groovy.util.BuilderSupport class.

The main methods to be implemented are the following :

- protected abstract void setParent(Object parent, Object child);
- protected abstract Object createNode(Object name); // a node without parameter and closure
- protected abstract Object createNode(Object name, Object value); //a node without parameters, but with closure
- protected abstract Object createNode(Object name, Map attributes); // a Node without closure but with parameters
- protected abstract Object createNode(Object name, Map attributes, Object value); //a node with closure and

parameters

- protected Object getName(String methodName)

*The BuilderSupport.java class*

```
package groovy.util;

import groovy.lang.Closure;
import groovy.lang.GroovyObjectSupport;

import java.util.List;
import java.util.Map;

import
org.codehaus.groovy.runtime.InvokerHelper;

public abstract class BuilderSupport extends
GroovyObjectSupport {

    private Object current;
    private Closure nameMappingClosure;
    private BuilderSupport proxyBuilder;

    public BuilderSupport() {
        this.proxyBuilder = this;
    }

    public BuilderSupport(BuilderSupport
proxyBuilder) {
        this(null, proxyBuilder);
    }
}
```

```
public BuilderSupport(Closure
nameMappingClosure, BuilderSupport
proxyBuilder) {
    this.nameMappingClosure =
nameMappingClosure;
    this.proxyBuilder = proxyBuilder;
}

public Object invokeMethod(String
methodName, Object args) {
    Object name = getName(methodName);
    return doInvokeMethod(methodName,
name, args);
}

protected Object doInvokeMethod(String
methodName, Object name, Object args) {
    Object node = null;
    Closure closure = null;
    List list =
InvokerHelper.asList(args);

    //System.out.println("Called
invokeMethod with name: " + name + "
arguments: " + list);

    switch (list.size()) {
        case 0:
            break;
        case 1:
```

```

        {
            Object object =
list.get(0);
            if (object instanceof
Map) {
                node =
proxyBuilder.createNode(name, (Map) object);
                } else if (object
instanceof Closure) {
                    closure = (Closure)
object;
                    node =
proxyBuilder.createNode(name);
                } else {
                    node =
proxyBuilder.createNode(name, object);
                }
            }
            break;
            case 2:
            {
                Object object1 =
list.get(0);
                Object object2 =
list.get(1);
                if (object1 instanceof
Map) {
                    if (object2 instanceof
Closure) {
                        closure =
(Closure) object2;

```

```

                                node =
proxyBuilder.createNode(name, (Map)
object1);
                                } else {
                                    node =
proxyBuilder.createNode(name, (Map) object1,
object2);
                                }
                                } else {
                                    if (object2 instanceof
Closure) {
                                        closure =
(Closure) object2;
                                        node =
proxyBuilder.createNode(name, object1);
                                    }
                                }
                                }
                                break;
                                case 3:
                                {
                                    Object attributes =
list.get(0);
                                    Object value =
list.get(1);
                                    closure = (Closure)
list.get(2);
                                    node =
proxyBuilder.createNode(name, (Map)
attributes, value);
                                }

```

```
        break;
    }

    if (node == null) {
        node =
proxyBuilder.createNode(name);
    }

    if (current != null) {
        proxyBuilder.setParent(current,
node);
    }

    if (closure != null) {
        // push new node on stack
        Object oldCurrent = current;
        current = node;

        // lets register the builder as
the delegate
        setClosureDelegate(closure,
node);

        closure.call();

        current = oldCurrent;
    }

    proxyBuilder.nodeCompleted(current,
node);
    return node;
}
```

```
    protected void
setClosureDelegate(Closure closure, Object
node) {
    closure.setDelegate(this);
}
```

```
    protected abstract void setParent(Object
parent, Object child);
    protected abstract Object
createNode(Object name);
    protected abstract Object
createNode(Object name, Object value);
    protected abstract Object
createNode(Object name, Map attributes);
    protected abstract Object
createNode(Object name, Map attributes,
Object value);
```

```
    protected Object getName(String
methodName) {
        if (nameMappingClosure != null) {
            return
nameMappingClosure.call(methodName);
        }
        return methodName;
    }
```

```
    protected void nodeCompleted(Object
parent, Object node) {
```

```
}  
  
protected Object getCurrent() {  
    return current;  
}  
  
protected void setCurrent(Object  
current) {
```

```
        this.current = current;
    }
}
```

#### *The NodeBuilder example*

To be able to write such a code :

```
def someBuilder = new NodeBuilder()

someBuilder.people(kind: 'folks',
groovy:true) {
    person(x:123, name: 'James',
cheese: 'edam') {
        project(name: 'groovy')
        project(name: 'geronimo')
    }
    person(x:234, name: 'bob',
cheese: 'cheddar') {
        project(name: 'groovy')
        project(name: 'drools')
    }
}
```

we need :

```
package groovy.util;
```

```
import java.util.ArrayList;
import java.util.Map;

/**
 * A helper class for creating nested trees
of Node objects for
 * handling arbitrary data
 *
 * @author <a
href="mailto:james@coredevelopers.net">James
Strachan</a>
 * @version $Revision: 1.3 $
 */
public class NodeBuilder extends
BuilderSupport {

    public static NodeBuilder newInstance()
{
        return new NodeBuilder();
    }

    protected void setParent(Object parent,
Object child) {
    }

    protected Object createNode(Object name)
{
        return new Node(getCurrentNode(),
name, new ArrayList());
    }
}
```

```
protected Object createNode(Object name,
Object value) {
    return new Node(getCurrentNode(),
name, value);
}
```

```
protected Object createNode(Object name,
Map attributes) {
    return new Node(getCurrentNode(),
name, attributes, new ArrayList());
}
```

```
protected Object createNode(Object name,
Map attributes, Object value) {
    return new Node(getCurrentNode(),
name, attributes, value);
}
```

```
protected Node getCurrentNode() {
```

```
        return (Node) getCurrent();
    }
}
```

*The MarkupBuilder.java class as second example*

```
package groovy.xml;

import groovy.util.BuilderSupport;
import groovy.util.IndentPrinter;

import java.io.PrintWriter;
import java.io.Writer;
import java.util.Iterator;
import java.util.Map;

/**
 * A helper class for creating XML or HTML
markup
 *
 * @author <a
href="mailto:james@coredevelopers.net">James
Strachan</a>
 * @author Stefan Matthias Aust
 * @version $Revision: 1.8 $
 */
public class MarkupBuilder extends
BuilderSupport {
```

```
private IndentPrinter out;
private boolean nospace;
private int state;
private boolean nodeIsEmpty = true;

public MarkupBuilder() {
    this(new IndentPrinter());
}

public MarkupBuilder(PrintWriter writer)
{
    this(new IndentPrinter(writer));
}

public MarkupBuilder(Writer writer) {
    this(new IndentPrinter(new
PrintWriter(writer)));
}

public MarkupBuilder(IndentPrinter out)
{
    this.out = out;
}

protected void setParent(Object parent,
Object child) {
}

/*
public Object getProperty(String
```

```

property) {
    if (property.equals("_")) {
        nospace = true;
        return null;
    } else {
        Object node =
createNode(property);
        nodeCompleted(getCurrent(),
node);
        return node;
    }
}
*/

protected Object createNode(Object name)
{
    toState(1, name);
    return name;
}

protected Object createNode(Object name,
Object value) {
    toState(2, name);
    out.print(">");
    out.print(value.toString());
    return name;
}

protected Object createNode(Object name,
Map attributes, Object value) {
    toState(1, name);

```

```

        for (Iterator iter =
attributes.entrySet().iterator();
iter.hasNext();) {
            Map.Entry entry = (Map.Entry)
iter.next();
            out.print(" ");

print(transformName(entry.getKey().toString(
)));
            out.print("='");

print(transformValue(entry.getValue().toStri
ng()));
            out.print("'");
        }
        if (value != null)
        {
            nodeIsEmpty = false;
            out.print(">" + value + "</" +
name + ">");
        }
        return name;
    }

    protected Object createNode(Object name,
Map attributes) {
        return createNode(name, attributes,
null);
    }

    protected void nodeCompleted(Object

```

```
parent, Object node) {
    toState(3, node);
    out.flush();
}

protected void print(Object node) {
    out.print(node == null ? "null" :
node.toString());
}

protected Object getName(String
methodName) {
    return
super.getName(transformName(methodName));
}

protected String transformName(String
name) {
    if (name.startsWith("_")) name =
name.substring(1);
    return name.replace('_', '-');
}

protected String transformValue(String
value) {
    return value.replaceAll("\\'",
"&quot;");
}

private void toState(int next, Object
name) {
```

```
switch (state) {
    case 0:
        switch (next) {
            case 1:
            case 2:
                out.print("<");
                print(name);
                break;
            case 3:
                throw new Error();
        }
        break;
    case 1:
        switch (next) {
            case 1:
            case 2:
                out.print(">");
                if (nospace) {
                    nospace = false;
                } else {
                    out.println();
                }
        }
        out.incrementIndent();

        out.printIndent();

        }
        out.print("<");
        print(name);
        break;
    case 3:
        if (nodeIsEmpty) {
```

```

        out.print("
/>");
    }
    break;
}
break;
case 2:
    switch (next) {
        case 1:
        case 2:
            throw new Error();
        case 3:
            out.print("</");
            print(name);
            out.print(">");
            break;
    }
    break;
case 3:
    switch (next) {
        case 1:
        case 2:
            if (nospace) {
                nospace = false;
            } else {
                out.println();
            }
    }
    out.print("<");
    print(name);

```

```
        break;
    case 3:
        if (nospace) {
            nospace = false;
        } else {
            out.println();

out.decrementIndent();

out.printIndent();

        }
        out.print("</");
        print(name);
        out.print(">");
        break;
    }
    break;
}
state = next;
```

```
    }  
}
```

## BuilderSupport

I was curious how the abstract BuildSupport class is working that does all those great things for e.g. the SwingBuilder and AntBuilder.

So I wrote the following Groovy Test that exposes its behaviour:

---

```
package groovy.util  
  
class SpoofBuilder extends BuilderSupport{  
    def log = []  
    protected void setParent(Object parent,  
Object child){  
        log << "sp"  
        log << parent  
        log << child  
    }  
    protected Object createNode(Object  
name){  
        log << 'cn1'  
        log << name  
        return 'x'  
    }  
    protected Object createNode(Object name,  
Object value){  
        log << 'cn2'  
        log << name  
    }  
}
```

```

        log << value
        return 'x'
    }
    protected Object createNode(Object name,
Map attributes){
        log << 'cn3'
        log << name
        attributes.each{entry -> log <<
entry.key; log << entry.value}
        return 'x'
    }
    protected Object createNode(Object name,
Map attributes, Object value){
        log << 'cn4'
        log << name
        attributes.each{entry -> log <<
entry.key; log << entry.value}
        log << value
        return 'x'
    }
    protected void nodeCompleted(Object
parent, Object node) {
        log << 'nc'
        log << parent
        log << node
    }
}

// simple node
def b = new SpoofBuilder()
assert b.log == []

```

```
def node = b.foo()
assert b.log == ['cn1', 'foo', 'nc', null,
node]
```

```
// simple node with value
def b = new SpoofBuilder()
def node = b.foo('value')
assert b.log == ['cn2', 'foo', 'value',
'nc', null, node]
```

```
// simple node with one attribute
def b = new SpoofBuilder()
def node = b.foo(name: 'value')
assert b.log == [
    'cn3', 'foo', 'name', 'value',
'nc', null, 'x']
```

```
// how is closure applied?
def b = new SpoofBuilder()
b.foo(){
    b.bar()
}
assert b.log == [
    'cn1', 'foo',
    'cn1', 'bar',
```

```
'sp', 'x', 'x',  
      'nc', 'x', 'x',  
'nc', null, 'x']
```

---

The SpoofBuilder is a sample instance of the abstract BuilderSupport class that does nothing but logging how it was called, returning 'x' for each node.

The test sections call the SpoofBuilder in various ways and the log reveals what methods were called during the "Build".

This test allowed me to verify my assumption on how the builder pattern works here. I used this knowledge to write a specialized AntBuilder for

## Canoo WebTest

. This "MacroStepBuilder" allows using the Canoo WebTest "steps" (that walk through a webapp for testing) from Groovy Code. Groovy has now become a first-class citizen in the

## Canoo WebTest Community

---

When writing the above test I stumbled over a few things, here are two of them:

- I was not able to write a fully fledged subclass of GroovyTestCase with separate methods for the various tests. I couldn't find out how to make the SpoofBuilder an inner class of my TestCase. I would very much appreciate help on this.
- Coming from Ruby I expected the << operator on Strings to operate on the String itself (like it does on Lists) rather than giving back a modified copy. It appears to me that << on Strings and on Lists is not consistent. Same with the "+" operator.

What I especially appreciated:

- == on Lists is clear and compact
- display of evaluated expression when assert fails saves a lot of work when writing assertions. Most of the time you need no extra message.

keep up the good work!

[mittie](#)

## **Compiling Groovy**

There are various options for compiling Groovy code and then either running it or using the Java objects it creates in Java code.

## Compiling Groovy code to bytecode using a script

There is an Ant task called **groovyc** which works pretty similarly to the **javac** Ant task which takes a bunch of groovy source files and compiles them into Java bytecode. Each groovy class then just becomes a normal Java class you can use inside your Java code if you wish.

Indeed the generated Java class is indistinguishable from a normal Java class, other than it implements the [GroovyObject](#) interface.

## Compiling Groovy code to bytecode using Ant and Maven

The [groovyc](#) Ant task is implemented by the [Groovyc](#) class. You can see an example of this in action inside Groovy's maven.xml file (just search for 'groovyc')

There is also an excellent [article on DeveloperWorks](#) which will show you how to compile Groovy code from within Maven, similarly to what is done with Ant.

You can also use the Ant task [from within Maven2](#), or the [groovy-maven-plugin](#)

## Dynamically using Groovy inside Java applications

If you don't want to explicitly compile groovy code to bytecode you can just [embed groovy](#) directly into your Java application.

## Runtime dependencies

As well as Java 1.4, or above, and the Groovy jar we also depend at runtime on the ASM library (asm and asm-tree mainly), as well as Antlr. You can also use the groovy-all-xxx.jar from your GROOVY\_HOME/embeddable directory, which embeds ASM and Antlr in its own namespace, to avoid Jar version hell.

## Compiling With Maven2

There are several approaches to compiling Groovy code in your Maven projects. [GMaven](#) is the most flexible and feature rich, but it has some difficulties with joint Java-Groovy projects and it is no longer under active development. The [Groovy-Eclipse compiler plugin for Maven](#) sidesteps the joint compilation issues. Read [this page](#) for a deeper discussion of the benefits and disadvantages of the two approaches.

A third approach is to use Maven's Ant plugin to compile a groovy project. Note that the Ant plugin is bound to the compile and test-compile phases of the build in the example below. It will be invoked during these phases and the contained tasks will be carried out which runs the Groovy compiler over the source and test directories. The resulting Java classes will coexist with and be treated like any standard Java classes compiled from Java source and will appear no different to the JRE, or the JUnit runtime.

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

  xsi:schemaLocation="http://maven.apache.org/
```

POM/4.0.0

```
http://maven.apache.org/maven-v4_0_0.xsd">
```

```
  <modelVersion>4.0.0</modelVersion>
```

```
  <groupId>com.mycomp.MyGroovy</groupId>
```

```
  <artifactId>MyGroovy</artifactId>
```

```
  <packaging>jar</packaging>
```

```
  <version>1.0-SNAPSHOT</version>
```

```
  <name>Maven Example building a Groovy  
project</name>
```

```
  <dependencies>
```

```
    <dependency>
```

```
      <groupId>junit</groupId>
```

```
      <artifactId>junit</artifactId>
```

```
      <version>3.8.1</version>
```

```
      <scope>test</scope>
```

```
    </dependency>
```

```
    <dependency>
```

```
      <groupId>groovy</groupId>
```

```
      <artifactId>groovy-all-1.0-jsr</artifactId>
```

```
      <version>05</version>
```

```
    </dependency>
```

```
  </dependencies>
```

```
  <build>
```

```
    <plugins>
```

```
      <plugin>
```

```
      <artifactId>maven-antrun-plugin</artifactId>
```

```
        <executions>
```

```
          <execution>
```

```
            <id>compile</id>
```

```

<phase>compile</phase>
    <configuration>
        <tasks>
            <mkdir
dir="{basedir}/src/main/groovy" />
            <taskdef
name="groovyc"
classname="org.codehaus.groovy.ant.Groovyc">
<classpath refid="maven.compile.classpath" />
            </taskdef>
            <mkdir
dir="{project.build.outputDirectory}" />
            <groovyc
destdir="{project.build.outputDirectory}"
srcdir="{basedir}/src/main/groovy/"
listfiles="true">
<classpath refid="maven.compile.classpath" />
            </groovyc>
        </tasks>
    </configuration>
<goals>
    <goal>run</goal>
</goals>
</execution>
<execution>

```

```
<id>test-compile</id>

<phase>test-compile</phase>
    <configuration>
        <tasks>
            <mkdir
dir="${basedir}/src/test/groovy" />
            <taskdef
name="groovyc"

classname="org.codehaus.groovy.ant.Groovyc">

<classpath refid="maven.test.classpath" />
            </taskdef>
            <mkdir
dir="${project.build.testOutputDirectory}" />
            <groovyc
destdir="${project.build.testOutputDirectory}"
srcdir="${basedir}/src/test/groovy/"
listfiles="true">

<classpath refid="maven.test.classpath" />
            </groovyc>
        </tasks>
    </configuration>
    <goals>
        <goal>run</goal>
    </goals>
</execution>
```

```
</executions>  
</plugin>
```

```
        </plugins>
    </build>
</project>
```

This assumes you have a Maven project setup with "groovy" subfolders as peers to the java src and test subfolders. You can use the java/jar archetype to set this up then rename the java folders to groovy or keep the java folders and just create groovy peer folders. There exists, also a groovy plugin which has not been tested or used in production. After defining the build section as in the above example, you can invoke the typical Maven build phases normally. For example, "mvn test" will execute the test phase, compiling Groovy source and Groovy test source and finally executing the unit tests. If you run "mvn jar" it will execute the jar phase bundling up all of your compiled production classes into a jar after all of the unit tests pass. For more detail on Maven build phases consult the Maven2 documentation.

## Creating an extension module

### Extension modules for extension methods

#### What is an extension method?

Since version 2.0, Groovy supports the notion of extension module. Basically, an extension module allows you to plugin extension methods to existing classes, just like regular Groovy does. For example, when you write:

### Standard extension method

```
def file = new File(...)
def contents = file.getText('utf-8')
```

The `getText` method doesn't exist on the `File` class. However, Groovy knows it because it is defined in a special class `ResourceGroovyMethods` (formerly, the method was defined in `DefaultGroovyMethods`):

### ResourceGroovyMethods.java

```
public static String getText(File file,
String charset) throws IOException {
    return
    IOGroovyMethods.getText(newReader(file,
charset));
}
```

You may notice that the extension method is defined using a static method in a "helper" class (where various extension methods are defined). The first argument of the *getText* method corresponds to the receiver, while additional parameters correspond to the arguments of the extension method. So here, we are defining a method called *getText* on the *File* class (because the first argument is a *File*), which takes a single argument as a parameter (the encoding *String*).

## Defining your own module

### *Concept*

Thanks to improved modularity, Groovy 2 now adds support for your custom extension methods. For that, you only have to write two things:

1. your extension class, just like *DefaultGroovyMethods*, *IOGroovyMethods*, ...
2. a descriptor file for your module

Then you just need to pack your extension module into a jar file and add it to the classpath! Groovy will automatically recognize the module at startup.

### *Writing your extension class*

As we saw before, adding methods to an existing class is as simple as writing a helper class which will only contain static methods. Imagine we want to add a method called *reverseToUpperCase* on the *String* class. This method would reverse the string then turn it to upper case. Then, you can write the following helper class:

## StringExtension

```
package com.example;

/**
 * An extension class for the String class.
 */
public class StringExtension {
    public static String
reverseToUpperCase(String self) {
        StringBuilder sb = new
StringBuilder(self);
        sb.reverse();
        return sb.toString().toUpperCase();
    }
}
```

That's all! Now imagine that you want to extend a class with a static method. For example, we would like to add an *andomUUID()* method to the *String* class which would return a random UUID string. Then, we must declare another extension class. The important thing to understand is that instance extension methods and static extension methods must be defined in two separate files. The only thing that will make a difference is the descriptor (see below). Then, the static extension helper class would look like this:

## StringStaticExtension.java

```
import java.util.UUID;

/**
 * Static extension methods for the String
 * class
 */
public class StringStaticExtension {
    public static String randomUUID(String
selfType) {
        return UUID.randomUUID().toString();
    }
}
```

### *The module descriptor*

For Groovy to be able to load your extension methods, you must declare your extension helper classes. You must create a file named `org.codehaus.groovy.runtime.ExtensionModule` into the `META-INF/services` directory:

## org.codehaus.groovy.runtime.ExtensionModule

```
moduleName=string-module
moduleVersion=1.0-test
extensionClasses=com.example.StringExtension
staticExtensionClasses=com.example.StringSta
ticExtension
```

The module descriptor requires 4 keys:

- *moduleName* : the name of your module
- *moduleVersion*: the version of your module. Note that version number is only used to check that you don't

load the same module in two different versions.

- *extensionClasses*: the list of extension helper classes for instance methods. You can provide several classes, given that they are comma separated.
- *staticExtensionClasses*: the list of extension helper classes for static methods. You can provide several classes, given that they are comma separated.

Note that it is not required for a module to define both static helpers and instance helpers, and that you may add several classes to a single module. You can also extend different classes in a single module without problem. It is even possible to use different classes in a single extension class, but it is recommended to group extension methods into classes by feature set.

## Advanced modules

Technically, the previous description about how to add extension methods to an existing class is a user-friendly way of doing it, but Groovy doesn't force you to use this standard mechanism. If you want to, you can write your own extension module implementation class, given that a module returns a list of MetaMethods to be added to a class. However, implementing such a module is not easy, and won't be covered by this tutorial. If you need to do that, please refer to the *org.codehaus.groovy.runtime.m12n.ExtensionModule* and *org.codehaus.groovy.runtime.m12n.ExtensionModuleRegistry* classes.

## FAQ

| *Are extension modules compatible with type checking?*

Yes, if the extension descriptor and the extension class is found on compilation classpath.

| *Are extension modules compatible with static compilation ?*

Yes, if the extension descriptor and the extension class is found on compilation classpath.

| *When I compile my project using Gradle and @CompileStatic, my code using an extension module fails compilation.*

As of Gradle 1.4, Gradle's native Groovy compiler integration doesn't seem to be compatible with extension modules. A fix is available in Groovy 2.1.2. As a workaround, you can use the Ant compiler integration:

```
compileGroovy.groovyOptions.useAnt = true  
compileTestGroovy.groovyOptions.useAnt =  
true
```

See <https://jira.codehaus.org/browse/GROOVY-6008>

## Design Patterns with Groovy

Using [design patterns](#) with Java is a well-established topic. Design patterns also apply to Groovy:

- some patterns carry over directly (and can make use of normal Groovy syntax improvements for greater readability)

- some patterns are no longer required because they are built right into the language or because Groovy supports a better way of achieving the intent of the pattern
- some patterns that have to be expressed at the design level in other languages can be implemented directly in Groovy (due to the way Groovy can blur the distinction between design and implementation)

### Patterns

- [Abstract Factory Pattern](#)
- [Adapter Pattern](#)
- [Bouncer Pattern](#)
- [Chain of Responsibility Pattern](#)
- [Composite Pattern](#)
- [Decorator Pattern](#)
- [Delegation Pattern](#)
- [Flyweight Pattern](#)
- [Iterator Pattern](#)
- [Loan my Resource Pattern](#)
- [Null Object Pattern](#)
- [Pimp my Library Pattern](#)
- [Proxy Pattern](#)
- [Singleton Pattern](#)
- [State Pattern](#)
- [Strategy Pattern](#)
- [Template Method Pattern](#)
- [Visitor Pattern](#)

### References

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 0-201-63361-2.
  - *The canonical reference of design patterns.*
2. Martin Fowler (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley. ISBN 0-201-48567-2.
3. Joshua Kerievsky (2004). *Refactoring To Patterns*. Addison-Wesley. ISBN 0-321-21335-1.
4. Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates (2004). *Head First Design Patterns*. O'Reilly. ISBN 0-596-00712-4.
  - *A great book to read, informative as well as amusing.*
5. Dierk Koenig with Andrew Glover, Paul King, Guillaume Laforge and Jon Skeet (2007). *Groovy in Action*. Manning. ISBN 1-932394-84-2.
  - *Discusses Visitor, Builder and other Patterns.*
6. Brad Appleton (1999). [Pizza Inversion - a Pattern for Efficient Resource Consumption](#).
  - *One of the most frequently used patterns by many software engineers!*
7. *Design Patterns in Dynamic Languages* by Neil Ford. Houston Java User's Group. Examples in Groovy and Ruby. [http://www.hjug.org/present/Neal\\_Ford-Design\\_Patterns\\_in\\_Dynamic\\_Languages-slides.pdf](http://www.hjug.org/present/Neal_Ford-Design_Patterns_in_Dynamic_Languages-slides.pdf)

See also: [Refactoring with Groovy](#)

### Abstract Factory Pattern

The [Abstract Factory Pattern](#) provides a way to encapsulate a group of individual factories that have a common theme. It embodies the intent of a normal factory, i.e. remove the need for code using an interface to know the concrete implementation behind the interface, but applies to a set of interfaces and selects an entire family of concrete classes which implement those interfaces.

As an example, I might have interfaces Button, TextField and Scrollbar. I might have WindowsButton, MacButton, FlashButton as concrete classes for Button. I might have WindowsScrollBar, MacScrollBar and FlashScrollBar as

concrete implementations for ScrollBar. Using the Abstract Factory Pattern should allow me to select which windowing system (i.e. Windows, Mac, Flash) I want to use once and from then on should be able to write code that references the interfaces but is always using the appropriate concrete classes (all from the one windowing system) under the covers.

#### Example

Suppose we want to write a game system. We might note that many games have very similar features and control.

We decide to try to split the common and game-specific code into separate classes.

First let's look at the game-specific code for a [Two-up](#) game:

```
class TwoupMessages {
    def welcome = 'Welcome to the twoup
game, you start with $1000'
    def done = 'Sorry, you have no money
left, goodbye'
}

class TwoupInputConverter {
    def convert(input) { input.toInteger() }
}

class TwoupControl {
    private money = 1000
    private random = new Random()
    private tossWasHead() {
        def next = random.nextInt()
        return next % 2 == 0
    }
    def moreTurns() {
        if (money > 0) {
            println "You have $money, how
much would you like to bet?"
            return true
        }
    }
}
```

```
    }  
    return false  
}  
def play(amount) {  
    def coin1 = tossWasHead()  
    def coin2 = tossWasHead()  
    if (coin1 && coin2) {  
        money += amount  
        println 'You win'  
    } else if (!coin1 && !coin2) {  
        money -= amount  
        println 'You lose'  
    }  
}
```

```
        } else println 'Draw'  
    }  
}
```

Now, let's look at the game-specific code for a number guessing game:

```
class GuessGameMessages {
    def welcome = 'Welcome to the guessing
game, my secret number is between 1 and 100'
    def done = 'Correct'
}
```

```
class GuessGameInputConverter {
    def convert(input) { input.toInteger() }
}
```

```
class GuessGameControl {
    private lower = 1
    private upper = 100
    private guess = new
Random().nextInt(upper - lower) + lower
    def moreTurns() {
        def done = (lower == guess || upper
== guess)
        if (!done) println "Enter a number
between $lower and $upper"
        !done
    }
    def play(nextGuess) {
        if (nextGuess <= guess) lower =
[lower, nextGuess].max()
        if (nextGuess >= guess) upper =
[upper, nextGuess].min()
    }
}
```

Now, let's write our factory code:

```
def guessFactory =  
[messages:GuessGameMessages,  
control:GuessGameControl,  
converter:GuessGameInputConverter]  
def twoupFactory = [messages:TwoupMessages,  
control:TwoupControl,  
converter:TwoupInputConverter]  
  
class GameFactory {  
    def static factory  
        def static getMessages() { return  
factory.messages.newInstance() }  
        def static getControl() { return  
factory.control.newInstance() }  
        def static getConverter() { return  
factory.converter.newInstance() }  
}
```

The important aspect of this factory is that it allows selection of an entire family of concrete classes.

Here is how we would use the factory:

```
GameFactory.factory = twoupFactory
def messages = GameFactory.messages
def control = GameFactory.control
def converter = GameFactory.converter
println messages.welcome
def reader = new BufferedReader(new
InputStreamReader(System.in))
while (control.moreTurns()){
    def input = reader.readLine().trim()
    control.play(converter.convert(input))
}
println messages.done
```

Note that the first line configures which family of concrete game classes we will use. It's not important that we selected which family to use by using the factory property as shown in the first line. Other ways would be equally valid examples of this pattern. For example, we may have asked the user which game they wanted to play or determined which game from an environment setting.

With the code as shown, the game might look like this when run:

**Welcome to the twoup game, you start with  
\$1000**

**You have 1000, how much would you like to  
bet?**

**300**

**Draw**

**You have 1000, how much would you like to  
bet?**

**700**

**You win**

**You have 1700, how much would you like to  
bet?**

**1700**

**You lose**

**Sorry, you have no money left, goodbye**

If we change the first line of the script to `GameFactory.factory = guessFactory`, then the sample run might look like this:

```
Welcome to the guessing game, my secret
number is between 1 and 100
Enter a number between 1 and 100
75
Enter a number between 1 and 75
35
Enter a number between 1 and 35
15
Enter a number between 1 and 15
5
Enter a number between 5 and 15
10
Correct
```

## Adapter Pattern

The [Adapter Pattern](#) (sometimes called the wrapper pattern) allows objects satisfying one interface to be used where another type of interface is expected. There are two typical flavours of the pattern: the *delegation* flavour and the *inheritance* flavour.

### Delegation Example

Suppose we have the following classes:

```
class SquarePeg {
    def width
}

class RoundPeg {
    def radius
}

class RoundHole {
    def radius
    def pegFits(peg) {
        peg.radius <= radius
    }
    String toString() { "RoundHole with
radius $radius" }
}
```

We can ask the RoundHole class if a RoundPeg fits in it, but if we ask the same question for a SquarePeg, then it will fail because the SquarePeg class doesn't have a radius property (i.e. doesn't satisfy the required interface).

To get around this problem, we can create an adapter to make it appear to have the correct interface. It would look like this:

```

class SquarePegAdapter {
  def peg
  def getRadius() {
    Math.sqrt(((peg.width/2) ** 2)*2)
  }
  String toString() {
    "SquarePegAdapter with peg width
    $peg.width (and notional radius $radius)"
  }
}

```

We can use the adapter like this:

```

def hole = new RoundHole(radius:4.0)
(4..7).each { w ->
  def peg = new SquarePegAdapter(peg:new
SquarePeg(width:w))
  if (hole.pegFits(peg))
    println "peg $peg fits in hole
$hole"
  else
    println "peg $peg does not fit in
hole $hole"
}

```

Which results in the following output:

**peg SquarePegAdapter with peg width 4 (and notional radius 2.8284271247461903) fits in hole RoundHole with radius 4.0**

**peg SquarePegAdapter with peg width 5 (and notional radius 3.5355339059327378) fits in hole RoundHole with radius 4.0**

**peg SquarePegAdapter with peg width 6 (and notional radius 4.242640687119285) does not fit in hole RoundHole with radius 4.0**

**peg SquarePegAdapter with peg width 7 (and notional radius 4.949747468305833) does not fit in hole RoundHole with radius 4.0**

#### **Inheritance Example**

Let's consider the same example again using inheritance. First, here are the original classes (unchanged):

```

class SquarePeg {
    def width
}

class RoundPeg {
    def radius
}

class RoundHole {
    def radius
    def pegFits(peg) {
        peg.radius <= radius
    }
    String toString() { "RoundHole with
radius $radius" }
}

```

An adapter using inheritance:

```

class SquarePegAdapter extends SquarePeg {
    def getRadius() {
        Math.sqrt(((width/2) ** 2)*2)
    }
    String toString() {
        "SquarePegAdapter with width $width
(and notional radius $radius)"
    }
}

```

Using the adapter:

```
def hole = new RoundHole(radius:4.0)
(4..7).each { w ->
  def peg = new SquarePegAdapter(width:w)
  if (hole.pegFits(peg))
    println "peg $peg fits in hole
$hole"
  else
    println "peg $peg does not fit in
hole $hole"
}
```

The output:

```
peg SquarePegAdapter with width 4 (and
notional radius 2.8284271247461903) fits in
hole RoundHole with radius 4.0
peg SquarePegAdapter with width 5 (and
notional radius 3.5355339059327378) fits in
hole RoundHole with radius 4.0
peg SquarePegAdapter with width 6 (and
notional radius 4.242640687119285) does not
fit in hole RoundHole with radius 4.0
peg SquarePegAdapter with width 7 (and
notional radius 4.949747468305833) does not
fit in hole RoundHole with radius 4.0
```

Adapting using Closures

As a variation of the previous examples, we could instead define the following interface:

```
interface RoundThing {
    def getRadius()
}
```

We can then define an adapter as a closure as follows:

```
def adapter = {
    p -> [getRadius:{Math.sqrt(((p.width/2)
** 2)*2)}] as RoundThing
}
```

And use it like this:

```
def peg = new SquarePeg(width:w)
if (hole.pegFits(adapter(peg)))
// ... as before
```

#### Adapting using the ExpandoMetaClass

As of Groovy 1.1, there is a built-in MetaClass which can automatically add properties and methods dynamically.

Here is how the example would work using that feature:

```
def peg = new SquarePeg(width:w)
peg.metaClass.radius =
Math.sqrt(((peg.width/2) ** 2)*2)
```

After you create a peg object, you can simply add a property to it on the fly. No need to change the original class and no need for an adapter class.

## Bouncer Pattern

The [Bouncer Pattern](#) describes usage of a method whose sole purpose is to either throw an exception (when particular conditions hold) or do nothing. Such methods are often used to defensively guard pre-conditions of a

method.

When writing utility methods, you should always guard against faulty input arguments. When writing internal methods, you may be able to ensure that certain pre-conditions always hold by having sufficient unit tests in place. Under such circumstances, you may reduce the desirability to have guards on your methods.

Groovy differs from other languages in that you frequently use the `assert` method within your methods rather than having a large number of utility checker methods or classes.

#### Null Checking Example

We might have a utility method such as:

```
class NullChecker {  
    static check(name, arg) {  
        if (arg == null)  
            throw new  
IllegalArgumentException(name + " is null")  
    }  
}
```

And we would use it like this:

```
public void doStuff(String name, Object  
value) {  
    NullChecker.check("name", name);  
    NullChecker.check("value", value);  
    // do stuff  
}
```

But a more Groovy way to do this would simply be like this:

```

public void doStuff(String name, Object
value) {
    assert name != null, 'name should not be
null'
    assert value != null, 'value should not
be null'
    // do stuff
}

```

#### Validation Example

As an alternative example, we might have this utility method:

```

public class NumberChecker {
    static final NUMBER_PATTERN =
/\d+(\.\d+(E-?\d+)?)?/
    static isNumber(str) {
        if (!str ==~ NUMBER_PATTERN)
            throw new
IllegalArgumentException(/Argument '$str'
must be a number/)
    }
    static isNotZero(number) {
        if (number == 0)
            throw new
IllegalArgumentException('Argument must not
be 0')
    }
}

```

And we would use it like this:

```

def stringDivide(String dividendStr, String
divisorStr) {
    NumberChecker.isNumber(dividendStr)
    NumberChecker.isNumber(divisorStr)
    def dividend = dividendStr.toDouble()
    def divisor = divisorStr.toDouble()
    NumberChecker.isNotZero(divisor)
    dividend / divisor
}

println stringDivide('1.2E2', '3.0')
// => 40.0

```

But with Groovy we could just as easily use:

```

def stringDivide(String dividendStr, String
divisorStr) {
    assert dividendStr =~
NumberChecker.NUMBER_PATTERN
    assert divisorStr =~
NumberChecker.NUMBER_PATTERN
    def dividend = dividendStr.toDouble()
    def divisor = divisorStr.toDouble()
    assert divisor != 0, 'Divisor must not
be 0'
    dividend / divisor
}

```

## Chain of Responsibility Pattern

In the [Chain of Responsibility Pattern](#), objects using and implementing an interface (one or more methods) are

intentionally loosely coupled. A set of objects that *implement* the interface are organised in a list (or in rare cases a tree). Objects using the interface make requests from the first *implementor* object. It will decide whether to perform any action itself and whether to pass the request further down the line in the list (or tree). Sometimes a default implementation for some request is also coded into the pattern if none of the implementors respond to the request.

#### Example

In this example, the script sends requests to the `lister` object. The `lister` points to a `UnixLister` object. If it can't handle the request, it sends the request to the `WindowsLister`. If it can't handle the request, it sends the request to the `DefaultLister`.

```
class UnixLister {
    private nextInLine
    UnixLister(next) { nextInLine = next }
    def listFiles(dir) {
        if (System.getProperty('os.name') ==
'Linux')
            println "ls $dir".execute().text
        else
            nextInLine.listFiles(dir)
    }
}

class WindowsLister {
    private nextInLine
    WindowsLister(next) { nextInLine = next
}
    def listFiles(dir) {
        if (System.getProperty('os.name') ==
'Windows XP')
            println "cmd.exe /c dir
$dir".execute().text
        else
            nextInLine.listFiles(dir)
    }
}
```

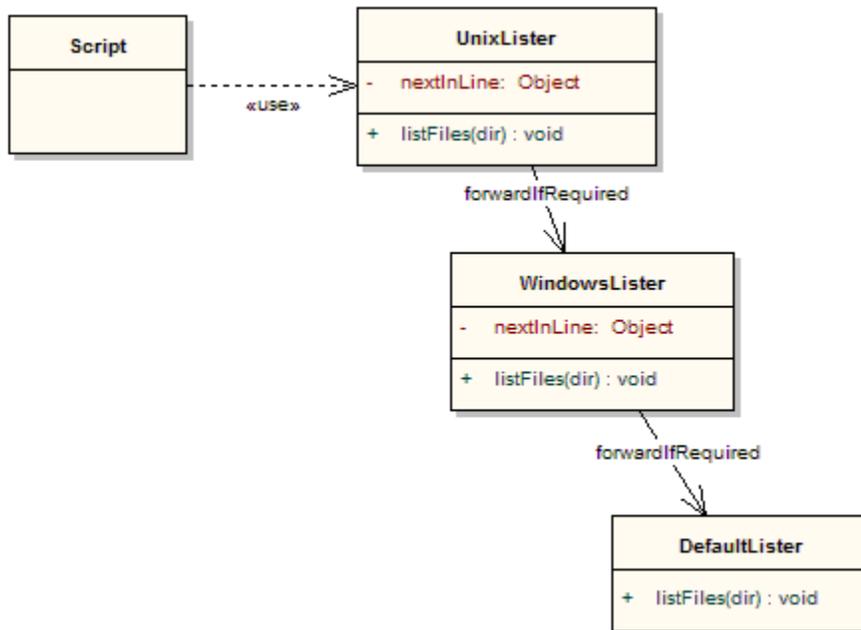
```
    }  
}  
  
class DefaultLister {  
    def listFiles(dir) {  
        new File(dir).eachFile{ f -> println  
f }  
    }  
}  
  
def lister = new UnixLister(new
```

```
WindowsLister(new DefaultLister()))
```

```
lister.listFiles('Downloads')
```

The output will be a list of files (with slightly different format depending on the operating system).

Here is a UML representation:



Variations to this pattern:

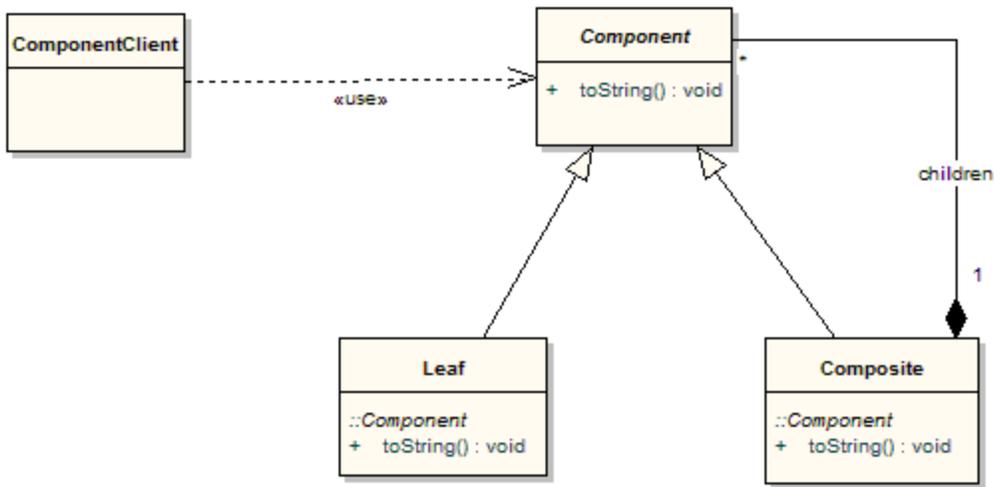
- we could have an explicit interface, e.g. `Lister`, to statically type the implementations but because of *duck-typing* this is optional
- we could use a chain tree instead of a list, e.g. `if (animal.hasBackbone()) delegate to VertebrateHandler else delegate to InvertebrateHandler`
- we could always pass down the chain even if we processed a request
- we could decide at some point to not respond and not pass down the chain
- we could use Groovy's meta-programming capabilities to pass unknown methods down the chain

## Composite Pattern

The [Composite Pattern](#) allows you to treat single instances of an object the same way as a group of objects. The pattern is often used with hierarchies of objects. Typically, one or more methods should be callable in the same way for either *leaf* or *composite* nodes within the hierarchy. In such a case, composite nodes typically invoke the same named method for each of their children nodes.

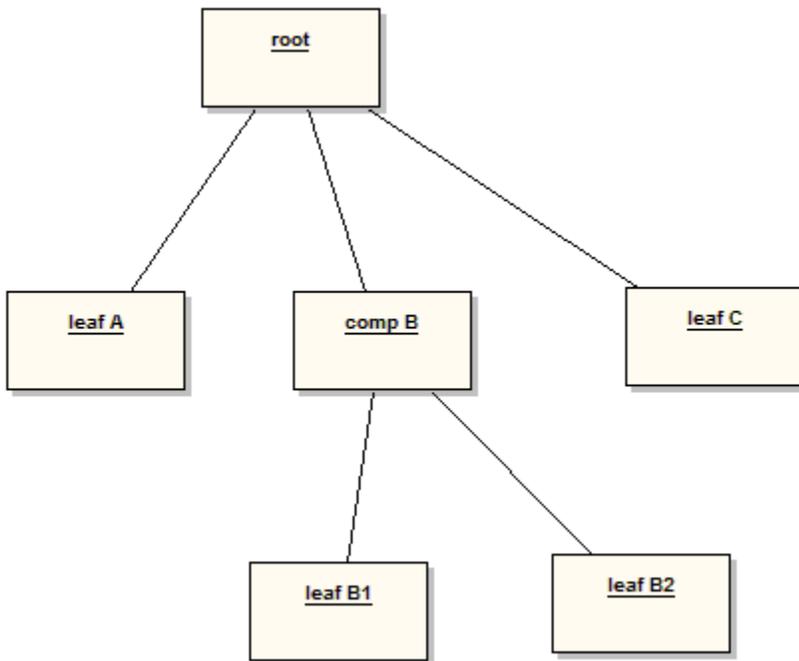
### An Example

Consider this usage of the composite pattern where we want to call `toString()` on either `Leaf` or `Composite` objects.



In Java, the `Component` class is essential as it provides the type used for both leaf and composite nodes. In Groovy, because of duck-typing, we don't need it for that purpose, however, it can still server as a useful place to place common behaviour between the leaf and composite nodes.

For our purposes, we will assemble the following hierarchy of components.



Here is the code:

```

// Empty code block placeholder
  
```

```

abstract class Component {
  def name
  def toString(indent) {
    ("-" * indent) + name
  }
}

class Composite extends Component {
  private children = []
  def toString(indent) {
    def s = super.toString(indent)
    children.each{ child ->
      s += "\n" + child.toString(indent+1)
    }
    return s
  }
  def leftShift(component) {
    children << component
  }
}

class Leaf extends Component {}

def root = new Composite(name:"root")
root << new Leaf(name:"leaf A")
def comp = new Composite(name:"comp B")
root << comp
root << new Leaf(name:"leaf C")
comp << new Leaf(name:"leaf B1")
comp << new Leaf(name:"leaf B2")
println root.toString(0)

```

Here is the resulting output:

```
root
-leaf A
-comp B
--leaf B1
--leaf B2
-leaf C
```

## Decorator Pattern

The [Decorator Pattern](#) provides a mechanism to embellish the behaviour of an object without changing its essential interface. A decorated object should be able to be substituted wherever the original (non-decorated) object was expected. Decoration typically does not involve modifying the source code of the original object and decorators should be able to be combined in flexible ways to produce objects with several embellishments.

### Traditional Example

Suppose we have the following `Logger` class.

```
class Logger {
    def log(String message) {
        println message
    }
}
```

There might be times when it is useful to timestamp a log message, or times when we might want to change the case of the message. We could try to build all of this functionality into our `Logger` class. If we did that, the `Logger` class would start to be very complex. Also, everyone would obtain all of features even when they might not want a small subset of the features. Finally, feature interaction would become quite difficult to control.

To overcome these drawbacks, we instead define two decorator classes. Uses of the `Logger` class are free to embellish their base logger with zero or more decorator classes in whatever order they desire. The classes look like this:

```

class TimeStampingLogger extends Logger {
  private Logger logger
  TimeStampingLogger(logger) {
    this.logger = logger
  }
  def log(String message) {
    def now = Calendar.instance
    logger.log("$now.time: $message")
  }
}

class UpperLogger extends Logger {
  private Logger logger
  UpperLogger(logger) {
    this.logger = logger
  }
  def log(String message) {
    logger.log(message.toUpperCase())
  }
}

```

We can use the decorators like so:

```

def logger = new UpperLogger(new
TimeStampingLogger(new Logger()))
logger.log("G'day Mate")
// => Tue May 22 07:13:50 EST 2007: G'DAY
MATE

```

You can see that we embellish the logger behaviour with both decorators. Because of the order we chose to apply

the decorators, our log message comes out capitalised and the timestamp is in normal case. If we swap the order around, let's see what happens:

```
logger = new TimeStampingLogger(new
UpperLogger(new Logger()))
logger.log('Hi There')
// => TUE MAY 22 07:13:50 EST 2007: HI THERE
```

Now the timestamp itself has also been changed to be uppercase.

#### A touch of dynamic behaviour

Our previous decorators were specific to `Logger` objects. We can use Groovy's Meta-Object Programming capabilities to create a decorator which is far more general purpose in nature. Consider this class:

```
class GenericLowerDecorator {
    private delegate
    GenericLowerDecorator(delegate) {
        this.delegate = delegate
    }
    def invokeMethod(String name, args) {
        def newargs = args.collect{ arg ->
            if (arg instanceof String)
return arg.toLowerCase()
            else return arg
        }
        delegate.invokeMethod(name, newargs)
    }
}
```

It takes any class and decorates it so that any `String` method parameter will automatically be changed to lower case.

```
logger = new GenericLowerDecorator(new  
TimeStampingLogger(new Logger()))  
logger.log('IMPORTANT Message')  
// => Tue May 22 07:27:18 EST 2007:  
important message
```

Just be careful with ordering here. The original decorators were restricted to decorating `Logger` objects. This decorator work with any object type, so we can't swap the ordering around, i.e. this won't work:

```
// Can't mix and match Interface-Oriented  
and Generic decorators  
// logger = new TimeStampingLogger(new  
GenericLowerDecorator(new Logger()))
```

We could overcome this limitation by generating an appropriate Proxy type at runtime but we won't complicate the example here.

#### **Runtime behaviour embellishment**

You can also consider using the `ExpandoMetaClass` from Groovy 1.1 to dynamically embellish a class with behaviour. This isn't the normal style of usage of the decorator pattern (it certainly isn't nearly as flexible) but may help you to achieve similar results in some cases without creating a new class.

Here's what the code looks like:

```
// current mechanism to enable
ExpandoMetaClass
GroovySystem.metaClassRegistry.metaClassCrea
tionHandle = new
ExpandoMetaClassCreationHandle()

def logger = new Logger()
logger.metaClass.log = { String m -> println
'message: ' + m.toUpperCase() }
logger.log('x')
// => message: X
```

This achieves a similar result to applying a single decorator but we have no way to easily apply and remove embellishments on the fly.

#### More dynamic decorating

Suppose we have a calculator class. (Actually any class would do.)

```
class Calc {
    def add(a, b) { a + b }
}
```

We might be interested in observing usage of the class over time. If it is buried deep within our codebase, it might be hard to determine when it is being called and with what parameters. Also, it might be hard to know if it is performing well. We can easily make a generic tracing decorator that prints out tracing information whenever any method on the `Calc` class is called and also provide timing information about how long it took to execute. Here is the code for the tracing decorator:

```

class TracingDecorator {
  private delegate
  TracingDecorator(delegate) {
    this.delegate = delegate
  }
  def invokeMethod(String name, args) {
    println "Calling $name$args"
    def before =
System.currentTimeMillis()
    def result =
delegate.invokeMethod(name, args)
    println "Got $result in
${System.currentTimeMillis()-before} ms"
    result
  }
}

```

Here is how to use the class in a script:

```

def tracedCalc = new TracingDecorator(new
Calc())
assert 15 == tracedCalc.add(3, 12)

```

And here is what you would see after running this script:

```

Calling add{3, 12}
Got 15 in 31 ms

```

The above timing example hooks into the lifecycle of Groovy objects (via `invokeMethod`). This is such an important style performing meta-programming that Groovy has special support for this style of decorating using *interceptors*.

Groovy even comes with a built-in `TracingInterceptor`. We can extend the built-in class like this:

```
class TimingInterceptor extends
TracingInterceptor {
    private beforeTime
    def beforeInvoke(object, String
methodName, Object[] arguments) {
        super.beforeInvoke(object,
methodName, arguments)
        beforeTime =
System.currentTimeMillis()
    }
    public Object afterInvoke(Object object,
String methodName, Object[] arguments,
Object result) {
        super.afterInvoke(object,
methodName, arguments, result)
        def duration =
System.currentTimeMillis() - beforeTime
        writer.write("Duration: $duration
ms\n")
        writer.flush()
        return result
    }
}
```

Here is an example of using this new class:

```
def proxy =
ProxyMetaClass.getInstance(util.CalcImpl.class)
proxy.interceptor = new TimingInterceptor()
proxy.use {
    assert 7 == new util.CalcImpl().add(1,
6)
}
```

And here is the output:

```
before util.CalcImpl.ctor()
after  util.CalcImpl.ctor()
Duration: 0 ms
before util.CalcImpl.add(java.lang.Integer,
java.lang.Integer)
after  util.CalcImpl.add(java.lang.Integer,
java.lang.Integer)
Duration: 16 ms
```

#### Decorating with `java.lang.reflect.Proxy`

If you are trying to decorate an object (i.e. just a particular instance of the class, not the class generally), then you can use Java's `java.lang.reflect.Proxy`. Groovy makes working with this easier than just Java. Below is a code sample taken out of a grails project that wraps a `java.sql.Connection` so that its `close` method is a no-op:

```

protected Sql getGroovySql() {
    final Connection con =
session.connection()
    def invoker = { object, method, args
->
        if (method.name == "close") {
            log.debug("ignoring call to
Connection.close() for use by
groovy.sql.Sql")
        } else {
            log.trace("delegating $method")
            return
con.invokeMethod(method.name, args)
        }
    } as Invocahandler;
    def proxy = Proxy.newProxyInstance(
getClass().getClassLoader(), [Connection] as
Class[], invoker )
    return new Sql(proxy)
}

```

If there were many methods to intercept, then this approach could be modified to look up closure in a map by method name and invoke it.

#### Decorating with Spring

The [Spring Framework](#) allows decorators to be applied with *interceptors* (you may have heard the terms *advice* or *a spect*). You can leverage this mechanism from Groovy as well.

First define a class that you want to decorate (we'll also use an interface as is normal Spring practice):

Here's the interface:

```
package util

interface Calc {
    def add(a, b)
}
```

Here's the class:

```
package util

class CalcImpl implements Calc {
    def add(a, b) { a + b }
}
```

Now, we define our wiring in a file called `beans.xml` as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema
/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:lang="http://www.springframework.org/s
chema/lang"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/
spring-beans.xsd
http://www.springframework.org/schema/lang
```

```
http://www.springframework.org/schema/lang/s
pring-lang.xsd">
    <bean id="performanceInterceptor"
autowire="no"

class="org.springframework.aop.interceptor.P
erformanceMonitorInterceptor">
    <property name="loggerName"
value="performance"/>
    </bean>
    <bean id="calc" class="util.CalcImpl"/>
    <bean
class="org.springframework.aop.framework.aut
oproxy.BeanNameAutoProxyCreator">
    <property name="beanNames"
value="calc"/>
    <property name="interceptorNames"
```

```
value="performanceInterceptor" />
  </bean>
</beans>
```

Now, our script looks like this:

```
import
org.springframework.context.support.ClassPat
hXmlApplicationContext

def ctx = new
ClassPathXmlApplicationContext("beans.xml")
def calc = ctx.getBean('calc')
println calc.add(3, 25)
```

And when we run it, we see the results:

```
21/05/2007 23:02:35
org.springframework.aop.interceptor.Performa
nceMonitorInterceptor invokeUnderTrace
FINEST: Stopwatch 'util.Calc.add': running
time (millis) = 16
```

You may have to adjust your `logging.properties` file for messages at log level `FINEST` to be displayed.

#### Asynchronous Decorators using GPar

Using the example code in [Panini](#) for inspiration. Here is a Groovy version that avoids using an `@AddedBehavior` annotation at the expense of not having as general an algorithm for selecting the methods to decorate. This isn't a limitation of the particular approach chosen but just a simplification for illustrative purposes (but don't assume below is an exact equivalent).

```
@Grab('org.codehaus.gpars:gpars:0.10')
import static
```

```
groovyx.gpars.GParsPool.withPool
```

```
interface Document {  
    void print()  
    String getText()  
}
```

```
class DocumentImpl implements Document {  
    def document  
    void print() { println document }  
    String getText() { document }  
}
```

```
def words(String text) {  
    text.replaceAll(/[^a-zA-Z]/, '  
' ).trim().split(/\s+/)*.toLowerCase()  
}
```

```
def avgWordLength = {  
    def words = words(it.text)  
    sprintf "Avg Word Length: %4.2f",  
words*.size().sum() / words.size()  
}
```

```
def modeWord = {  
    def wordGroups = words(it.text).groupBy  
{it}.collectEntries{ k, v -> [k, v.size()] }  
    def maxSize = wordGroups*.value.max()  
    def maxWords = wordGroups.findAll{  
it.value == maxSize }  
    "Mode Word(s): ${maxWords*.key.join(',  
' )} ($maxSize occurrences)"
```

```

}
def wordCount = { d -> "Word Count: " +
words(d.text).size() }

def asyncDecorator(Document d, Closure c) {
ProxyGenerator.INSTANCE.instantiatedDelegate(
[print: {
    withPool {
        def result = c.callAsync(d)
        d.print()
        println result.get()
    }
}], [Document], d)
}

```

```

Document d =
asyncDecorator(asyncDecorator(asyncDecorator
(
    new DocumentImpl(document:"This is
the file with the words in it\n\t\nDo you
see the words?\n"),
//          new DocumentImpl(document: new
File('AsyncDecorator.groovy').text),

```

```
wordCount), modeWord),  
avgWordLength)  
d.print()
```

## Delegation Pattern

The [Delegation Pattern](#) is a technique where an object's behavior (public methods) is implemented by delegating responsibility to one or more associated objects.

Groovy allows the traditional style of applying the delegation pattern, e.g. see [Replace Inheritance with Delegation](#).

### Implement Delegation Pattern using ExpandoMetaClass

The [ExpandoMetaClass](#) allows usage of this pattern to be encapsulated in a library. This allows Groovy to emulate similar libraries available for the Ruby language.

Consider the following library class:

```

class Delegator {
  private targetClass
  private delegate
  Delegator(targetClass, delegate) {
    this.targetClass = targetClass
    this.delegate = delegate
  }
  def delegate(String methodName) {
    delegate(methodName, methodName)
  }
  def delegate(String methodName, String
asMethodName) {

targetClass.metaClass."$asMethodName" =
delegate."&"$methodName"
  }
  def delegateAll(String[] names) {
    names.each { delegate(it) }
  }
  def delegateAll(Map names) {
    names.each { k, v -> delegate(k, v)
}
  }
  def delegateAll() {
    delegate.class.methods*.name.each {
delegate(it) }
  }
}

```

With this in your classpath, you can now apply the delegation pattern dynamically as shown in the following

examples. First, consider we have the following classes:

```
class Person {
  String name
}

class MortgageLender {
  def borrowAmount(amount) {
    "borrow $$amount"
  }
  def borrowFor(thing) {
    "buy $thing"
  }
}

def lender = new MortgageLender()

def delegator = new Delegator(Person,
lender)
```

We can now use the *delegator* to automatically borrow methods from the *lender* object to extend the *Person* class. We can borrow the methods as is or with a rename:

```
delegator.delegate 'borrowFor'  
delegator.delegate 'borrowAmount',  
'getMoney'  
  
def p = new Person()  
  
println p.borrowFor('present') // => buy  
present  
println p.getMoney(50) // =>  
borrow $50
```

The first line above, adds the *borrowFor* method to the *Person* class by delegating to the *lender* object. The second line adds a *getMoney* method to the *Person* class by delegating to the *lender* object's *borrowAmount* method.

Alternatively, we could borrow multiple methods like this:

```
delegator.delegateAll 'borrowFor',  
'borrowAmount'
```

Which adds these two methods to the *Person* class.

Or if we want all the methods, like this:

```
delegator.delegateAll()
```

Which will make all the methods in the delegate object available in the *Person* class.

Alternatively, we can use a map notation to rename multiple methods:

```
delegator.delegateAll  
borrowAmount: 'getMoney',  
borrowFor: 'getThing'
```

## Implement Delegation Pattern using @Delegate annotation

Since version 1.6 you can use the built-in delegation mechanism which is based on AST transformation. This make delegation even easier:

```
class Person {  
    def name  
    @Delegate MortgageLender mortgageLender  
= new MortgageLender()  
}  
  
class MortgageLender {  
    def borrowAmount(amount) {  
        "borrow $$amount"  
    }  
    def borrowFor(thing) {  
        "buy $thing"  
    }  
}  
  
def p = new Person()  
  
assert "buy present" ==  
p.borrowFor('present')  
assert "borrow $50" == p.borrowAmount(50)
```

## Flyweight Pattern

The [Flyweight Pattern](#) is a pattern for greatly reducing memory requirements by not requiring that heavy-weight objects be created in large numbers when dealing with systems that contain many things that are mostly the same. If for instance, a document was modeled using a complex character class that knew about unicode, fonts, positioning, etc., then the memory requirements could be quite large for large documents if each physical character in the document required its own character class instance. Instead, characters themselves might be kept within Strings and we might have one character class (or a small number such as one character class for each font type) that knew the specifics of how to deal with characters.

In such circumstances, we call the state that is shared with many other things (e.g. the character type) *intrinsic* state. It is captured within the heavy-weight class. The state which distinguishes the physical character (maybe just its ASCII code or Unicode) is called its *extrinsic* state.

#### Example

First we are going to model some complex aircraft (the first being a hoax competitor of the second - not that is relevant to the example).

```
class Boeing797 {
    def wingspan = '80.8 m'
    def capacity = 1000
    def speed = '1046 km/h'
    def range = '14400 km'
    // ...
}
```

```
class Airbus380 {
    def wingspan = '79.8 m'
    def capacity = 555
    def speed = '912 km/h'
    def range = '10370 km'
    // ...
}
```



If we want to model our fleet, our first attempt might involve using many instances of these heavy-weight objects. It turns out though that only a few small pieces of state (our extrinsic state) change for each aircraft, so we will have singletons for the heavy-weight objects and capture the extrinsic state (bought date and asset number in the code below) separately.

```
class FlyweightFactory {  
    static instances = [797: new
```

```

Boeing797(), 380: new Airbus380()]
}

class Aircraft {
    private type           // intrinsic state
    private assetNumber   // extrinsic state
    private bought        // extrinsic state
    Aircraft(typeCode, assetNumber, bought)
{
    type =
FlyweightFactory.instances[typeCode]
    this.assetNumber = assetNumber
    this.bought = bought
}
    def describe() {
        println ""
Asset Number: $assetNumber
Capacity: $type.capacity people
Speed: $type.speed
Range: $type.range
Bought: $bought
""
    }
}

def fleet = [
    new Aircraft(380, 1001, '10-May-2007'),
    new Aircraft(380, 1002, '10-Nov-2007'),
    new Aircraft(797, 1003, '10-May-2008'),
    new Aircraft(797, 1004, '10-Nov-2008')
]

```

```
]
```

```
fleet.each{ p -> p.describe() }
```

So here, even if our fleet contained hundreds of planes, we would only have one heavy-weight object for each type of aircraft.

As a further efficiency measure, we might use lazy creation of the flyweight objects rather than create the initial map up front as in the above example.

Running this script results in:

**Asset Number: 1001**  
**Capacity: 555 people**  
**Speed: 912 km/h**  
**Range: 10370 km**  
**Bought: 10-May-2007**

**Asset Number: 1002**  
**Capacity: 555 people**  
**Speed: 912 km/h**  
**Range: 10370 km**  
**Bought: 10-Nov-2007**

**Asset Number: 1003**  
**Capacity: 1000 people**  
**Speed: 1046 km/h**  
**Range: 14400 km**  
**Bought: 10-May-2008**

**Asset Number: 1004**  
**Capacity: 1000 people**  
**Speed: 1046 km/h**  
**Range: 14400 km**  
**Bought: 10-Nov-2008**

## Iterator Pattern

The [Iterator Pattern](#) allows sequential access to the elements of an aggregate object without exposing its underlying representation.

Groovy has the iterator pattern built right in to many of its *closure* operators, e.g. `each` and `eachWithIndex` as well as the `for .. in` loop.

For example:

```
def printAll(container) {
    for (item in container) { println item }
}

def numbers = [ 1,2,3,4 ]
def months = [ Mar:31, Apr:30, May:31 ]
def colors = [ java.awt.Color.BLACK,
java.awt.Color.WHITE ]
printAll numbers
printAll months
printAll colors
```

Results in the output:

```
1
2
3
4
May=31
Mar=31
Apr=30
java.awt.Color[r=0,g=0,b=0]
java.awt.Color[r=255,g=255,b=255]
```

Another example:

```
colors.forEachWithIndex{ item, pos ->
    println "Position $pos contains '$item' "
}
```

Results in:

```
Position 0 contains  
'java.awt.Color[r=0,g=0,b=0]'  
Position 1 contains  
'java.awt.Color[r=255,g=255,b=255]'
```

The iterator pattern is also built in to other special operators such as the `eachByte`, `eachFile`, `eachDir`, `eachLine`, `eachObject`, `eachMatch` operators for working with streams, URLs, files, directories and regular expressions matches.

### Loan my Resource Pattern

The [Loan my Resource](#) pattern ensures that a resource is deterministically disposed of once it goes out of scope.

This pattern is built in to many Groovy helper methods. You should consider using it yourself if you need to work with resources in ways beyond what Groovy supports.

#### Example

Consider the following code which works with a file. First we might write some line to the file and then print its size:

```
def f = new File('junk.txt')  
f.withPrintWriter { pw ->  
    pw.println(new Date())  
    pw.println(this.class.name)  
}  
println f.size()  
// => 42
```

We could also read back the contents of the file a line at a time and print each line out:

```

f.eachLine { line ->
    println line
}
// =>
// Mon Jun 18 22:38:17 EST 2007
// RunPattern

```

Note that normal Java `Reader` and `PrintWriter` objects were used under the covers by Groovy but the code writer did not have to worry about explicitly creating or closing those resources. The built-in Groovy methods `loan` the respective reader or writer to the closure code and then tidy up after themselves. So, you are using this pattern without having to do any work.

Sometimes however, you wish to do things slightly differently to what you can get for free using Groovy's built-in mechanisms. You should consider utilising this pattern within your own resource-handling operations.

Consider how you might process the list of words on each line within the file. We could actually do this one too using Groovy's built-in functions, but bear with us and assume we have to do some resource handling ourselves. Here is how we might write the code without using this pattern:

```

def reader = f.newReader()
reader.splitEachLine(' ') { wordList ->
    println wordList
}
reader.close()
// =>
// [ "Mon", "Jun", "18", "22:38:17", "EST",
"2007" ]
// [ "RunPattern" ]

```

Notice that we now have an explicit call to `close()` in our code. If we didn't code it just right (here we didn't surround the code in a `try ... finally` block, we run the risk of leaving the file handle open.

Let's now apply the loan pattern. First, we'll write a helper method:

```

def withListOfWordsForEachLine(File f,
Closure c) {
    def r = f.newReader()
    try {
        r.splitEachLine(' ', c)
    } finally {
        r?.close()
    }
}

```

Now, we can re-write our code as follows:

```

withListOfWordsForEachLine(f) { wordList ->
    println wordList
}
// =>
// [ "Mon", "Jun", "18", "22:38:17", "EST",
"2007" ]
// [ "RunPattern" ]

```

This is much simpler and has removed the explicit `close()`. This is now catered for in one spot so we can apply the appropriate level of testing or reviewing in just one spot to be sure we have no problems.

### Null Object Pattern

The [Null Object Pattern](#) involves using a special object place-marker object representing null. Typically, if you have a reference to null, you can't invoke `reference.field` or `reference.method()`. You receive the dreaded `NullPointerException`. The null object pattern uses a special object representing null, instead of using an actual `null`. This allows you to invoke field and method references on the null object. The result of using the null object should semantically be equivalent to *doing nothing*.

#### Simple Example

Suppose we have the following system:

```
class Job {
  def salary
}

class Person {
  def name
  def Job job
}

def people = [
  new Person(name:'Tom', job:new
Job(salary:1000)),
  new Person(name:'Dick', job:new
Job(salary:1200)),
]

def biggestSalary = people.collect{ p ->
p.job.salary }.max()
println biggestSalary
```

When run, this prints out 1200. Suppose now that we now invoke:

```
people << new Person(name:'Harry')
```

If we now try to calculate `biggestSalary` again, we receive a null pointer exception.

To overcome this problem, we can introduce a `NullJob` class and change the above statement to become:

```

class NullJob extends Job { def salary = 0 }

people << new Person(name:'Harry', job:new
NullJob())
biggestSalary = people.collect{ p ->
p.job.salary }.max()
println biggestSalary

```

This works as we require but it's not always the best way to do this with Groovy. Groovy's safe-dereference operator (`?.`) operator and null aware closures often allow Groovy to avoid the need to create a special null object or null class. This is illustrated by examining a groovier way to write the above example:

```

people << new Person(name:'Harry')
biggestSalary = people.collect{ p ->
p.job?.salary }.max()
println biggestSalary

```

Two things are going on here to allow this to work. First of all, `max()` is *'null aware'* so that `[300, null, 400].max() == 400`. Secondly, with the `?.` operator, an expression like `p?.job?.salary` will be equal to null if `salary` is equal to null, or if `job` is equal to null or if `p` is equal to null. You don't need to code a complex nested `if ... then ... else` to avoid a `NullPointerException`.

#### Tree Example

Consider the following example (inspired by [this](#)) where we want to calculate size, cumulative sum and cumulative product of all the values in a tree structure.

Our first attempt has special logic within the calculation methods to handle null values.

```

class NullHandlingTree {
    def left, right, value

    def size() {
        1 + (left ? left.size() : 0) +
        (right ? right.size() : 0)
    }
}

```

```
    }

    def sum() {
        value + (left ? left.sum() : 0) +
(right ? right.sum() : 0)
    }

    def product() {
        value * (left ? left.product() : 1) *
(right ? right.product() : 1)
    }
}

def root = new NullHandlingTree(
    value:2,
    left: new NullHandlingTree(
        value:3,
        right: new
NullHandlingTree(value:4),
        left: new NullHandlingTree(value:5)
    )
)
```

```
println root.size()
println root.sum()
println root.product()
```

If we introduce the null object pattern (here by defining the `NullTree` class), we can now simplify the logic in the `size()`, `sum()` and `product()` methods. These methods now much more clearly represent the logic for the normal (and now universal) case. Each of the methods within `NullTree` returns a value which represents doing nothing.

```
class Tree {
    def left = new NullTree(), right = new
NullTree(), value

    def size() {
        1 + left.size() + right.size()
    }

    def sum() {
        value + left.sum() + right.sum()
    }

    def product() {
        value * left.product() *
right.product()
    }
}

class NullTree {
    def size() { 0 }
    def sum() { 0 }
    def product() { 1 }
}
```

```
def root = new Tree(  
  value:2,  
  left: new Tree(  
    value:3,  
    right: new Tree(value:4),  
    left: new Tree(value:5)  
  )  
)
```

```
println root.size()  
println root.sum()  
println root.product()
```

The result of running either of these examples is:

```
4  
14  
120
```

Note: a slight variation with the null object pattern is to combine it with the singleton pattern. So, we wouldn't write `new NullTree()` wherever we needed a null object as shown above. Instead we would have a single null object instance which we would place within our data structures as needed.

## Pimp my Library Pattern

The [Pimp my Library](#) Pattern suggests an approach for extending a library that nearly does everything that you need but just needs a little more. It assumes that you do not have source code for the library of interest.

### Example

Suppose we want to make use of the built-in Integer facilities in Groovy (which build upon the features already in Java). Those libraries have nearly all of the features we want but not quite everything. We may not have all of the source code to the Groovy and Java libraries so we can't just change the library. Instead we augment the library. Groovy has a number of ways to do this. One way is to use a Category.

First, we'll define a suitable category.

```

class EnhancedInteger {
    static boolean greaterThanAll(Integer
self, Object[] others) {
        greaterThanAll(self, others)
    }
    static boolean greaterThanAll(Integer
self, others) {
        others.every{ self > it }
    }
}

```

We have added two methods which augment the Integer methods by providing the `greaterThanAll` method. Categories follow conventions where they are defined as static methods with a special first parameter representing the class we wish to extend. The `greaterThanAll(Integer self, others)` static method becomes the `greaterThanAll(other)` instance method.

We defined two versions of `greaterThanAll`. One which works for collections, ranges etc. The other which works with a variable number of Integer arguments.

Here is how you would use the category.

```

use(EnhancedInteger) {
    assert 4.greaterThanAll(1, 2, 3)
    assert !5.greaterThanAll(2, 4, 6)
    assert 5.greaterThanAll(-4..4)
    assert 5.greaterThanAll([])
    assert !5.greaterThanAll([4, 5])
}

```

As you can see, using this technique you can effectively enrich an original class without having access to its source code. Moreover, you can apply different enrichments in different parts of the system as well as work with un-enriched objects if we need to.

## Proxy Pattern

The [Proxy Pattern](#) allows one object to act as a pretend replacement for some other object. In general, whoever is

using the proxy, doesn't realise that they are not using the real thing. The pattern is useful when the real object is hard to create or use: it may exist over a network connection, or be a large object in memory, or be a file, database or some other resource that is expensive or impossible to duplicate.

#### Example

One common use of the proxy pattern is when talking to remote objects in a different JVM. Here is the client code for creating a proxy that talks via sockets to a server object as well as an example usage:

```
class AccumulatorProxy {  
  def accumulate(args) {  
    def result  
    def s = new Socket("localhost",  
54321)  
    s.withObjectStreams{ ois, oos ->  
      oos << args  
      result = ois.readObject()  
    }  
    s.close()  
    return result  
  }  
}  
  
println new  
AccumulatorProxy().accumulate([1, 2, 3, 4,  
5, 6, 7, 8, 9, 10])  
// => 55
```

Here is what your server code might look like (start this first):

```

class Accumulator {
  def accumulate(args) {
    args.inject(0){ total, arg -> total
+= arg }
  }
}

def port = 54321
def accumulator = new Accumulator()
def server = new ServerSocket(port)
println "Starting server on port $port"
while(true) {
  server.accept() { socket ->
    socket.withObjectStreams { ois, oos
->
      def args = ois.readObject()
      oos <<
accumulator.accumulate(args)
    }
  }
}
}

```

This example was inspired by this [Ruby example](#).

## Singleton Pattern

The [Singleton Pattern](#) is used to make sure only one object of a particular class is ever created. This can be useful when exactly one object is needed to coordinate actions across a system; perhaps for efficiency where creating lots of identical objects would be wasteful, perhaps because a particular algorithm needing a single point of control is required or perhaps when an object is used to interact with a non-shareable resource.

Weaknesses of the Singleton pattern include:

- It can reduce reuse. For instance, there are issues if you want to use inheritance with Singletons. If `SingletonB` extends `SingletonA`, should there be exactly (at most) one instance of each or should the creation of an object from one of the classes prohibit creation from the other. Also, if you decide both classes can have

an instance, how do you override the `getInstance()` method which is static?

- It is also hard to test singletons in general because of the static methods but Groovy can support that if required.

**Example: The Classic Java Singleton**

Suppose we wish to create a class for collecting votes. Because getting the right number of votes may be very important, we decide to use the singleton pattern. There will only ever be one `VoteCollector` object, so it makes it easier for us to reason about that objects creation and use.

```
class VoteCollector {  
    def votes = 0  
    private static final INSTANCE = new  
VoteCollector()  
    static getInstance(){ return INSTANCE }  
    private VoteCollector() {}  
    def display() { println  
"Collector:${hashCode()}, Votes:$votes" }  
}
```

Some points of interest about this code:

- it has a private constructor, so no `VoteCollector` objects can be created in our system (except for the `INSTANCE` we create)
- the `INSTANCE` is also private, so it can't be changed once set
- we haven't made the updating of votes thread-safe at this point (it doesn't add to this example)
- the vote collector instance is not lazily created (if we never reference the class, the instance won't be created; however, as soon as we reference the class, the instance will be created even if not needed initially)

We can use this singleton class in some script code as follows:

```

def collector = VoteCollector.instance
collector.display()
collector.votes++
collector = null

Thread.start{
    def collector2 = VoteCollector.instance
    collector2.display()
    collector2.votes++
    collector2 = null
}.join()

def collector3 = VoteCollector.instance
collector3.display()

```

Here we used the instance 3 times. The second usage was even in a different thread (but don't try this in a scenario with a new class loader).

Running this script yields (your hashcode value will vary):

```

Collector:15959960, Votes:0
Collector:15959960, Votes:1
Collector:15959960, Votes:2

```

Variations to this pattern:

- To support lazy-loading and multi-threading, we could just use the `synchronized` keyword with the `getInstance()` method. This has a performance hit but will work.
- We can consider variations involving double-checked locking and the `volatile` keyword (for Java 5 and above), but see the limitations of this approach [here](#).

**Example: Singleton via MetaProgramming**

Groovy's meta-programming capabilities allow concepts like the singleton pattern to be enacted in a far more fundamental way. This example illustrates a simple way to use Groovy's meta-programming capabilities to achieve the singleton pattern but not necessarily the most efficient way.

Suppose we want to keep track of the total number of calculations that a calculator performs. One way to do that is to use a singleton for the calculator class and keep a variable in the class with the count.

First we define some base classes. A `Calculator` class which performs calculations and records how many such calculations it performs and a `Client` class which acts as a facade to the calculator.

```
class Calculator {
    private total = 0
    def add(a, b) { total++; a + b }
    def getTotalCalculations() { 'Total
Calculations: ' + total }
    String toString() { 'Calc: ' +
hashCode()}
}

class Client {
    def calc = new Calculator()
    def executeCalc(a, b) { calc.add(a, b) }
    String toString() { 'Client: ' +
hashCode()}
}
```

Now we can define and register a *MetaClass* which intercepts all attempts to create a `Calculator` object and always provides a pre-created instance instead. We also register this *MetaClass* with the Groovy system:

```

class CalculatorMetaClass extends
MetaClassImpl {
    private final static INSTANCE = new
Calculator()
    CalculatorMetaClass() {
super(Calculator) }
    def invokeConstructor(Object[]
arguments) { return INSTANCE }
}

def registry =
GroovySystem.metaClassRegistry
registry.setMetaClass(Calculator, new
CalculatorMetaClass())

```

Now we use instances of our `Client` class from within a script. The client class will attempt to create new instances of the calculator but will always get the singleton.

```

def client = new Client()
assert 3 == client.executeCalc(1, 2)
println "$client, $client.calc,
$client.calc.totalCalculations"

client = new Client()
assert 4 == client.executeCalc(2, 2)
println "$client, $client.calc,
$client.calc.totalCalculations"

```

Here is the result of running this script (your hashcode values may vary):

```
Client: 7306473, Calc: 24230857, Total  
Calculations: 1  
Client: 31436753, Calc: 24230857, Total  
Calculations: 2
```

#### Guice Example

We can also implement the Singleton Pattern using [Guice](#). This example relies on annotations. Annotations are a Groovy 1.1 feature and will need to be run on a Java 5 or above JVM.

Consider the Calculator example again.

Guice is a Java-oriented framework that supports Interface-Oriented design. Hence we create a `Calculator` interface first. We can then create our `CalculatorImpl` implementation and a `Client` object which our script will interact with. The `Client` class isn't strictly needed for this example but allows us to show that non-singleton instances are the default. Here is the code:

```
// require(groupId: 'aopalliance',  
artifactId: 'aopalliance', version: '1.0')  
// require(groupId: 'com.google.code.guice',  
artifactId: 'guice', version: '1.0')  
  
import com.google.inject.*  
  
interface Calculator {  
    def add(a, b)  
}  
  
class CalculatorImpl implements Calculator {  
    private total = 0  
    def add(a, b) { total++; a + b }  
    def getTotalCalculations() { 'Total  
Calculations: ' + total }  
    String toString() { 'Calc: ' +
```

```
hashCode() }  
}
```

```
class Client {  
    @Inject Calculator calc  
    def executeCalc(a, b) { calc.add(a, b) }  
    String toString() { 'Client: ' +  
hashCode() }  
}
```

```
def injector = Guice.createInjector (  
    [configure: { binding ->  
        binding.bind(Calculator)  
            .to(CalculatorImpl)  
            .asEagerSingleton() } ] as
```

```
Module  
)
```

```
client = injector.getInstance(Client)  
assert 3 == client.executeCalc(1, 2)  
println "$client, $client.calc,  
$client.calc.totalCalculations"
```

```
client = injector.getInstance(Client)
```

```
assert 4 == client.executeCalc(2, 2)
println "$client, $client.calc,
$client.calc.totalCalculations"
```

Note the `@Inject` annotation in the `Client` class. We can always tell right in the source code which fields will be injected.

In this example we chose to use an *explicit* binding. All of our dependencies (ok, only one in this example at the moment) are configured in the binding. The Guice injector knows about the binding and injects the dependencies as required when we create objects. For the singleton pattern to hold, you must always use Guice to create your instances. Nothing shown so far would stop you creating another instance of the calculator manually using `new CalculatorImpl()` which would of course violate the desired singleton behaviour.

In other scenarios (though probably not in large systems), we could choose to express dependencies using annotations, such as the following example shows:

```

import com.google.inject.*

@ImplementedBy(CalculatorImpl)
interface Calculator {
    // as before ...
}

@Singleton
class CalculatorImpl implements Calculator {
    // as before ...
}

class Client {
    // as before ...
}

def injector = Guice.createInjector()

// ...

```

Note the `@Singleton` annotation on the `CalculatorImpl` class and the `@ImplementedBy` annotation in the `Calculator` interface.

When run, the above example (using either approach) yields (your hashcode values will vary):

```

Client: 8897128, Calc: 17431955, Total
Calculations: 1
Client: 21145613, Calc: 17431955, Total
Calculations: 2

```

You can see that we obtained a new client object whenever we asked for an instance but it was injected with the

same calculator object.

#### Spring Example

We can do the Calculator example again using Spring as follows:

```
// require(groupId:'org.springframework',
artifactId:'spring-core', version:'2.1m1')
// require(groupId:'org.springframework',
artifactId:'spring-beans', version:'2.1m1')

import
org.springframework.beans.factory.support.*

interface Calculator {
    def add(a, b)
}

class CalculatorImpl implements Calculator {
    private total = 0
    def add(a, b) { total++; a + b }
    def getTotalCalculations() { 'Total
Calculations: ' + total }
    String toString() { 'Calc: ' +
hashCode()}
}

class Client {
    Client(Calculator calc) { this.calc =
calc }
    def calc
    def executeCalc(a, b) { calc.add(a, b) }
    String toString() { 'Client: ' +
```

```
hashCode() }
}

// Here we 'wire' up our dependencies
through the API. Alternatively,
// we could use XML-based configuration or
the Grails Bean Builder DSL.
def factory = new
DefaultListableBeanFactory()
factory.registerBeanDefinition('calc', new
RootBeanDefinition(CalculatorImpl))
def beanDef = new RootBeanDefinition(Client,
false)
beanDef.setAutowireMode(AbstractBeanDefiniti
on.AUTOWIRE_AUTODETECT)
factory.registerBeanDefinition('client',
beanDef)

client = factory.getBean('client')
assert 3 == client.executeCalc(1, 2)
println "$client, $client.calc,
$client.calc.totalCalculations"

client = factory.getBean('client')
```

```
assert 4 == client.executeCalc(2, 2)
println "$client, $client.calc,
$client.calc.totalCalculations"
```

And here is the result (your hashcode values will vary):

```
Client: 29418586, Calc: 10580099, Total
Calculations: 1
Client: 14800362, Calc: 10580099, Total
Calculations: 2
```

#### Further information

- [Simply Singleton](#)
- [Use your singletons wisely](#)
- [Double-checked locking and the Singleton pattern](#)
- [Lazy Loading Singletons](#)
- [Implementing the Singleton Pattern in C#](#)

#### State Pattern

The [State Pattern](#) provides a structured approach to partitioning the behaviour within complex systems. The overall behaviour of a system is partitioned into well-defined states. Typically, each state is implemented by a class. The overall system behaviour can be determined firstly by knowing the *current state* of the system; secondly, by understanding the behaviour possible while in that state (as embodied in the methods of the class corresponding to that state).

#### Example

Here is an example:

```
class Client {
    def context = new Context()
    def connect() {
        context.state.connect()
    }
    def disconnect() {
        context.state.disconnect()
    }
}
```

```
def send_message(message) {
    context.state.send_message(message)
}
def receive_message() {
    context.state.receive_message()
}
}
```

```
class Context {
    def state = new Offline(this)
}
```

```
class ClientState {
    def context
    ClientState(context) {
        this.context = context
        inform()
    }
}
```

```
class Offline extends ClientState {
    Offline(context) {
        super(context)
    }
    def inform() {
        println "offline"
    }
    def connect() {
        context.state = new Online(context)
    }
    def disconnect() {
```

```
        println "error: not connected"
    }
    def send_message(message) {
        println "error: not connected"
    }
    def receive_message() {
        println "error: not connected"
    }
}

class Online extends ClientState {
    Online(context) {
        super(context)
    }
    def inform() {
        println "connected"
    }
    def connect() {
        println "error: already connected"
    }
    def disconnect() {
        context.state = new Offline(context)
    }
    def send_message(message) {
        println "\"$message\" sent"
    }
    def receive_message() {
        println "message received"
    }
}
```

```
client = new Client()  
client.send_message("Hello")  
client.connect()  
client.send_message("Hello")
```

```
client.connect()  
client.receive_message()  
client.disconnect()
```

Here is the output:

```
offline  
error: not connected  
connected  
"Hello" sent  
error: already connected  
message received  
offline
```

This example was inspired from a similar [Ruby Example](#). One of the great things about a dynamic language like Groovy though is that we can take this example and express it in many different ways depending on our particular needs. Some potential variations for this example are shown below.

#### Variation 1: Leveraging Interface-Oriented Design

One approach we could take is to leverage [Interface-Oriented Design](#). To do this, we could introduce the following interface:

```
interface State {  
    def connect()  
    def disconnect()  
    def send_message(message)  
    def receive_message()  
}
```

Then our `Client`, `Online` and `Offline` classes could be modified to implement that interface, e.g.:

```
class Client implements State {  
    // ... as before ...  
}
```

```
class Online implements State {  
    // ... as before ...  
}
```

You might ask: Haven't we just introduced additional boilerplate code? Can't we rely on duck-typing for this? The answer is 'yes' and 'no'. We can get away with duck-typing but one of the key intentions of the state pattern is to partition complexity. If we know that the *client* class and each *state* class all satisfy one interface, then we have placed some key boundaries around the complexity. We can look at any state class in isolation and know the bounds of behaviour possible for that state.

We don't have to use interfaces for this, but it helps express the intent of this particular style of partitioning and it helps reduce the size of our unit tests (we would have to have additional tests in place to express this intent in languages which have less support for interface-oriented design).

#### Variation 2: Extract State Pattern Logic

Alternatively, or in combination with other variations, we might decide to extract some of our State Pattern logic into helper classes. For example, we could define the following classes in a state pattern package/jar/script:

```
abstract class InstanceProvider {  
    static def registry =  
GroovySystem.metaClassRegistry  
    static def create(objectClass, param) {  
  
registry.getMetaClass(objectClass).invokeCon  
structor([param] as Object[])  
    }  
}  
  
abstract class Context {  
    private context  
    protected setContext(context) {
```

```
        this.context = context
    }
    def invokeMethod(String name, Object
arg) {
        context.invokeMethod(name, arg)
    }
    def startFrom(initialState) {

setContext(InstanceProvider.create(initialSt
ate, this))
    }
}

abstract class State {
    private client

    State(client) { this.client = client }

    def transitionTo(nextState) {

client.setContext(InstanceProvider.create(ne
```

```
xtState, client))
    }
}
```

This is all quite generic and can be used wherever we want to introduce the state pattern. Here is what our code would look like now:

```
class Client extends Context {
  Client() {
    startFrom(Offline)
  }
}

class Offline extends State {
  Offline(client) {
    super(client)
    println "offline"
  }
  def connect() {
    transitionTo(Online)
  }
  def disconnect() {
    println "error: not connected"
  }
  def send_message(message) {
    println "error: not connected"
  }
  def receive_message() {
    println "error: not connected"
  }
}
```

```
}
```

```
class Online extends State {  
  Online(client) {  
    super(client)  
    println "connected"  
  }  
  def connect(){  
    println "error: already connected"  
  }  
  def disconnect(){  
    transitionTo(Offline)  
  }  
  def send_message(message) {  
    println "\"$message\" sent"  
  }  
  def receive_message(){  
    println "message received"  
  }  
}
```

```
client = new Client()  
client.send_message("Hello")  
client.connect()  
client.send_message("Hello")
```

```
client.connect()  
client.receive_message()  
client.disconnect()
```

You can see here the `startFrom` and `transitionTo` methods begin to give our example code a DSL feel.

#### Variation 3: Bring on the DSL

Alternatively, or in combination with other variations, we might decide to fully embrace a Domain Specific Language (DSL) approach to this example.

We can define the following generic helper functions (first discussed [here](#)):

```
class Grammar {  
  def fsm  
  
  def event  
  def fromState  
  def toState  
  
  Grammar(a_fsm) {  
    fsm = a_fsm  
  }  
  
  def on(a_event) {  
    event = a_event  
    this  
  }  
  
  def on(a_event, a_transitioner) {  
    on(a_event)  
    a_transitioner.delegate = this  
    a_transitioner.call()  
  }
```

```
        this
    }

    def from(a_fromState) {
        fromState = a_fromState
        this
    }

    def to(a_toState) {
        assert a_toState, "Invalid toState:
$a_toState"
        toState = a_toState
        fsm.registerTransition(this)
        this
    }

    def isValid() {
        event && fromState && toState
    }

    public String toString() {
```

```
        "$event: $fromState=>$toState"  
    }  
}
```

```
class FiniteStateMachine {  
    def transitions = [:]  
  
    def initialState  
    def currentState  
  
    FiniteStateMachine(a_initialState) {  
        assert a_initialState, "You need to  
provide an initial state"  
        initialState = a_initialState  
        currentState = a_initialState  
    }  
  
    def record() {  
        Grammar.newInstance(this)  
    }  
  
    def reset() {  
        currentState = initialState  
    }  
  
    def isState(a_state) {  
        currentState == a_state  
    }  
}
```

```

def registerTransition(a_grammar) {
    assert a_grammar.isValid(), "Invalid
transition ($a_grammar)"
    def transition
    def event = a_grammar.event
    def fromState = a_grammar.fromState
    def toState = a_grammar.toState

    if (!transitions[event]) {
        transitions[event] = [:]
    }

    transition = transitions[event]
    assert !transition[fromState],
"Duplicate fromState $fromState for
transition $a_grammar"
    transition[fromState] = toState
}

def fire(a_event) {
    assert currentState, "Invalid
current state '$currentState': passed into
constructor"
    assert
transitions.containsKey(a_event), "Invalid
event '$a_event', should be one of
${transitions.keySet()}"
    def transition =
transitions[a_event]
    def nextState =
transition[currentState]

```

```
    assert nextState, "There is no  
transition from '$currentState' to any other  
state"
```

```
    currentState = nextState
```

```
        currentState
    }
}
```

Now we can define and test our state machine like this:

```
class StatePatternDslTest extends
GroovyTestCase {
    private fsm

    protected void setUp() {
        fsm =
FiniteStateMachine.newInstance('offline')
        def recorder = fsm.record()

recorder.on('connect').from('offline').to('o
nline')

recorder.on('disconnect').from('online').to(
'offline')

recorder.on('send_message').from('online').t
o('online')

recorder.on('receive_message').from('online'
).to('online')
    }

    void testInitialState() {
        assert fsm.isState('offline')
    }
}
```

```
}

void testOfflineState() {
    shouldFail{
        fsm.fire('send_message')
    }
    shouldFail{
        fsm.fire('receive_message')
    }
    shouldFail{
        fsm.fire('disconnect')
    }
    assert 'online' ==
fsm.fire('connect')
}
```

```
void testOnlineState() {
    fsm.fire('connect')
    fsm.fire('send_message')
    fsm.fire('receive_message')
    shouldFail{
        fsm.fire('connect')
    }
    assert 'offline' ==
```

```

fsm.fire('disconnect')
    }
}

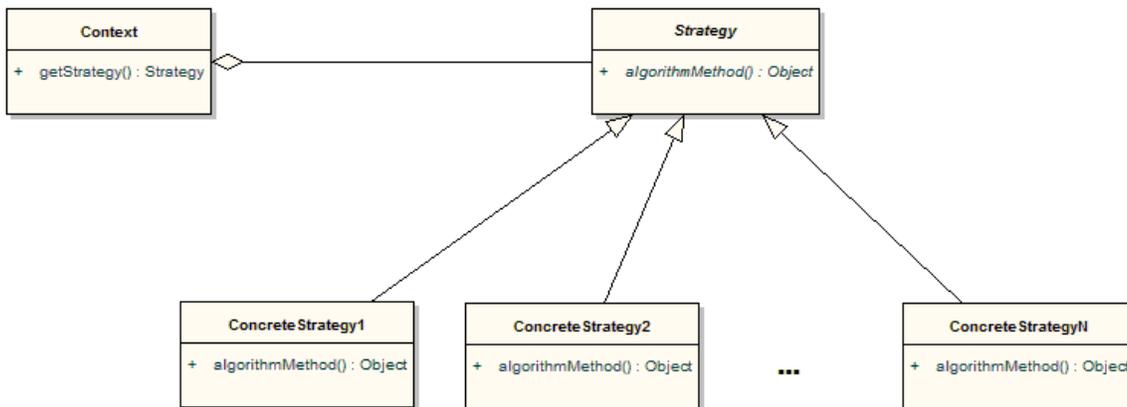
```

This example isn't an exact equivalent of the others. It doesn't use predefined `Online` and `Offline` classes. Instead it defines the entire state machine on the fly as needed. See the [previous reference](#) for more elaborate examples of this style.

See also: [Model-based testing using ModelJUnit](#)

## Strategy Pattern

The [Strategy Pattern](#) allows you to abstract away particular algorithms from their usage. This allows you to easily swap the algorithm being used without having to change the calling code. The general form of the pattern is:



In Groovy, because of its ability to treat code as a first class object using anonymous methods (which we loosely call *Closures*), the need for the strategy pattern is greatly reduced. You can simply place algorithms inside Closures.

### Example

First let's look at the traditional way of encapsulating the Strategy Pattern.

```

interface Calc {
    def execute(n, m)
}

class CalcByMult implements Calc {
    def execute(n, m) { n * m }
}

```

```
class CalcByManyAdds implements Calc {
  def execute(n, m) {
    def result = 0
    n.times{
      result += m
    }
    return result
  }
}

def sampleData = [
  [3, 4, 12],
  [5, -5, -25]
]

Calc[] multiplicationStrategies = [
  new CalcByMult(),
  new CalcByManyAdds()
]

sampleData.each{ data ->
  multiplicationStrategies.each{ calc ->
    assert data[2] ==
```

```

calc.execute(data[0], data[1])
    }
}

```

Here we have defined an interface `Calc` which our concrete strategy classes will implement (we could also have used an abstract class). We then defined two algorithms for doing simple multiplication: `CalcByMult` the normal way, and `CalcByManyAdds` using only addition (don't try this one using negative numbers - yes we could fix this but it would just make the example longer). We then use normal [polymorphism](#) to invoke the algorithms.

Here is the Groovier way to achieve the same thing using Closures:

```

def multiplicationStrategies = [
    { n, m -> n * m },
    { n, m -> def result = 0; n.times{
result += m }; result }
]

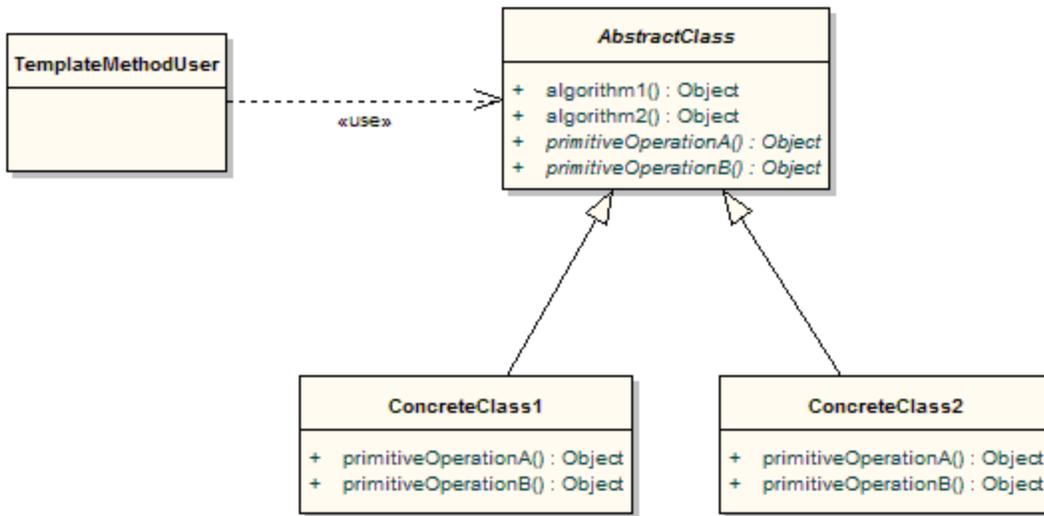
def sampleData = [
    [3, 4, 12],
    [5, -5, -25]
]

sampleData.each{ data ->
    multiplicationStrategies.each{ calc ->
        assert data[2] == calc(data[0],
data[1])
    }
}

```

## Template Method Pattern

The [Template Method Pattern](#) abstracts away the details of several algorithms. The generic part of an algorithm is contained within a base class. Particular implementation details are captured within base classes. The generic pattern of classes involved looks like this:



### Example

In this example, `Accumulator` captures the essence of the accumulation algorithm. The base classes `Sum` and `Product` provide particular customised ways to use the generic accumulation algorithm.

```

abstract class Accumulator {
    protected initial
    abstract doAccumulate(total, v)
    def accumulate(values) {
        def total = initial
        values.each { v -> total =
doAccumulate(total, v) }
        total
    }
}

class Sum extends Accumulator {
    def Sum() { initial = 0 }
    def doAccumulate(total, v) { total + v }
}

class Product extends Accumulator {
    def Product() { initial = 1 }
    def doAccumulate(total, v) { total * v }
}

println new Sum().accumulate([1,2,3,4])
println new Product().accumulate([1,2,3,4])

```

The resulting output is:

```

10
24

```

In this particular case, you could use Groovy's inject method to achieve a similar result using Closures:

```
Closure addAll = { total, item -> total +=
item }
def accumulated = [1, 2, 3, 4].inject(0,
addAll)
println accumulated // => 10
```

Thanks to duck-typing, this would also work with other objects which support an add (plus() in Groovy) method, e.g.:

```
accumulated = [ "1", "2", "3", "4"
].inject("", addAll)
println accumulated // => "1234"
```

We could also do the multiplication case as follows:

```
Closure multAll = { total, item -> total *=
item }
accumulated = [1, 2, 3, 4].inject(1,
multAll)
println accumulated // => 24
```

Using closures this way looks more like the [Strategy Pattern](#) but if we realise that the built-in `inject` method is the generic part of the algorithm for our template method, then the Closures become the customised parts of the template method pattern.

## Visitor Pattern

The [Visitor Pattern](#) is one of those well-known but not often used patterns. I think this is strange, as it is really a nice thing.

The goal of the pattern is to separate an algorithm from an object structure. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures.

### *Simple Example*

This example (inspired by [this](#)) considers how to calculate the bounds of shapes (or collections of shapes). Our first attempt uses the traditional visitor pattern. We will see a more Groovy way to do this shortly.

```

abstract class Shape { }

class Rectangle extends Shape {
    def x, y, width, height

    Rectangle(x, y, width, height) {
        this.x = x; this.y = y; this.width =
width; this.height = height
    }

    def union(rect) {
        if (!rect) return this
        def minx = [rect.x, x].min()
        def maxx = [rect.x + width, x +
width].max()
        def miny = [rect.y, y].min()
        def maxy = [rect.y + height, y +
height].max()
        new Rectangle(minx, miny, maxx -
minx, maxy - miny)
    }

    def accept(visitor) {
        visitor.visit_rectangle(this)
    }
}

class Line extends Shape {
    def x1, y1, x2, y2

```

```
    Line(x1, y1, x2, y2) {
        this.x1 = x1; this.y1 = y1; this.x2
= x2; this.y2 = y2
    }
```

```
    def accept(visitor){
        visitor.visit_line(this)
    }
}
```

```
class Group extends Shape {
    def shapes = []
    def add(shape) { shapes += shape }
    def remove(shape) { shapes -= shape }
    def accept(visitor) {
        visitor.visit_group(this)
    }
}
```

```
class BoundingRectangleVisitor {
    def bounds

    def visit_rectangle(rectangle) {
        if (bounds)
            bounds = bounds.union(rectangle)
        else
            bounds = rectangle
    }

    def visit_line(line) {
        def line_bounds = new
```

```

Rectangle(line.x1, line.y1, line.x2-line.y1,
line.x2-line.y2)
    if (bounds)
        bounds =
bounds.union(line_bounds)
    else
        bounds = line_bounds
}

def visit_group(group) {
    group.shapes.each { shape ->
shape.accept(this) }
}

def group = new Group()
group.add(new Rectangle(100, 40, 10, 5))
group.add(new Rectangle(100, 70, 10, 5))
group.add(new Line(90, 30, 60, 5))
def visitor = new BoundingRectangleVisitor()

```

```
group.accept(visitor)
bounding_box = visitor.bounds
println bounding_box.dump()
```

That took quite a bit of code.

We can improve the clarity of our code (and make it about half the size) by making use of Groovy Closures as follows:

```
abstract class Shape {
    def accept(Closure yield) { yield(this)
}
}

class Rectangle extends Shape {
    def x, y, w, h
    def bounds() { this }
    def union(rect) {
        if (!rect) return this
        def minx = [ rect.x, x ].min()
        def maxx = [ rect.x + w, x + w
].max()
        def miny = [ rect.y, y ].min()
        def maxy = [ rect.y + h, y + h
].max()
        new Rectangle(x:minx, y:miny, w:maxx
- minx, h:maxy - miny)
    }
}

class Line extends Shape {
    def x1, y1, x2, y2
```

```
def bounds() {  
    new Rectangle(x:x1, y:y1, w:x2-y1,  
h:x2-y2)  
    }  
}
```

```
class Group {  
    def shapes = []  
    def leftShift(shape) { shapes += shape }  
    def accept(Closure yield) {  
shapes.each{it.accept(yield)} }  
}
```

```
def group = new Group()  
group << new Rectangle(x:100, y:40, w:10,  
h:5)  
group << new Rectangle(x:100, y:70, w:10,  
h:5)  
group << new Line(x1:90, y1:30, x2:60, y2:5)  
def bounds
```

```
group.accept{ bounds =
it.bounds().union(bounds) }
println bounds.dump()
```

*Advanced Example*

```
interface Visitor {
    public void visit(NodeType1 n1);
    public void visit(NodeType2 n2);
}

interface Visitable {
    public void accept(Visitor visitor);
}

class NodeType1 implements Visitable {
    Visitable[] children = new
Visitable[0];
    public void accept(Visitor visitor) {
        visitor.visit(this);
        for(int i = 0; i < children.length;
++i) {
            children[i].accept(visitor);
        }
    }
}

class NodeType2 implements Visitable {
    Visitable[] children = new
Visitable[0];
```

```
public void accept(Visitor visitor) {
    visitor.visit(this);
    for(int i = 0; i < children.length;
++i) {
        children[i].accept(visitor);
    }
}
}
```

```
public class NodeType1Counter implements
Visitor {
    int count = 0;
    public void visit(NodeType1 n1) {
        count++;
    }
}
```

```
    }  
    public void visit(NodeType2 n2){}  
}
```

If we now use `NodeType1Counter` on a tree like this:

```
NodeType1 root = new NodeType1()  
root.children = new Visitable[2];  
root.children[0] = new NodeType1();  
root.children[1] = new NodeType2();
```

Then we have one `NodeType1` object as root and one of the children is also a `NodeType1` instance. The other child is a `NodeType2` instance. That means using `NodeType1Counter` here should count 2 `NodeType1` objects.

#### Why to use this

As you can see here very good we have a visitor that has a state while the tree of objects is not changed. That's pretty useful in different areas, for example you could have a visitor counting all node types, or how many different types are used, or you could use methods special to the node to gather information about the tree and much more.

#### What happens if we add a new type?

In this case we have to do much work.. we have to change `Visitor` to accept the new type, we have to write the new type itself of course and we have to change every `Visitor` we have already implemented. After very few changes you will modify all your `Visitors` to extend a default implementation of the visitor, so you don't need to change every `Visitor` each time you add a new type.

#### What if we want to have different iteration patterns?

Then you have a problem. since the node describes how to iterate, you have no influence and stop iteration at a point or change the order. so maybe we should change this a little to this:

```
interface Visitor {  
    public void visit(NodeType1 n1);  
    public void visit(NodeType2 n2);  
}
```

```
class DefaultVisitor implements Visitor{
    public void visit(NodeType1 n1) {
        for(int i = 0; i <
n1.children.length; ++i) {
            n1.children[i].accept(visitor);
        }
    }
    public void visit(NodeType2 n2) {
        for(int i = 0; i <
n2.children.length; ++i) {
            n2.children[i].accept(visitor);
        }
    }
}
```

```
interface Visitable {
    public void accept(Visitor visitor);
}
```

```
class NodeType1 implements Visitable {
    Visitable[] children = new
Visitable[0];
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }
}
```

```
class NodeType2 implements Visitable {
    Visitable[] children = new
Visitable[0];
```

```
        public void accept(Visitor visitor) {
            visitor.visit(this);
        }
    }
```

```
public class NodeType1Counter extends
DefaultVisitor {
    int count = 0;
    public void visit(NodeType1 n1) {
        count++;
    }
}
```

```
        super.visit(n1);
    }
}
```

Some small changes but with big effect... the visitor is now recursive and tells me how to iterate. The implementation in the Nodes is minimized to `visitor.visit(this);`, `DefaultVisitor` is now able to catch the new types, we can stop iteration by not delegating to `super`. Of course the big disadvantage now is that it is no longer iterative, but you can't get all the benefits.

#### **Make it Groovy**

The question now is how to make that a bit more Groovy. Didn't you find `visitor.visit(this);` strange? Why is it there? The answer is to simulate double dispatch. In Java the compile time type is used, so when I `visitor.visit(children[i]);` then the compiler won't be able to find the correct method, because `Visitor` does not contain a method `visit(Visitable)`. And even if it would, we would like to visit the more special methods with `NodeType1` or `NodeType2`.

Now Groovy is not using the static type, Groovy uses the runtime type. This means I could do `visitor.visit(children[i])` directly. Hmm.. since we minimized the `accept` method to just do the double dispatch part and since the runtime type system of Groovy will already cover that.. do we need the `accept` method? I think you can guess that I would answer no. But we can do more. We had the disadvantage of not knowing how to handle unknown tree elements. We had to extend the interface `Visitor` for that, resulting in changes to `DefaultVisitor` and then we have the task to provide a useful default like iterating the node or not doing anything at all. Now with Groovy we can catch that case by adding a `visit(Visitable)` method that does nothing. that would be the same in Java btw.

But don't let us stop here... do we need the `Visitor` interface? If we don't have the `accept` method, then we don't need the `Visitor` interface at all. So the new code would be:

```
class DefaultVisitor {
    void visit(NodeType1 n1) {
        n1.children.each { visit(it) }
    }
    void visit(NodeType2 n2) {
        n2.children.each { visit(it) }
    }
    void visit(Visitable v) {}
}
```

```
interface Visitable {}
```

```
class NodeType1 implements Visitable {
    Visitable[] children = []
}
```

```
class NodeType2 implements Visitable {
    Visitable[] children = []
}
```

```
public class NodeType1Counter extends
DefaultVisitor {
    int count = 0;
    public void visit(NodeType1 n1) {
        count++
        super.visit(n1)
    }
}
```

Looks like we saved a few lines of code here. But we made more. The Visitable nodes now do not refer to any Visitor class or interface. For me this is the best level of separation you could get here. But do we really need to stop here? No. Let us change the Visitable interface a little and let it return the children we want to visit next. This allows us a general iteration method.

```

class DefaultVisitor {
    void visit(Visitable v) {
        doIteration(v)
    }
    doIteraton(Visitable v) {
        v.children.each { visit(it) }
    }
}

interface Visitable {
    Visitable[] getChildren()
}

class NodeType1 implements Visitable {
    Visitable[] children = []
}

class NodeType2 implements Visitable {
    Visitable[] children = []
}

public class NodeType1Counter extends
DefaultVisitor {
    int count = 0
    public void visit(NodeType1 n1) {
        count++
        super.visit(n1)
    }
}

```

DefaultVisitor now looks a bit different. I added a doIteration method that will get the children it should iterate over and then call visit on each element. Per default this will call `visit(Visitable)` which then iterates over the children of this child. I changed Visitable to ensure that any node will be able to return children (even if empty). I didn't have to change the NodeType1 and NodeType2 class, because the way the children field was defined already made them a property, which means Groovy is so nice to generate a get method for us. No the really interesting part is NodeType1Counter, it is interesting because we have not changed it. `super.visit(n1)` will now call `visit(Visitable)` which will call `doIterate` which will start the next level of iteration. So no change. But `visit(it)` will call `visit(NodeType1)` if it is of type NodeType1. In fact we don't need the doIterate method, we could do that in `visit(Visitable)` too, but I thought this variant is better, because it allows us to write a new Visitor that overwrites `visit(Visitable)` for error cases which of course means we must not do `super.visit(n1)` but `doIterate(n1)`.

#### Summary

In the end we got ~40% less code, a robust and stable architecture and we completely removed the Visitor from the Visitable. I heard about visitor implementations based on Reflection to get a more generic version. Well, with this you see there is really no need to do such thing. If we add new types we don't need to change anything. It is said that the visitor pattern doesn't fit extreme programming techniques very well because you need to make changes to so many classes all the time. I think I proved that this is because of Java not because the pattern is bad or something.

There are variants of the Visitor pattern, like the acyclic visitor pattern, that tries to solve the problem of adding new node types with special visitors. I don't like that very much, it works with casts, catches the `ClassCastException` and other nasty things. In the end it tries to solve something we don't even get with the Groovy version.

One more thing. `NodeType1Counter` could be implemented in Java as well. Groovy will recognize the visit methods and call them as needed because `DefaultVisitor` is still Groovy and does all the magic.

#### Further Information

1. [Componentization: the Visitor example](#)

## Dynamic language beans in Spring

### Background

The [Spring Framework](#) is a leading full-stack Java/J2EE application framework. It is aimed primarily at Java projects and delivers significant benefits such as reducing development effort and costs while providing facilities to improve test coverage and quality. It also has mechanisms for allowing beans to be backed by dynamic language code including Groovy and other languages.

Let's have a look at how to use Spring's support for writing dynamic language backed beans to extend an existing Java application. The existing Java application prints out information about countries in sorted order. We want to be able to add new countries and new sorting algorithms but using different languages for each one. Our goal is to not have to change the original Java code but add the new features just through the application wiring.

### The Existing Application (Spring and Java)

Suppose we have the following interface:

```
package spring;
```

```
public interface Country {  
    long getPopulation();  
    String getCapital();  
}
```

And the following implementation class:

```

package spring;

public class USA implements Country {
    private final String capital;
    private final long population;

    public USA(String capital, long
population) {
        this.population = population;
        this.capital = capital;
    }

    public String toString() {
        return "USA[Capital=" + capital + ",
Population=" + population + " ]";
    }

    public long getPopulation() {
        return population;
    }

    public String getCapital() {
        return capital;
    }
}

```

Spring supports the [Dependency Injection](#) style of coding. This style encourages classes which depend on classes like USA not to hard-code that dependency, e.g. no fragments of code like 'new USA(...)' and no 'import ...USA;'. Such a style allows us to change the concrete implementations we depend on for testing purposes or at a future point in time as our program evolves. In our example above, we might declaratively state our dependencies in a beans.xml file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema
/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xmlns:lang="http://www.springframework.org/s
chema/lang"
      xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/
spring-beans-2.0.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/s
pring-lang-2.0.xsd">
  <bean id="usa" class="spring.USA">
    <constructor-arg value="Washington,
D.C." />
    <constructor-arg value="298444215" />
  </bean>
</beans>
```

In this example, the `<constructor-arg/>` element allows us to use constructor-based injection.

Having done this, we can get access to our beans through a variety of mechanisms. Normally, access to the beans is mostly transparent. In our case though, we are going to make access explicit so you can see what is going on. Our main method might look like this:

```
package spring;

import
org.springframework.context.ApplicationConte
xt;
import
org.springframework.context.support.ClassPat
hXmlApplicationContext;

import java.util.Arrays;

public class SortMain {
    public static void main(String[] args) {
        try {
            ApplicationContext ctx = new
ClassPathXmlApplicationContext("beans.xml");
            Country usa = (Country)
ctx.getBean("usa");
            System.out.println("USA Info:\n"
+ usa);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Running this results in:

**USA Info:**

**USA[Capital=Washington, D.C.,  
Population=298444215]**

*Spring and Groovy*

We can extend this example and introduce Groovy in a number of ways. Firstly, we can create the following Groovy class:

```
package spring  
public class Australia implements Country {  
    String capital  
    long population  
    String toString() {  
        return "Australia[Capital=" +  
capital + ", Population=" + population + "]"  
    }  
}
```

And this one too:

```
package spring  
public class NewZealand extends Australia  
implements Country {  
    String toString() {  
        return "NewZealand[Capital=" +  
capital + ", Population=" + population + "]"  
    }  
}
```

So long as the corresponding `.class` file is available on the classpath (e.g. by your IDE or by manually running `gro`

ovyc), we can simply reference this class in our beans.xml file. E.g. for the first class we can use:

```
...
<bean id="country1"
class="spring.Australia">
    <property name="capital"
value="Canberra" />
    <property name="population"
value="20264082" />
</bean>
...
```

Alternatively, if the source file is on the classpath, we can use special Spring notation to reference it, e.g. for the second class we can use:

```
...
<lang:groovy id="country3"
script-source="classpath:spring/NewZealand.g
roovy">
    <lang:property name="capital"
value="Wellington" />
    <lang:property name="population"
value="4076140" />
</lang:groovy>
...
```

In these examples, the <property/> and <lang:property/> elements allows us to use setter-based injection.

### ***Spring and JRuby***

If we prefer to code in another language (with a few restrictions - see below), Spring supports other languages too, e.g.:

```
require 'java'

include_class 'spring.Country'

class Fiji < Country
  def getCapital()
    @capital
  end
  def getPopulation()
    @population
  end
  def setCapital(capital)
    @capital = capital
  end
  def setPopulation(population)
    @population = population
  end
  def to_s()
    "Fiji[Capital=" + @capital + ",
Population=" + getPopulation().to_s() + "]"
  end
end
Fiji.new()
```

```
...
<lang:jruby id="country4"
script-interfaces="spring.Country"
script-source="classpath:spring/Fiji.rb">
    <lang:property name="capital"
value="Suva" />
    <lang:property name="population"
value="905949" />
</lang:jruby>
...
```

### *Spring and Groovy Again*

But wait there's more ...

Suppose now that we wish to sort our countries according to population size. We don't want to use Java's built-in sort mechanisms as some of them rely on our objects implementing the `Comparable` interface and we don't want that noise in our Ruby script. Instead we will use Groovy. We could simply write a `Sort` class in Groovy and reference as we have done above. This time however we are going to use an additional Spring feature and have the scripting code within our `beans.xml` file. First we define the following Java interface:

```
package spring;

import java.util.List;

public interface Sorter {
    List sort(Country[] unsorted);
}
```

We can then include the Groovy sort code directly into the `beans.xml` file as follows:

```

...
<lang:groovy id="sorter">
<lang:inline-script><![CDATA[
    package spring
    class CountrySorter implements Sorter {
        String order
        List sort(Country[] items) {
            List result =
items.toList().sort{ p1, p2 -> p1.population
<=> p2.population }
            if (order == "reverse") return
result.reverse() else return result
        }
    }
]]></lang:inline-script>
    <lang:property name="order"
value="forward" />
</lang:groovy>
...

```

### *Putting it all together*

We now combine all of the approaches above in a final example.

Here is the complete `beans.xml` file:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema
/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-

```

instance"

```
xmlns:lang="http://www.springframework.org/s  
chema/lang"
```

```
    xsi:schemaLocation="
```

```
http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/  
spring-beans-2.0.xsd
```

```
http://www.springframework.org/schema/lang
```

```
http://www.springframework.org/schema/lang/s  
pring-lang-2.0.xsd">
```

```
<bean id="country1"
```

```
class="spring.Australia">
```

```
    <property name="capital"
```

```
value="Canberra" />
```

```
    <property name="population"
```

```
value="20264082" />
```

```
</bean>
```

```
<bean id="country2" class="spring.USA">
```

```
    <constructor-arg value="Washington,  
D.C." />
```

```
    <constructor-arg value="298444215" />
```

```
</bean>
```

```
<lang:groovy id="country3"
```

```
script-source="classpath:spring/NewZealand.g  
roovy">
```

```
    <lang:property name="capital"
```

```
value="Wellington" />
```

```
    <lang:property name="population"
```

```
value="4076140" />
```

```
</lang:groovy>
```

```
<lang:jruby id="country4"
script-interfaces="spring.Country"
script-source="classpath:spring/Fiji.rb">
  <lang:property name="capital"
value="Suva" />
  <lang:property name="population"
value="905949" />
</lang:jruby>
<lang:groovy id="sorter">
<lang:inline-script><![CDATA[
  package spring
  class CountrySorter implements Sorter {
    String order
    List sort(Country[] items) {
      List result =
items.toList().sort{ p1, p2 -> p1.population
<=> p2.population }
      if (order == "reverse") return
result.reverse() else return result
    }
  }
]]></lang:inline-script>
  <lang:property name="order"
```

```
value="forward" />
</lang:groovy>
</beans>
```

Our Java code to use this example looks like:

```
package spring;

import java.util.Arrays;
import
org.springframework.context.ApplicationConte
xt;
import
org.springframework.context.support.ClassPat
hXmlApplicationContext;

public class SortMain {
    public static void main(String[] args) {
        try {
            ApplicationContext ctx = new
ClassPathXmlApplicationContext("beans.xml");
            Country[] countries = {
                (Country)
ctx.getBean("country1"), (Country)
ctx.getBean("country2"),
                (Country)
ctx.getBean("country3"), (Country)
ctx.getBean("country4")
            };
            Sorter sorter = (Sorter)
ctx.getBean("sorter");
```

```
        System.out.println("Unsorted:\n"
+ Arrays.asList(countries));
        System.out.println("Sorted:\n" +
sorter.sort(countries));
    } catch (Exception e) {
        e.printStackTrace();
    }
```

```
    }  
  }  
}
```

And the resulting output (with a little bit of hand formatting) looks like:

**Unsorted:**

```
[Australia[Capital=Canberra,  
Population=20264082],  
  USA[Capital=Washington, D.C.,  
Population=298444215],  
  NewZealand[Capital=Wellington,  
Population=4076140],  
  JRuby object [Fiji[Capital=Suva,  
Population=905949]]]
```

**Sorted:**

```
[JRuby object [Fiji[Capital=Suva,  
Population=905949]],  
  NewZealand[Capital=Wellington,  
Population=4076140],  
  Australia[Capital=Canberra,  
Population=20264082],  
  USA[Capital=Washington, D.C.,  
Population=298444215]]]
```

**What we didn't tell you yet**

- Spring supports [BeanShell](#) in addition to JRuby and Groovy
- Spring supports the concept of *refreshable* beans when using the `<lang:language/>` element so that if your bean source code changes, the bean will be reloaded (see GINA or the Spring doco for more details)
- The Groovy scripting examples in the current Spring documentation are based on an old version of Groovy, ignore the `@Property` keywords and use the latest `groovy-all-xxx.jar` instead of the jars they recommend

- The `<lang: language/>` element currently only supports setter-based injection
- Spring automatically converts between the object models of the various languages but there are some limitations (particularly with JRuby - see the next section)

### **Current Limitations**

Currently using the Groovy language through Spring is extensively supported. Using other languages like JRuby takes a little more care. Try restricting your JRuby methods to ones which take and return simple data types, e.g. long and String. You may also find that certain operations don't work as you'd expect when working between Ruby and other languages, e.g. if you defined a `compareTo` method in our `Fi.ji.rb` file, it would return `long` by default rather than the `int` which Java is expecting. In addition, the `compareTo` method takes an `other` object as a parameter. Currently the wrapping of this `other` object from Java or Groovy into a Ruby object hides the original Java methods.

### **Further Information**

- Section 11.5 of [GINA](#)
- Spring [documentation](#) for [scripting](#)
- The Spring example in [Groovy and JMX](#)

## **Embedding Groovy**

Groovy is a great language just on its own in various scenarios. It is also extremely useful in mixed Groovy/Java environments. With this in mind, Groovy has been designed to be very lightweight and easy to embed into any Java application system.

There are three main approaches for natively integrating Groovy with Java. Each of these is discussed in more detail below.

Alternatively, you can use the [Bean Scripting Framework](#) to embed any scripting language into your Java code, giving you other language options if you needed them (though we can't imagine why 😊). Using BSF allows you to be more loosely coupled to your scripting language; however, native integration is more light weight and offers closer integration.

### **Evaluate scripts or expressions using the shell**

You can evaluate any expression or script in Groovy using the [GroovyShell](#). The GroovyShell allows you to pass in and out variables via the [Binding](#) object.

```
// call groovy expressions from Java code
Binding binding = new Binding();
binding.setVariable("foo", new Integer(2));
GroovyShell shell = new
GroovyShell(binding);

Object value = shell.evaluate("println
'Hello World!'; x = 123; return foo * 10");
assert value.equals(new Integer(20));
assert binding.getVariable("x").equals(new
Integer(123));
```

#### Evaluate Scripts with a Common Base Class

It is often useful to make your groovy scripts extend a base class of your choosing so that the script can access common methods. This can be achieved by setting the script base class property on the compile configuration and passing this new compiler configuration to the shell.

```

class ScriptBaseTest {

    @Test
    void extend_groovy_script() {
        def compiler = new
CompilerConfiguration()

        compiler.setScriptBaseClass("ScriptBaseTestS
cript")

        def shell = new
GroovyShell(this.class.classLoader, new
Binding(), compiler)

        assertEquals shell.evaluate("foo()"),
"this is foo"
    }
}

abstract class ScriptBaseTestScript extends
Script {
    def foo() {
        "this is foo"
    }
}

```

### Dynamically loading and running Groovy code inside Java

You can use the [GroovyClassLoader](#) to load classes dynamically into a Java program and execute them (or use them) directly. The following Java code shows an example:

```
ClassLoader parent =
getClass().getClassLoader();
GroovyClassLoader loader = new
GroovyClassLoader(parent);
Class groovyClass = loader.parseClass(new
File("src/test/groovy/script/HelloWorld.groovy"));

// let's call some method on an instance
GroovyObject groovyObject = (GroovyObject)
groovyClass.newInstance();
Object[] args = {};
groovyObject.invokeMethod("run", args);
```

If you have an interface you wish to use which you implement in the Groovy script you can use it as follows:

```
GroovyClassLoader gcl = new
GroovyClassLoader();
Class clazz =
gcl.parseClass(myStringwithGroovyClassSource
, "SomeName.groovy");
Object aScript = clazz.newInstance();
MyInterface myObject = (MyInterface)
aScript;
myObject.interfaceMethod();
...
```

This works fine if the Groovy class implements the interface MyInterface. myObject can from then on be used as every other Java object implementing MyInterface.

One thing to remember is that the parseClass will try to create an object from your String fileName. Another way to

do the gcl.parseClass is:

```
Class clazz = gcl.parseClass(new  
File("SomeName.groovy"));
```

Full Example:

```
TestInterface.java  
public interface TestInterface {  
    public void printIt();  
}  
  
Tester.groovy  
public class Tester implements TestInterface  
{  
    public void printIt() {  
        println "this is in the test class";  
    }  
}  
  
TestClass.java -- inside of a method  
String fileName = "Tester.groovy";  
GroovyClassLoader gcl = new  
GroovyClassLoader();  
Class clazz = gcl.parseClass(new  
File(fileName));  
Object aScript = clazz.newInstance();  
  
TestInterface ifc = (TestInterface) aScript;  
ifc.printIt();
```

Note that all of the error handling has been removed -- you won't be able to do this in a java class. I actually use the Interface invocation for Groovy inside of a Utility Class.

## The GroovyScriptEngine

The most complete solution for people who want to embed groovy scripts into their servers and have them reloaded on modification is the GroovyScriptEngine. You initialize the GroovyScriptEngine with a set of CLASSPATH like roots that can be URLs or directory names. You can then execute any Groovy script within those roots. The GSE will also track dependencies between scripts so that if any dependent script is modified the whole tree will be recompiled and reloaded.

Additionally, each time you run a script you can pass in a Binding that contains properties that the script can access. Any properties set in the script will also be available in that binding after the script has run. Here is a simple example:

/my/groovy/script/path/hello.groovy:

```
output = "Hello, ${input}!"
```

```
import groovy.lang.Binding;  
import groovy.util.GroovyScriptEngine;  
  
String[] roots = new String[] {  
    "/my/groovy/script/path" };  
GroovyScriptEngine gse = new  
GroovyScriptEngine(roots);  
Binding binding = new Binding();  
binding.setVariable("input", "world");  
gse.run("hello.groovy", binding);  
System.out.println(binding.getVariable("outp  
ut"));
```

This will print "Hello, world!".

## Embedding a Groovy Console in a Java Application

An interactive Groovy interpreter can be embedded in a running application for experimentation and testing. For a tutorial on how to do, including example code, see the cookbook example [Embedding a Groovy Console in a Java Server Application](#).

An example for the integration of Groovy as scripting language into an application can be found at [Integrating Groovy in an application - a success story](#)

## Runtime dependencies

As well as Java 1.4 and the Groovy jar we also depend at runtime on the ASM library constituted of five jars (asm-2.2.jar, asm-attrs-2.2.jar, asm-analysis-2.2, asm-tree-2.2.jar, and asm-util-2.2.jar) plus the ANTLR library (antlr-2.7.5.jar). That's it. So just add these 7 jars to your classpath and away you go, you can happily embed Groovy into your application.

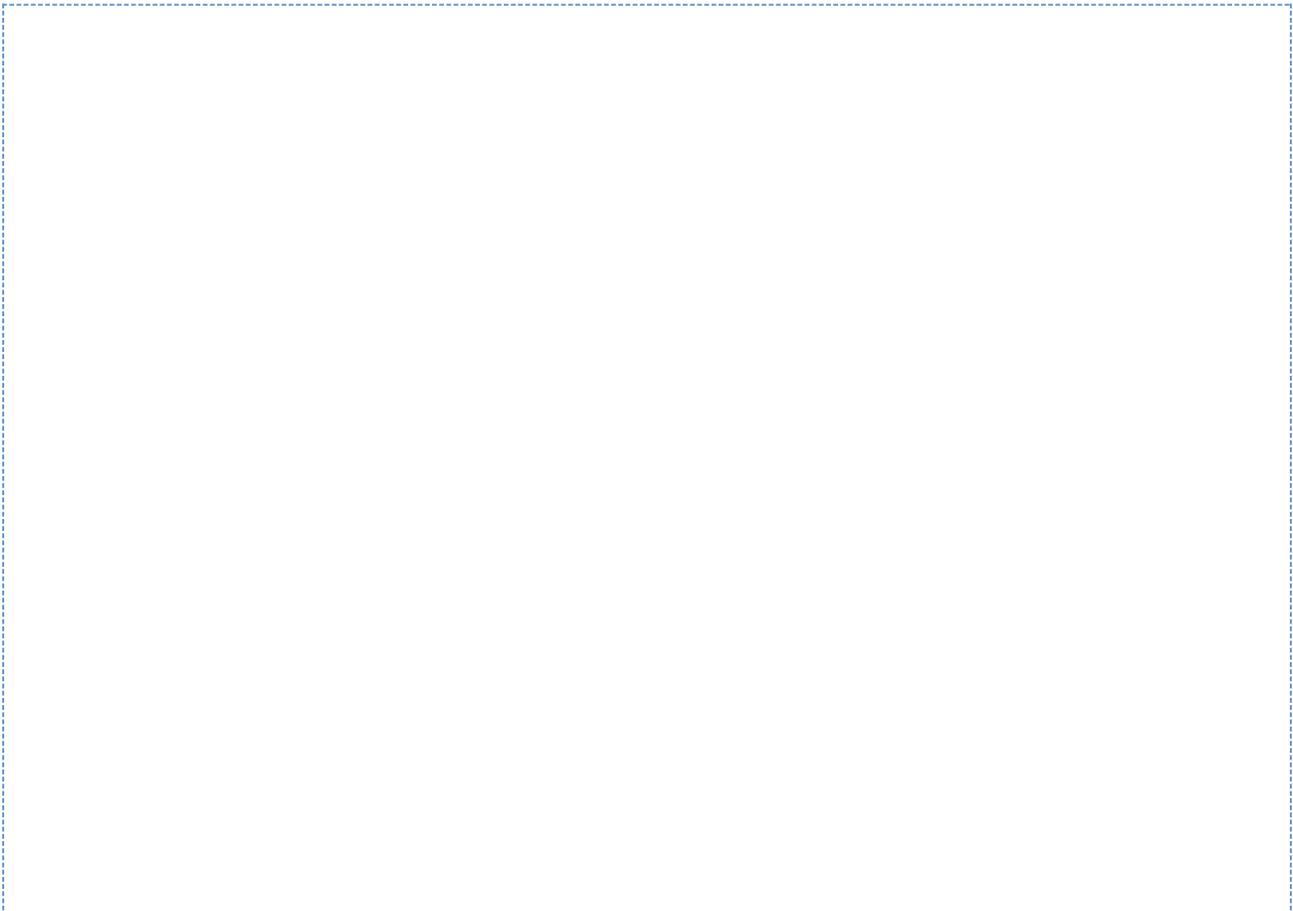
Alternatively, instead of several jars, you can use groovy-all-1.0-beta-x.jar included in the GROOVY\_HOME/embeddable directory of your distribution: this jar contains both Groovy and ASM combined in a single and convenient archive, with the ASM classes in a different namespace, so conflicts with other libraries also using ASM will be avoided 😊

## Mixed Java and Groovy Applications

This example looks at the issues surrounding a mixed Java/Groovy application. This issue only arises when there is mutual dependencies between your mixed language source files. So, if part of your system is pure Java for instance, you won't have this problem. You would just compile that part of your system first and reference the resulting class/jar file(s) from the part of your system that was written in Groovy.

## The legacy version of our application

Suppose you have an initial application written in Java. It deals with packages sent through the post. It has the following `Postpack` class (you would typically need some additional fields but we have simplified our domain to keep the example simple):



```
package v1;

public class Postpack implements Comparable
{
    private final int weight;
    private final int zip;

    public int getWeight() {
        return weight;
    }

    public int getZip() {
        return zip;
    }

    public String toString() {
        return "Postpack[Weight=" + weight +
", Zip=" + zip + " ]";
    }

    public int compareTo(Object o) {
        Postpack other = (Postpack) o;
        return zip - other.getZip();
    }

    public Postpack(int weight, int zip) {
        this.weight = weight;
        this.zip = zip;
    }
}
```

Now suppose you also have a sort helper class `ZipSorter` as follows:

```
package v1;

import java.util.List;
import java.util.Collections;

public class ZipSorter {
    public List sort(List items) {
        Collections.sort(items);
        return items;
    }
}
```

Finally, you have a main application as follows:

```

package v1;

import java.util.Arrays;

public class SortMain {
    private static Postpack[] packs = { new
Postpack(60, 12345),
                                         new
Postpack(55, 98765),
                                         new
Postpack(50, 54321) };
    private static ZipSorter sorter = new
ZipSorter();
    public static void main(String[] args) {
        System.out.println("Sorted=" +
sorter.sort(Arrays.asList(packs)));
    }
}

```

### A futile attempt at a quick hack

We have been asked to make a version 2 of our application which supports not only `Postpack` objects but also `Box` objects. We must also support the ability to sort by weight as well as `Zip`. We have been given incredibly short time scales to develop the application, so we decide to write all of the new functionality using Groovy.

We start by creating a Groovy `Box` class and make it behave in the way `ZipSorter` is expecting, e.g. we make it implement `Comparable` even though this wouldn't be needed if everything was going to be in Groovy. We then modify `ZipSorter` to know about `Box`. We then create `WeightSorter` and write it to know about both `Postpack` and `Box`.

To simplify development (we think) we create separate `groovy` and `java` source directories. We develop our files incrementally in our IDE and everything works fine. We think we are finished, so we do a rebuild all for our project. All of a sudden the project won't compile. We dive out to `ant` and use the `javac` and `groovyc` tasks to put compile our separated source directories. Still no luck. What happened?

We inadvertently introduced a cyclic dependency into our codebase and we only got away with it originally because the incremental development style we were using hid away the problem. The issue is that IDEs and current build

systems like Ant use different compilers for Java and Groovy. So while Java and Groovy are the same at the bytecode level, their respective compilers know nothing about the source code of the other language. (Recent discussions have begun about how to eventually remove this separation).

Our hack failed because if we run `javac` first, `ZipSorter` won't have the `Box` class available because it is written in Groovy. If we run `groovyc` first, `WeightSorter` doesn't have the `Postpack` class because it is written in Java. Similarly, if using our IDE, we will face the same deadly embrace problem.

### A proper version 2 of our application

The way we get around this problem is to define some common interfaces as follows:

```
package v2;

public interface Parcel {
    int getWeight();
    int getZip();
}
```

```
package v2;

import java.util.List;

public interface Sorter {
    List sort(Parcel[] unsorted);
}
```

Now we write our Java and Groovy parts of the system being careful to refer only to the interfaces, e.g. the Java files would become:

```
package v2;

public class Postpack implements Parcel,
Comparable {
    private final int weight;
    private final int zip;

    public int getWeight() {
        return weight;
    }

    public int getZip() {
        return zip;
    }

    public String toString() {
        return "Postpack[Weight=" + weight +
", Zip=" + zip + " ]";
    }

    public Postpack(int weight, int zip) {
        this.weight = weight;
        this.zip = zip;
    }

    public int compareTo(Object o) {
        Parcel other = (Parcel) o;
        return zip - other.getZip();
    }
}
```

```
package v2;

import java.util.List;
import java.util.Arrays;

public class ZipSorter implements Sorter {
    public List sort(Parcel[] items) {
        Arrays.sort(items);
        return Arrays.asList(items);
    }
}
```

And the Groovy ones look like:

```
package v2
public class Box implements Parcel,
Comparable {
    int weight
    int zip
    String toString() {
        return "Box[Weight=" + weight + ",
Zip=" + zip + "]"
    }
    int compareTo(other) { return zip -
other.zip }
}
```

```
package v2
public class WeightSorter implements Sorter
{
    List sort(Parcel[] items) {
        items.toList().sort{ p1, p2 ->
p1.weight <=> p2.weight }
    }
}
```

Finally, our main method looks like:

```

package v2;

import java.util.Arrays;

public class SortMain {
    private static Box box1 = new Box();
    private static Parcel[] packs = { new
Postpack(60, 12345), box1,
                                     new
Postpack(50, 54321) };
    private static Sorter zipSorter = new
ZipSorter();
    private static Sorter weightSorter = new
WeightSorter();
    public static void main(String[] args) {
        box1.setWeight(55);
        box1.setZip(99999);
        System.out.println("Unsorted:
" + Arrays.asList(packs));
        System.out.println("Sorted by
weight: " + weightSorter.sort(packs));
        System.out.println("Sorted by zip:
" + zipSorter.sort(packs));
    }
}

```

We need to compile the interfaces first, then we can compile the Groovy or Java files (excluding `SortMain`) in either order. Finally we compile `SortMain` as it is the class that knows about the concrete implementations. If we were using dependency injection or our own factory methods we could have reduced or eliminated the need to treat `SortMain` as a special case, e.g. using Spring we could have the concrete classes listed in an external `beans.xml` file and `SortMain` whether it was written in Java or Groovy could have been compiled along with all the other files written in its language.

Now when we run the program we get the following output:

```
Unsorted:          [Postpack[Weight=60,
Zip=12345], Box[Weight=55, Zip=99999],
Postpack[Weight=50, Zip=54321]]
Sorted by weight: [Postpack[Weight=50,
Zip=54321], Box[Weight=55, Zip=99999],
Postpack[Weight=60, Zip=12345]]
Sorted by zip:     [Postpack[Weight=60,
Zip=12345], Postpack[Weight=50, Zip=54321],
Box[Weight=55, Zip=99999]]
```

For more details, see chapter 11 of [GINA](#).

### Influencing class loading at runtime

When writing a script, it may be unwieldy to call the script defining the whole classpath at the command line, e.g.

```
groovy -cp %JAXB_HOME%\bin\activation.jar;%JAXB_HOME%\bin\... myscript.groovy
```

You can go the other way - let the script itself find the jars it needs and add them to the classpath before using them. To do this, you need to

1. get the groovy rootloader

```
def loader = this.class.classLoader.rootLoader
```

2. introduce the necessary urls to groovy rootloader. Use whatever logic suits your situations to find the jars / class directories

```
def jardir = new File( System.getenv(
'JAXB_HOME' ), 'lib' )
def jars    = jardir.listFiles().findAll {
it.name.endsWith('.jar') }
jars.each {
loader.addURL(it.toURI().toURL()) }
```

3. Load the classes you need:

```
// in a script run from command line this is  
ok:
```

```
JAXBContext = Class.forName(  
'javax.xml.bind.JAXBContext' )  
Marshaller = Class.forName(  
'javax.xml.bind.Marshaller' )
```

```
// if the groovy script / class is loaded  
from a java app, then the above may fail as  
it uses the same classloader to load the  
class as the containing script / class was  
loaded by. In that case, this should work:
```

```
JAXBContext = Class.forName(  
'javax.xml.bind.JAXBContext', true, loader )  
Marshaller = Class.forName(  
'javax.xml.bind.Marshaller', true, loader )
```

4. To instantiate the classes, use the newInstance method:

```
def jaxbContext = JAXBContext.newInstance(  
MyDataClass )
```

Note that newInstance is on steroids when called from groovy. In addition to being able to call the parameterless constructor (as w/ Java's Class.newInstance()), you can give any parameters to invoke any constructor, e.g.

```
def i = MyClass.newInstance( "Foo", 12 ) //  
invokes the constructor w/ String and int as  
params
```

You can also pass a map to initialize properties, e.g.

```
def i2 = MyClass.newInstance(foo:'bar',
boo:12) // creates a new instance using the
parameterless constructor and then sets
property foo to 'bar' and property boo to 12
```

The downside of using this approach is that you can't inherit from the classes you load this way - classes inherited from need to be known before the script starts to run.

## Internationalization

Groovy leverages Java's I18N features. Here is a Swing example using Dates, Numbers and Locales:

```
import groovy.swing.SwingBuilder
import static
javax.swing.JFrame.EXIT_ON_CLOSE as EXIT
import static
java.text.NumberFormat.getNumberInstance as
numberFormat
import static
java.text.DateFormat.getDateInstance as
dateFormat
import static java.text.DateFormat.FULL
def locales = [
    new Locale("en", "US"), new
Locale("ja", "JP"), new Locale("iw", "IL"),
    new Locale("hi", "IN"), new
Locale("th", "TH", "TH"), new Locale("de",
"DE"),
    new Locale("es", "ES"), new Locale("zh",
"CN"), new Locale("it", "IT") ]
def swing = new SwingBuilder()
```

```

int row = 0
def d = new Date()
def frame = swing.frame(title:'Frame',
size:[300,300],
    defaultCloseOperation:EXIT) {
    tableLayout {
        locales.each { loc ->
            tr {
                td {
label(numberFormat(loc).format(++row)) }
                td { label(loc.toString()) }
                td {
label(loc.getDisplayLanguage(loc)) }
                td { label(dateFormat(FULL,
loc).format(d)) }
            }
        }
    }
}

```

```
}  
frame.pack()  
frame.show()
```

Which should look something like this:



You can also use properties files to capture language strings. Using the `LablesBundle.properties`, `LablesBundle.properties_de` and `LablesBundle_fr.properties` files from [this](#) tutorial. We can then use this code:

```
def locales = [Locale.FRENCH, Locale.GERMAN,  
Locale.ENGLISH]  
def keys = [ "s1", "s2", "s3", "s4" ]  
[locales, keys].combinations().each{ loc,  
key ->  
    def labels =  
ResourceBundle.getBundle("LabelsBundle",  
loc)  
    println "Locale = ${loc.toString()}, key  
= $key, value = ${labels.getString(key)}"  
}
```

To produce this result:

```
Locale = fr, key = s1, value = Ordinateur
Locale = de, key = s1, value = Computer
Locale = en, key = s1, value = computer
Locale = fr, key = s2, value = Disque dur
Locale = de, key = s2, value = Platte
Locale = en, key = s2, value = disk
Locale = fr, key = s3, value = Moniteur
Locale = de, key = s3, value = Monitor
Locale = en, key = s3, value = monitor
Locale = fr, key = s4, value = Clavier
Locale = de, key = s4, value = Tastatur
Locale = en, key = s4, value = keyboard
```

## Meta-annotations

### What are meta-annotations ?

Meta-annotations, also known as annotation aliases are annotations that are replaced at compile time by other annotations (one meta-annotation is an alias for one or more annotations). This feature was added in Groovy 2.1.0. It can be used to dramatically reduce the verbosity of code involving multiple annotations.

### Writing a meta-annotation

Let's start with a simple example. Imagine you have the *@Service* and *@Transactional* annotations and that you want to annotate a class with both:

```
@Service
@Transactional
class MyTransactionalService {}
```

What meta-annotations allows you is to replace the two annotations by a single one. First of all, you need to define your alias:

```
import groovy.transform.AnnotationCollector

@Service
@Transactional
@AnnotationCollector
public @interface TransactionalService {
}
```

We are defining a *@TransactionalService* annotation here, a meta-annotation, that *collects* the *@Service* and *@Transactional* annotations. This is done by annotating the interface with *@AnnotationCollector*.

### Understanding the behaviour

Before going further, we have to understand how meta-annotations work. First of all, Groovy supports both precompiled and source form meta-annotations. This means that your meta-annotation *may* be precompiled, or you can have it in the same source tree as the one you are currently compiling. Second, this is a Groovy feature only. There is no chance for you to annotate a Java class with a meta-annotation and hope it will do the same as in Groovy. Likewise, you cannot write a meta-annotation in Java: both the meta-annotation definition **and** usage have to be Groovy code.

When the Groovy compiler encounters a class annotated with a meta-annotation, it **replaces** it with the collected annotations. That is, in our previous example, that it will replace *@TransactionalService* with *@Transactional* and *@Service*. This happens during the semantic analysis phase, that is to say exactly when AST transformations are collected.

More than replacing the alias, a meta-annotation is capable of processing the collected annotations, including processing arguments. The default processor includes interesting behaviour with regards to annotation parameters.

### Meta-annotation parameters

Here, we will imagine two annotations, each of them taking one argument:

```
@Timeout(after=100)
@Dangerous(type="explosive")
```

And that you want create a meta-annotation named *@Explosive*:

```
@Timeout(after=100)  
@Dangerous(type="explosive")  
@AnnotationCollector  
public @interface Explosive {}
```

Then, by default, when the annotations are replaced, they will get the values as they were defined in the alias. More interesting, the alias supports overriding specific values:

```
@Explosive(after=0)  
class Bomb {}
```

Here, the value provided as a parameter to *@Explosive* overrides the one defined in the annotation.

#### ***Name clashes***

What happens if two annotations define the same parameter name? The default processor will copy the annotation value to all annotations that accept this parameter. Take this example (you can run it in a Groovy console, for example):

```
import java.lang.annotation.Retention
import java.lang.annotation.RetentionPolicy
@Retention(RetentionPolicy.RUNTIME)
public @interface Foo {
    String value()
}
@Retention(RetentionPolicy.RUNTIME)
public @interface Bar {
    String value()
}
@Foo
@Bar
@groovy.transform.AnnotationCollector
public @interface FooBar {}
@Foo('a')
@Bar('b')
class Bob {}
println Bob.class.getAnnotation(Foo)
println Bob.class.getAnnotation(Bar)
@FooBar('a')
class Joe {}
println Joe.class.getAnnotation(Foo)
println Joe.class.getAnnotation(Bar)
```

this would print:

```
@Foo(value=a)  
@Bar(value=b)  
@Foo(value=a)  
@Bar(value=a)
```

which means that in the second case, the annotation value was copied in both annotations. But what happens if, for example, the annotation type for Foo is String, and the one for Bar is an integer? The current behaviour of the annotation is to fail at compile time. Does it mean that there's nothing more you can do? No. Meta-annotations support a special feature called a processor, that will help you deal with such problems.

#### *Custom annotation processors*

A custom annotation processor will let you choose how to explode a meta-annotation. The behaviour of the meta-annotation is, in this case, totally up to you. You can do whatever you want. Let's take a look, for example, on how `@CompileDynamic` is implemented in Groovy 2.1.0. `@CompileDynamic` is a meta-annotation that replaces itself with `@CompileStatic(TypeCheckingMode.SKIP)`. The problem is that the default meta annotation processor doesn't support enums and we need to produce that enum. So, instead of defining `@CompileDynamic` like this:

```
@CompileStatic(TypeCheckingMode.SKIP)  
@AnnotationCollector  
public @interface CompileDynamic {}
```

We will define it like this:

```
@AnnotationCollector(processor =  
    "org.codehaus.groovy.transform.CompileDynamicProcessor")  
public @interface CompileDynamic {  
    }
```

The first thing you may notice is that our interface is no longer annotated with `@CompileStatic`. The reason for this is that we rely on the extra `processor` parameter, that references a class which will **generate** the annotation. And here is what this class looks like:

```

public class CompileDynamicProcessor extends
AnnotationCollectorTransform {
    private static final ClassNode
COMPILESTATIC_NODE =
ClassHelper.make(CompileStatic.class);
    private static final ClassNode
TYPECHECKINGMODE_NODE =
ClassHelper.make(TypeCheckingMode.class);
    public List<AnnotationNode>
visit(AnnotationNode collector,
AnnotationNode aliasAnnotationUsage,
AnnotatedNode aliasAnnotated, SourceUnit
source) {
        AnnotationNode node = new
AnnotationNode(COMPILESTATIC_NODE);
        node.addMember("value", new
PropertyExpression(new
ClassExpression(TYPECHECKINGMODE_NODE),
"SKIP"));
        return
Collections.singletonList(node);
    }
}

```

Our custom processor extends the default one and overrides the *public List<AnnotationNode> visit(AnnotationNode collector, AnnotationNode aliasAnnotationUsage, AnnotatedNode aliasAnnotated, SourceUnit source)* method. This method is responsible for returning the list of annotations that will be added to the AST tree in place of the alias. The body of this method, hence, is only there to generate the appropriate *AnnotationNode*, nothing more! Of course, you can rely on the superclass to deal with arguments, for example.

**Optimising Groovy bytecodes with Soot**

Here's a quick steps to get your Groovy classes optimised using Soot framework.

### ***Problems***

When using Groovy, sometime we encounter the OutOfMemoryException because PermGen space ran out. Optimisation of class files could help saving memory usage of JVM.

### ***Soot Framework***

Soot Framework is a bytecode optimisation, and manipulation framework, developed at McGill University, for Java. It comes with a vast set of optimisation techniques. We can also develop our own optimisation using the framework provided.

### ***Quick Optimisation***

With Soot, you can just run

```
> soot.Main -O my.package.MyGroovyClass
```

to obtain a optimised class.

### ***Analysis***

A trivial class is used in this analysis. The class is illustrated in Jimple format, an intermediate representation supported by Soot to see what's done with Soot optimisation.

Here's a command to get a Jimple file:

```
> soot.Main -f jimple  
org.codehaus.groovy.aop.tests.Target
```

and the following is a command for applying optimisation:

```
> soot.Main -O -f jimple  
org.codehaus.groovy.aop.tests.Target
```

We've now got two Jimple files, and can use a text editor to compare them. You can get this two files from the attachment below to see their differences.

The result from the optmisation shows that Soot can reduce the file size by 8.5%. This number is expected to be large when we apply the optimisation to the whole program.

### ***Second Experiment***

This is an updated result when applying the more complex optimisation to two of Groovy packages, groovy.lang and groovy.xml.

```
> soot.Main -O -via-shimple -process-dir  
groovy-1.1
```

Please note that not all Groovy classes were used in this experiment. Just two mentioned packages resides in the "groovy-1.1" folder. This experiment has done by performing optimisation package that requires Static Single Assignment (SSA) representation of Java classes. Soot supports this through Shimple format. The "-via-shimple" option shown in the command line is to enable this kind of optimisation. The result is better than using only Jimple optimisation. Size of Groovy classes in the two packages is reduced by **16.34%**.

## Polyglot Programming with Groovy

A fun activity in the late 1980's and early 1990's was to write [polyglot](#) programs that ran in multiple languages, e.g. check out [this one](#) which runs in COBOL, Pascal, Fortran, C, PostScript, Linux/Unix shell script (bash, sh, csh), x86 machine language and Perl! More recently the term has resurfaced (e.g. [Neal](#), [Martin](#) and others) in recognition of the fact that a developer no longer lives in a world dominated just by one language, e.g. COBOL, Java, C#. As a minimum, a developer must know JavaScript for AJAX sights, SQL for databases, XML, various build file dialects, various scripting languages etc., in addition to one or more mainstream languages.

This interest has been increasing lately with multiple languages being [emphasised](#) on both the [JVM](#) and the [CLR](#). Groovy obviously supports a polyglot relationship with Java (e.g. A Groovy class can inherit from a Java class which can inherit from a Groovy class). In other cases Groovy reduces the need to introduce multiple languages because it gives you a Groovy syntax where you might otherwise have to introduce a new language, e.g. using GPath instead of introducing XPath, using datasets instead of SQL, AntBuilder instead of Ant XML files and so on. Other features just make integration with other languages a little easier, e.g. Groovy's regex, multi-line string, GString and template support.

Groovy also has other Polyglot friendly features or integration possibilities:

- Consider combining [Functional Programming with Groovy](#) with Groovy
- Considering using some [Constraint or logic programming](#) with your Groovy
- Consider accessing other [languages via JSR-223](#)
- Consider using Groovy as a [front end scripting language](#) for your Hadoop Cluster
- [Calling Scala from Groovy](#)
- [Calling Clojure from Groovy](#)

## Calling Clojure from Groovy

There are a number of ways to mix Groovy and Clojure. You can access Clojure classes from Groovy and vice versa by pre-compiling.

You can access via Java 6's scripting API: [JSR-223 access to other JVM languages](#)

You can also get access to Clojure's classes from Groovy, e.g. (for Groovy 1.7 snapshot):

```
@Grab(group='org.clojure', module='clojure',  
version='1.0.0')  
import clojure.lang.*  
  
def ss = StringSeq.create('The quick brown  
fox')  
def done = false  
while (!done) {  
    println ss.first()  
    ss = ss.next()  
    done = !ss  
}
```

Or interact via creating a new Process (again for Groovy 1.7 snapshot):

```

@Grab(group='org.clojure', module='clojure',
version='1.0.0')
import clojure.lang.Script

def src = new File('temp.clj')
src.text = '''
(defn factorial [n]
  (if (< n 2)
    1
    (* n (factorial (- n 1)))))
(println (factorial 4))
'''

def path = System.getProperty('user.home') +
'/.groovy/grapes/org.clojure/clojure/jars/cl
ojure-1.0.0.jar'
new AntBuilder().with {
  java(classname:Script.name,
classpath:path) {
    arg(value:src.path)
  }
}

```

Another option is to use the Clojure API like this:

```

@Grab(group='org.clojure', module='clojure',
version='1.0.0')
import clojure.lang.Compiler
import clojure.lang.RT

def src = new File('temp.clj')
src.text = '''
(ns groovy)
(defn factorial [n]
  (if (< n 2)
      1
      (* n (factorial (- n 1)))))
'''

src.withReader { reader ->
  Compiler.load reader
}

def fac = RT.var('groovy', 'factorial')
println fac.invoke(5)

```

There is also a Clojure plugin for Grails which provides easy access to execute clojure code from any Grails artifact (controllers, taglibs, services etc...): <http://grails.org/plugin/clojure>  
 Not to be left behind Griffon also sports a clojure plugin, based on the Grails one: <http://griffon.codehaus.org/Clojure+Plugin>. Among it's features you'll find: script loading at runtime (like Grails), class generation using clojure's macros and a repl.

### Calling Scala from Groovy

First write a Scala class:

## Complex.scala

```
class Complex(real: double, imaginary:
double) {
  def re = real
  def im = imaginary
  override def toString() = "" + re + (if
(im < 0) "" else "+") + im + "i"
}
```

Compile this using scalac:

```
> scalac Complex.scala
```

Now write our Groovy Program:

## ComplexMain.groovy

```
println new Complex(1.2, 3.4)
```

Now run the program (assuming scala-library.jar is in the CLASSPATH):

```
> groovy ComplexMain
```

Which produces:

```
1.2+3.4i
```

Note that in this example it would have been just as easy to write our Complex class using Groovy as follows:

```

class Complex {
  def re, im
  Complex (double real, double imaginary) {
    re = real
    im = imaginary
  }
  String toString() { "$re" + (im<0 ? '-' : '+') + im + 'i' }
}

```

but in other cases you may have some existing Scala classes you wish to reuse from Groovy.

## Constraint Programming

[Constraint programming](#) is a style of programming where relations between variables are stated in the form of constraints. This form of programming was first made popular by logic programming languages such as [Prolog](#) but the style is now also used outside logic programming specific languages. Constraints differ from the common primitives of other programming languages in that they do not specify one or more steps to execute but rather the properties of a solution to be found. If you want use a constraint-specific programming language, maybe Prolog is for you. If however you like Groovy, but want to apply a touch of constraint style flavor to your code, read on to learn about Gecode/J, Choco and tuProlog integration possibilities.

### *Gecode/J*

Gecode/J is a Java interface for the Gecode C++ constraint programming library. Let's have a look at making the [Money example](#) from the Gecode/J library a bit Groovier. If you haven't seen the example before, it involves solving the following puzzle:

```

S E N D
+  M O R E
= M O N E Y

```

How can I replace the letters in the puzzle with numbers so that the numbers add up in the expected way? See [here](#) for further details.

### A little helper library

First we can create a little helper library to make our subsequent code a little Groovier:

```

import static org.gecode.Gecode.*
import static
org.gecode.GecodeEnumConstants.*
import org.gecode.*

class GroovySpace extends Space {
    public GroovySpace() {
        super()
    }
    public GroovySpace(Boolean share,
GroovySpace space) {
        super(share, space)
    }
    def notEqual(IntVar v, val) {
        rel(this, v, IRT_NQ, val, ICL_BND)
    }
    def distinct(VarArray<IntVar> v) {
        distinct(this, v, ICL_BND)
    }
    def expressionsEqual(Expr e1, Expr e2) {
        post(this, e1, IRT_EQ, e2)
    }
    def branchOverVariables(VarArray<IntVar>
v) {
        branch(this, v, BVAR_SIZE_MIN, BVAL_MIN)
    }
    def variableOverRange(v, r) {
        new IntVar(this, v, r.from, r.to)
    }
}

```

Here is how we could now code the solution.

```
import org.gecode.*

class Money extends GroovySpace {
    public VarArray<IntVar> letters

    public Money() {
        super()

        // We need 8 variables. Each one gets
the name of it's
        // corresponding letter, and an initial
domain of [0..9]
        letters = new VarArray<IntVar>(8)
        "SENDMORY".each{
letters.add(variableOverRange(it, 0..9)) }
        def s = letters[0]
        def e = letters[1]
        def n = letters[2]
        def d = letters[3]
        def m = letters[4]
        def o = letters[5]
        def r = letters[6]
        def y = letters[7]

        // set up constraints
        notEqual(s, 0)
        notEqual(m, 0)
        expressionsEqual(
```

```
    new Expr()
      .p(1000, s).p(100, e).p(10, n).p(d)
      .p(1000, m).p(100, o).p(10, r).p(e),
    new Expr()
      .p(10000, m).p(1000, o).p(100,
n).p(10, e).p(y))
  distinct(letters)
```

```
  // start looking for solutions
  branchOverVariables(letters)
}
```

```
public Money(Boolean share, Money money) {
  super(share, money)
  letters = new VarArray<IntVar>(this,
share, money.letters)
}
}
```

```
def opt = new Options("SEND+MORE=MONEY")
```

```
opt.gui = true  
opt.parse(args)  
opt.doSearch(new Money())
```

It relies on the same `options` class used in the original Java example.

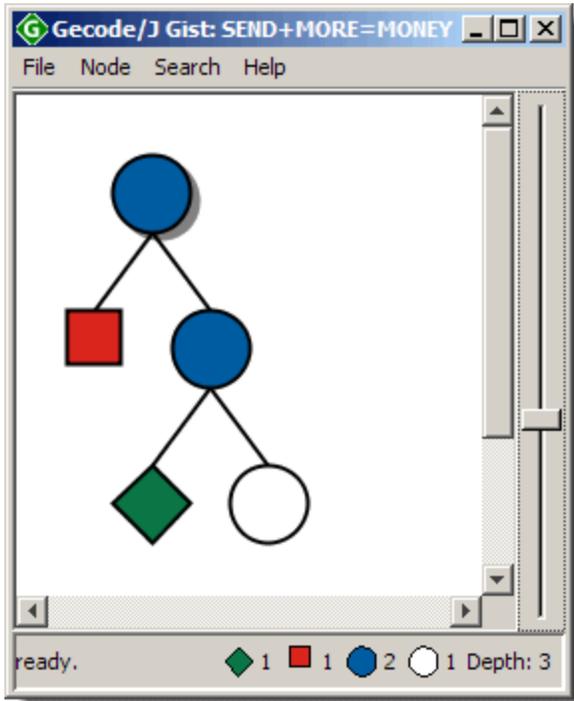
Running this code (we used a Java 6 JVM with the following JVM argument, `-Djava.library.path=C:\Geocode\bin`) yields:

```
SEND+MORE=MONEY:  
letters=[S=9,E=5,N=6,D=7,M=1,O=0,R=8,Y=2]
```

**Summary:**

```
runtime:           3  
solutions:        1  
propagations:    11  
failures:         1  
clones:           2  
commits:          3  
peak memory:     6KB
```

Running the program with the `-gui` command line argument yields the following GUI display:



It would be fun to take this example a little further with a proper DSL to define the constraints.

### *Choco*

[Choco](#) is a java library for constraint satisfaction problems (CSP), constraint programming (CP) and explanation-based constraint solving (e-CP). It is built on an event-based propagation mechanism with backtrackable structures. It's a great alternative to Gecode/J especially if you want a Java only solution.

### Choco Magic Squares

We'll look at solving the [Magic Square](#) problem (though we haven't coded up making the diagonals add up too, just the rows and columns). Converting [this example](#) to Groovy yields:

```
import choco.integer.IntVar
import choco.integer.IntDomainVar
import choco.Problem

int n = 4
int nsq = n * n
println "Magic Square Problem with n = " + n

def p = new Problem()
// create variables
```

```

def vars = new IntDomainVar[nsq]
for (i in 0..<n)
  for (j in 0..<n)
    vars[i * n + j] =
p.makeEnumIntVar("C${i}_${j}", 1, nsq)
int sumAllSquares = nsq * (nsq + 1) / 2
def sum = p.makeEnumIntVar("S", 1,
sumAllSquares)

// vars should be pairwise distinct
for (i in 0..<nsq)
  for (j in 0..<i)
    p.post(p.neq(vars[i], vars[j]))

int[] coeffs = [1] * n

for (i in 0..<n) {
  // all cols should add to sum
  IntVar[] col = (0..<n).collect{ j ->
vars[i * n + j] }
  p.post(p.eq(p.scalar(coeffs, col), sum))
  // all rows should add to sum
  IntVar[] row = (0..<n).collect{ j ->
vars[j * n + i] }
  p.post(p.eq(p.scalar(coeffs, row), sum))
}

p.solve() // find first solution
for (i in 0..<n) {
  for (j in 0..<n)
    print vars[i * n +

```

```

j].val.toString().padLeft(3)
  println()
}

```

And has this output when run:

### Magic Square Problem with n = 4

```

 1  2 15 16
 6 11 10  7
13  9  4  8
14 12  5  3

```

#### Going Further

We can have a little more fun with this example. First, let's make the diagonals also add up:

```

IntVar[] downdiag = (0..<n).collect{ i ->
vars[i * n + i] }
IntVar[] updiag = (0..<n).collect{ i ->
vars[i * n + n - i - 1] }
p.post(p.eq(p.scalar(coeffs, downdiag),
sum))
p.post(p.eq(p.scalar(coeffs, updiag), sum))

```

Now, let's also make the left kite (defined as below):

```

--  xx  --  --
xx  --  xx  --
--  --  --  --
--  xx  --  --

```

Also add up to the magic square value:

```
IntVar[] leftkite = [vars[1], vars[4],  
vars[6], vars[13]]  
p.post(p.eq(p.scalar(coeffs, leftkite),  
sum))
```

Finally, let's make the middle columns in the bottom row equal to '1514':

```
IntVar[] date = [vars[13], vars[14]]  
int[] datecoeffs = [100, 1]  
p.post(p.eq(p.scalar(datecoeffs, date),  
1514))
```

Let's see what solutions we now have:

```
def more = p.solve()  
while (more) {  
  println "Success at move:  
$p.solver.searchSolver.nodeCount"  
  for (i in 0..  
n) {  
    for (j in 0..  
n) {  
      print vars[i * n +  
j].val.toString().padLeft(3)  
      println()  
    }  
    more = p.nextSolution()  
  }  
}
```

Which yields the following results:

## Dürer Magic Square Problem

Success at move: 1231

16	1	4	13
7	10	11	6
9	8	5	12
2	15	14	3

Success at move: 1236

16	1	4	13
11	6	7	10
5	12	9	8
2	15	14	3

Success at move: 1299

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Success at move: 1300

16	3	2	13
9	6	7	12
5	10	11	8
4	15	14	1

The third solution in this list matches the famous Albrecht Dürer engraving.

This engraving has the following properties (from wikipedia):

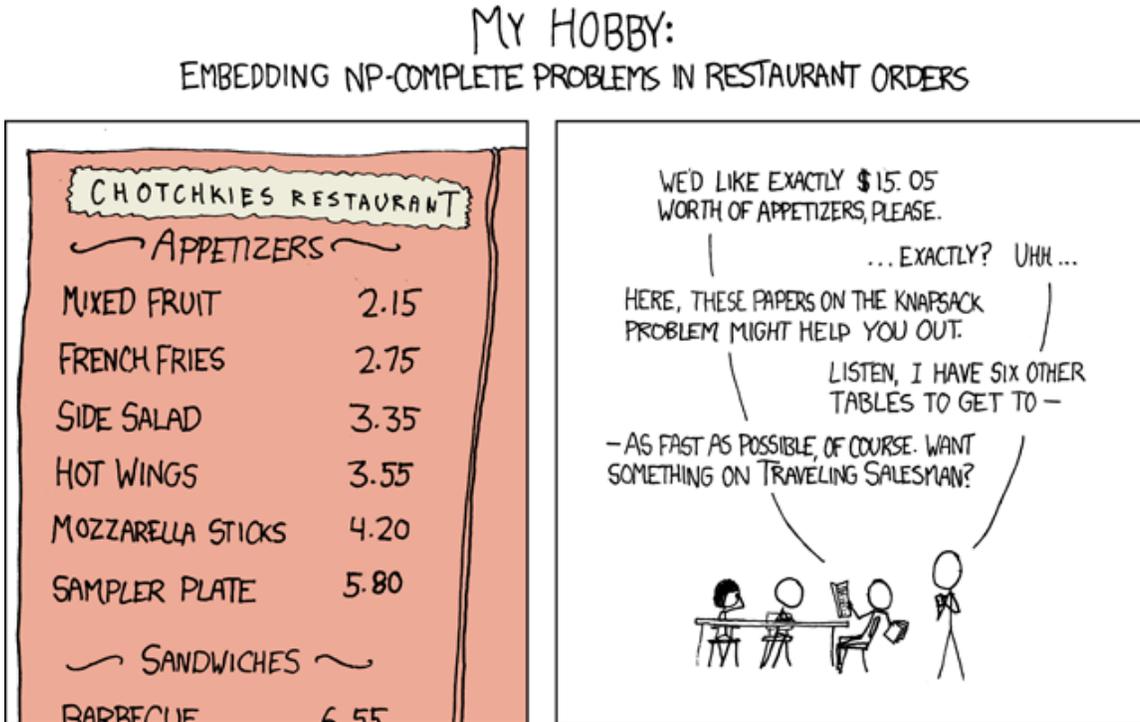
"The order-4 magic square in Albrecht Dürer's engraving Melancholia I is believed to be the first seen in European art. The sum 34 can be found in the rows, columns, diagonals, each of the quadrants, the center four squares, the corner squares, the four outer numbers clockwise from the corners (3+8+14+9) and likewise the four counter-clockwise (the locations of four queens in the two solutions of the 8 queens puzzle), the two sets of four symmetrical numbers (2+8+9+15 and 3+5+12+14) and the sum of the middle two entries of the two outer columns and rows (e.g. 5+9+8+12), as



well as several kite-shaped quartets, e.g. 3+5+11+15; the two numbers in the middle of the bottom row give the date of the engraving: 1514."

#### Choco menu selection example

Inspired by this [xkcd cartoon](#), here is a solution to a little menu selection problem.



This was solved using Choco version 2.1. Choco also supports reals, but to make things easy, I used price in cents.

```
import static choco.Choco.*
import
choco.kernel.model.variables.integer.Integer
Variable

def m = new choco.cp.model.CPModel()
def s = new choco.cp.solver.CPSolver()

def menu = [
    'Mixed fruit'           : 215,
    'French fries'         : 275,
    'Side salad'           : 335,
```

```

    'Hot wings'           : 355,
    'Mozzarella sticks'  : 420,
    'Sampler plate'      : 580
]
def numOrdered = new
IntegerVariable[menu.size()]
def priceEach = new int[menu.size()]
def sum = 1505
menu.forEachWithIndex { name, price, i ->
    // number ordered >= 0
    // number ordered * price <= sum
    numOrdered[i] = makeIntVar(name, 0,
sum.intdiv(price))
    priceEach[i] = price
}
m.addConstraint(eq(scalar(numOrdered,
priceEach), sum))
s.read(m)

def more = s.solve()
while (more) {
    println "Found a solution:"
    numOrdered.each {
        def v = s.getVar(it)
        if (v.val) println " $v.val *
$v.name"
    }
}

```

```
    }  
    more = s.nextSolution()  
}
```

It produces this output:

```
Found a solution:  
  7 * Mixed fruit  
Found a solution:  
  1 * Mixed fruit  
  2 * Hot wings  
  1 * Sampler plate
```

### *tuProlog*

[tuProlog](#) is a Java-based light-weight Prolog for Internet applications and infrastructures. It offers numerous integration options: allowing Prolog syntax to be used in the form of *theories*; programatic theory construction using a Java-based API; and mechanisms to call back into Groovy and Java from Prolog.

### **Magic Squares**

We can code up an order 3 Magic square problem as follows:

```
//  
require(url:'http://www.alice.unibo.it/tuProlog/', jar:'tuprolog.jar', version:'2.1')  
import alice.tuprolog.*  
  
def engine = new Prolog()  
def digits = (1..9).collect{ "d($it)."  
}.join('\n')  
def vars = ('A'..'I')  
def combos = []  
( 'A'..'H' ).each { outer ->
```

```

    (outer.next()..'I').each{ inner ->
        combos << /$outer=\=$inner/
    }
}
def all = vars.join(',')
engine.theory = new Theory("""
$digits
sum(X,Y,Z):-d(X),d(Y),d(Z), (X+Y+Z)=:=15.
rows($all):-sum(A,B,C),sum(D,E,F),sum(G,H,I)
.
cols($all):-sum(A,D,G),sum(B,E,H),sum(C,F,I)
.
diags($all):-sum(A,E,I),sum(G,E,C).
distinct($all):-${combos.join(',')}.
""")
def answer =
engine.solve("rows($all),cols($all),diags($a
ll),distinct($all).")
def squares = ('A'..'I').collect{
answer.getTerm(it).toString().padLeft(3) }
for (row in 0..2) {
    for (col in 0..2)

```

```
        print squares[row * 3 + col]
    println()
}
```

The result is:

```
2  7  6
  9  5  1
  4  3  8
```

Family Trees

Finally, here is a Groovy version of the typical first Prolog program, which demonstrates the simplest of *relational reasoning*:

```
//
require(url:'http://www.alice.unibo.it/tuprolog/', jar:'tuprolog.jar', version:'2.1')
import alice.tuprolog.*

/** Pretty Printing */
def pprint(soln) {
    println soln.isSuccess() ? "$soln.query
= $soln.solution" : 'no solution found'
}

/** Prolog clauses */
def getTheory() {
    new Theory(" "

parent(pam, bob).
parent(tom, bob).
```

```
parent(tom, liz).  
parent(bob, ann).  
parent(bob, pat).  
parent(pat, jim).
```

```
female(pam).  
male(tom).  
male(bob).  
female(liz).  
female(pat).  
female(ann).  
male(jim).
```

```
offspring(X,Y) :- parent(Y,X).
```

```
mother(X,Y) :-  
    parent(X,Y),  
    female(X).
```

```
father(X,Y) :-  
    parent(X,Y),  
    male(X).
```

```
grandparent(X,Z) :-  
    parent(X,Y),  
    parent(Y,Z).
```

```
grandmother(X,Y) :-  
    grandparent(X,Y),  
    female(X).
```

```
grandfather(X,Y) :-  
    grandparent(X,Y),  
    male(X).
```

```
sister(X,Y) :-
/* different(X,Y), */
  parent(Z,X),
  parent(Z,Y),
  female(X).
brother(X,Y) :-
/* different(X,Y), */
  parent(Z,X),
  parent(Z,Y),
  male(X).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Z) :-
  parent(X,Y),
  ancestor(Y,Z).

""")
}

def engine = new Prolog()
engine.theory = theory
pprint engine.solve('ancestor(tom,X).')
```

```
while(engine.hasOpenAlternatives()) {  
    pprint engine.solveNext()  
}
```

The Prolog engine will determine that Tom's ancestors are: bob, liz, ann, pat & jim.

```
ancestor(tom, X) = ancestor(tom, bob)  
ancestor(tom, X) = ancestor(tom, liz)  
ancestor(tom, X) = ancestor(tom, ann)  
ancestor(tom, X) = ancestor(tom, pat)  
ancestor(tom, X) = ancestor(tom, jim)  
no solution found
```

## Functional Programming with Groovy

[Functional programming](#) is a style of programming that emphasizes the application of functions to solve computing problems. This is in contrast with imperative programming, which emphasizes changes in state and the execution of sequential commands. If you want use a functional-only programming language, you should consider something like [Haskell](#). If however you like Groovy but want to apply some functional style magic, read on.

### *Functional Basics*

Groovy's functions (like Java's) can be used to define functions which contain no imperative steps, e.g. a factorial function may look like:

```
def fac(n) { n == 0 ? 1 : n * fac(n - 1) }  
assert 24 == fac(4)
```

In Groovy, we gain some slight syntactic sugar over Java in that we can leave out the *return* statements (the last evaluated expression is the default return value).

You can also define the above factorial function using a Closure:

```
def fac2 = { n -> n == 0 ? 1 : n * call(n -  
1) }  
assert 24 == fac2(4)
```

As you can see from the last line, this Closure version gets called in the same way as the function version (although a long hand `fac2.call(4)` is also supported). The closure version has the advantage that we can pass the `fac2` Closure around (e.g. as a parameter) or place it within a data structure.

We can also convert between functions and Closures, e.g.

```
def fac3 = this.&fac  
assert 24 == fac3(4)
```

The `fac3` variable will also be a closure.

We can of course start to mix and match functional and imperative coding styles as in this quicksort-like example:

```
def sort(list) {
  if (list.isEmpty()) return list
  anItem = list[0]
  def smallerItems = list.findAll{it <
anItem}
  def equalItems = list.findAll{it ==
anItem}
  def largerItems = list.findAll{it >
anItem}
  sort(smallerItems) + equalItems +
sort(largerItems)
}

assert [1, 3, 4, 5] == sort([1, 4, 5, 3])
assert [1, 1, 3, 4, 4, 5, 8] == sort([4, 1,
4, 1, 8, 5, 3])
assert ['a', 'b', 'c'] == sort(['c', 'b',
'a'])
```

### *Curry functions*

You can fix the values for one or more arguments to a closure instance using the `curry()` method as follows:

```

def joinTwoWordsWithSymbol = { symbol,
first, second -> first + symbol + second }
assert joinTwoWordsWithSymbol('#', 'Hello',
'World') == 'Hello#World'

def concatWords =
joinTwoWordsWithSymbol.curry(' ')
assert concatWords('Hello', 'World') ==
'Hello World'

def prependHello =
concatWords.curry('Hello')
// def prependHello =
joinTwoWordsWithSymbol.curry(' ', 'Hello')
assert prependHello('World') == 'Hello
World'

```

If you supply one argument to the `curry()` method you will fix the first argument. If you supply  $N$  arguments you will fix arguments  $1..N$ . See reference 1 or Chapter 5 of [GINA](#) for further details.

### *Lazy evaluation*

One particular style of functional programming of particular merit is to make use of lazy evaluation. This allows you to define infinite structures (see the next section), devise particularly efficient solutions to certain kinds of problems, and come up with very elegant solutions to otherwise hard problems. The good news is that several parts of Groovy already make use of this style and they typically hide away the hard bits so you don't need to know what magic is happening on your behalf. Here's some examples:

- `XmlSlurper` allows arbitrary GPath expressions to be crafted. As you create the expressions, you might think that XML parsing is going on behind the covers pulling XML nodes into and out of lists to match what your expressions are asking for. This is not the case. Instead a lazy representation of your GPath is stored away. When you need to evaluate the final result of a GPath expression, it calculates just what it needs to determine the expressions resulting value. [See chapter 12 of [GINA](#) for more information about `XmlSlurper`.]
- Groovy's `DataSet` feature does the same thing for data stored in relational databases. As you build up your dataset queries, no connections or operations to the database are happening under the covers. At the time when you need the result, an optimised query minimising SQL traffic is invoked to return the required result. [See section 10.2 of [GINA](#) for more information about `DataSets`.]

### *Infinite structures*

See reference 2 below for all the details, but to give you a flavour, first you must define some lazy list handling functions, then you can define and use infinite streams. Here is an example:

```
// general purpose lazy list class
class LazyList {
  def car
  private Closure cdr
  LazyList(def car, Closure cdr) {
this.car=car; this.cdr=cdr }
  def LazyList getCdr() { cdr ? cdr.call()
: null }
  def List take(n) {
    def r = []; def l = this
    n.times { r.add(l.car); l = l.cdr }
    r
  }
  def LazyList filter(Closure pred) {
    if (pred(car)) return
pred.owner.cons(car, { getCdr().filter(pred)
})
    else return getCdr().filter(pred)
  }
}

// general purpose lazy list function
def cons(val, Closure c) { new LazyList(val,
c) }

// now define and use infinite streams
def integers(n) { cons(n, { integers(n+1) })
}
```

```
def naturalnumbers = integers(1)
assert '1 2 3 4 5 6 7 8 9 10' ==
naturalnumbers.take(10).join(' ')
def evennumbers = naturalnumbers.filter{ it
```

```

% 2 == 0 }
assert '2 4 6 8 10 12 14 16 18 20' ==
evennumbers.take(10).join(' ')

```

If you are not familiar with traditional functional programming terms like `cons`, `car` and `cdr`, you may find the refactored version below (creds to Alexander Kriegisch) somewhat more readable:

```

class LazyList {
  def head
  private Closure tail
  LazyList(def head, Closure tail) {
this.head=head; this.tail=tail }
  def LazyList getTail() { tail ?
tail.call() : null }
  def List getFirst(n) {
    def result = []; def current = this
    n.times { result.add(current.head);
current = current.tail }
    result
  }
  def LazyList filter(Closure matchExpr) {
    if (matchExpr(head))
      return
matchExpr.owner.prepend(head, {
getTail().filter(matchExpr) })
    else
      return
getTail().filter(matchExpr)
  }
}

```

```
// general purpose lazy list function
def prepend(val, Closure c) { new
LazyList(val, c) }

// now define and use infinite streams
def integers(n) { prepend(n, { integers(n+1)
}) }
def naturalnumbers = integers(1)
assert '1 2 3 4 5 6 7 8 9 10' ==
naturalnumbers.getFirst(10).join(' ')
def evennumbers = naturalnumbers.filter{ it
```

```
% 2 == 0 }  
assert '2 4 6 8 10 12 14 16 18 20' ==  
evennumbers.getFirst(10).join(' ')
```

### *Using Functional Java*

You can also consider using Java libraries which support functional programming, e.g. using [Functional Java](#) for an example similar to the evens example above:

```
import fj.data.Stream  
  
// some metaprogramming to make fj mesh well  
with Groovy  
Stream.metaClass.filter = { Closure c ->  
delegate.filter(c as fj.F) }  
Stream.metaClass.getAt = { n ->  
delegate.index(n) }  
Stream.metaClass.getAt = { Range r ->  
r.collect{ delegate.index(it) } }  
Stream.metaClass.asList = {  
delegate.toCollection().asList() }  
  
def evens = Stream.range(0).filter{ it % 2  
== 0 }  
assert evens.take(5).asList() == [0, 2, 4,  
6, 8]  
assert (8..12).collect{ evens[it] } == [16,  
18, 20, 22, 24]  
assert evens[3..6] == [6, 8, 10, 12]
```

Or for something a little more adventurous, we can consider calculating primes:

```
import fj.data.Stream
import static fj.data.Stream.cons

// some metaprogramming to make Closures
mesh well with fj functions and products
Stream.metaClass.filterTail = { Closure c ->
delegate.tail()._1().filter(c as fj.F) }
Stream.metaClass.static.cons = { h, Closure
c -> delegate.cons(h, ['_1':c] as fj.P1) }

def sieve(nums) {
  cons nums.head(), { sieve nums.filterTail{
n -> n % nums.head() != 0 } }
}

println sieve(Stream.range(2)).index(100) //
=> 547
```

Remember to add `functionaljava.jar` to your classpath. You will need a version greater than version 2.8 of Functional Java.

Another example:

```

@Grab('org.functionaljava:functionaljava:3.0
')
import static fj.data.Array.array
import static fj.data.List.list
import static fj.Show.*
import fj.F

arrayShow(intShow).println(array(1, 2,
3).map({ it + 42 } as F)) // => {43,44,45}
assert array(1, 2, 3).map({ it + 42 } as
F).toCollection().toList() == [43, 44, 45]
assert [1, 2, 3].collect{ it + 42 } == [43,
44, 45] // native Groovy for comparison

listShow(intShow).println(list(1, 2,
3).bind({ it % 2 ? list(it * 2) : list() }
as F)) // => <2,6>
assert list(1, 2, 3).bind({ it % 2 ? list(it
* 2) : list() } as
F).toCollection().toList() == [2, 6]
assert [1, 2, 3].collect{ it % 2 ? [it * 2]
: [] }.sum() == [2, 6] // native Groovy for
comparison

```

Or with a little bit of metaprogramming like so:

```

import static fj.data.Java.ArrayList_List as
asList
fj.data.List.metaClass.toList = {
delegate.toCollection().toList() }
List.metaClass.bind = { c ->
asList().f(delegate).bind({
asList().f(c(it)) } as F) }
List.metaClass.map = { c ->
asList().f(delegate).map(c as F) }

```

The above examples can be re-written like this:

```

assert [1, 2, 3].bind{ it % 2 ? [it * 2] :
[] }.toList() == [2, 6]
assert [1, 2, 3].map{ it + 42 }.toList() ==
[43, 44, 45]

```

### ***More Information***

See also:

1. [Practically Groovy: Functional programming with curried closures](#)
2. [Infinite Streams in Groovy](#)
3. [Functional Programming Languages](#)
4. [Why Functional Programming Matters](#)
5. [Functional programming in the Java language](#)
6. [Post on functional programming in Java - maybe a tad verbose](#)
7. [FunctionalJ - A library for Functional Programming in Java](#)
8. [Weak versus strong languages, who wins the fight?](#)
9. [Programming Languages: Application and Interpretation](#)
10. [Beyond Groovy 1.0: Groovy goes Lisp](#)
11. [Concurrency with Groovy](#)

### **JSR-223 access to other JVM languages**

[JSR-223](#) is designed to allow Java access to many scripting languages. This means that it also provides hooks for Groovy to those other languages.

JSR-223 comes with Java 6. Here is how you can use it. First we define a script manager:

```
mgr = new javax.script.ScriptEngineManager()
```

Now, we can call out to JavaScript:

```
// included in Java 6  
println 'javascript: ' +  
mgr.getEngineByName("javascript").eval(''  
function factorial(n) {  
    if (n == 0) {  
        return 1;  
    }  
    return n * factorial(n - 1);  
}  
factorial(4)  
''')
```

Or JRuby:

```
// requires jruby and jruby-engine jars
println 'jruby: ' +
mgr.getEngineByName("jruby").eval(''
def factorial(n)
  if n == 0
    1
  else
    n * factorial(n - 1)
  end
end
factorial(4)
''')
```

Or Jython:

```
// requires jython and jython-engine jars
engine = mgr.getEngineByName("jython")
engine.eval(''
def factorial(n):
  i=fact=1
  while i <= n:
    fact=fact*i
    i=i+1
  return fact

result = factorial(4)
''')
println 'jython: ' + engine.result
```

Or Clojure:

```
// requires clojure and clojure-engine jars
// Note: doesn't seem officially supported,
hacked a version from here:
// http://wiki.github.com/pmf/clojure-jsr223
engine = mgr.getEngineByName("clojure")
println 'clojure: ' + engine.eval(''
(defn factorial [n]
  (if (< n 2)
    1
    (* n (factorial (- n 1)))))
(factorial 4)
''')
```

Or Jaskell:

```
// requires jaskell and jaskell engine and
jparsec and jfunutil jars
engine = mgr.getEngineByName("jaskell")
engine.eval('factorial n = if n > 0 then n *
factorial (n-1) else 1')
println 'jaskell: ' + engine.eval('factorial
4')
```

The output from running these scripts is:

**javascript: 24.0**

**jruby: 24**

**ython: 24**

**clojure: 24**

**jaskell: 24**

See also: [Accessing Groovy from Java via JSR-223](#)

## Refactoring with Groovy

Code and Design [Refactoring](#) is any change to a computer systems' code or design which improves its readability or simplifies its structure without changing its results.

Here we illustrate refactoring techniques which are applicable to Groovy. In general, most Java refactoring techniques apply equally well to Groovy. Sometimes Groovy adds a new twist as well.

We don't discuss tool support to auto implement some of these refactoring techniques. Consult your IDE plugin documentation to see if it has any specific support that you can use.

### *Refactoring Techniques*

- [Introduce Assertion](#)
- [Replace Inheritance with Delegation](#)

### *References*

1. Martin Fowler (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley. ISBN 0-201-48567-2.
2. Joshua Kerievsky (2004). *Refactoring To Patterns*. Addison-Wesley. ISBN 0-321-21335-1.
3. Michael Feathers (2005). *Working Effectively with Legacy Code*. Prentice Hall. ISBN 0-13-117705-2.
4. Dierk Koenig with Andrew Glover, Paul King, Guillaume Laforge and Jon Skeet (2007). *Groovy in Action*. Manning. ISBN 1-932394-84-2. Discusses the importance of having tests in place before refactoring.

See also: [Design Patterns with Groovy](#)

## Introduce Assertion

The [Introduce Assertion](#) refactoring recommends that if a section of code is going to make an assumption about the current state of the program, that an explicit assertion should check those assumptions first.

A common usage of this refactoring is to check that each method (and potentially each constructor) checks its preconditions using assertions. This is a particular form of defensive programming. Some argue that if you have sufficient tests in place, you don't need to apply defensive programming. For the small extra performance penalty, it seems like a small price to pay for extra resilience in our system.

### Example

Suppose we have the following method:

```

/**
 * Interleave two strings.
 * Assumes input parameters aren't null.
 */
def interleave(String a, String b) {
    int maxlength = [a.size(),
b.size()].max()
    def sb = new StringBuffer()
    (0..

```

If we call it with valid parameters, everything is fine. If we call it with null we will receive a `NullPointerException` during the method's execution. This can sometimes be difficult to track down.

Applying this refactoring gives us the following code:

```

package introduceAssertion

def interleave(String a, String b) {
  assert a != null, 'First parameter must
not be null'
  assert b != null, 'Second parameter must
not be null'
  int maxlength = [a.size(),
b.size()].max()
  def sb = new StringBuffer()
  (0..<maxlength).each{
    if (it < a.size()) sb << a[it]
    if (it < b.size()) sb << b[it]
  }
  sb
}

println interleave('Hello', 'World!')
// => HWeolrllod!
//println interleave(null, 'World!')
// => AssertionError: First parameter must
not be null.

```

This is better because we become aware of any problems straight away.

### Replace Inheritance with Delegation

The [Replace Inheritance with Delegation](#) refactoring (sometimes known as *Replace Implementation Inheritance With Composition*) provides a systematic way to replace inheritance hierarchies.

Inheritance hierarchies allow system designers to elegantly express relationships between different types of objects in a system. However, such hierarchies can be difficult to implement and refactor. Many developers favour restricting the use of inheritance hierarchies to very simple scenarios. If you started out using inheritance hierarchies (perhaps you had a simple scenario to start with) but are now finding the inheritance hierarchy is now getting in the way, apply this pattern.

### Property Example

Suppose we have the following property-centric class:

```
class Person {  
    def name  
    def age  
    def nationality  
}
```

We might have a related class such as this:

```
class StaffMemberUsingInheritance extends  
Person {  
    def salary  
}
```

For this simple case, we can stop here. For more complicated cases, the inheritance might start getting in the way. Here is how you can remove it using traditional delegation:

```

class StaffMemberUsingDelegation {
    private delegate = new Person()
    def salary
    def getName() {
        delegate.name
    }
    def setName(name) {
        delegate.name = name
    }
    def getAge() {
        delegate.age
    }
    def setAge(age) {
        delegate.age = age
    }
    def getNationality() {
        delegate.nationality
    }
    def setNationality(nationality) {
        delegate.nationality = nationality
    }
}

```

It looks like we have greatly increased the size of our code. This is because the earlier example was making use of Groovy's compact property notation which we can't use in this case. We should notice however, that most of this code is fairly boiler-plate in style.

Even though this example is not too bad, it becomes annoying to have to read, write and maintain this boiler-plate code. Instead, we can make use of Groovy's Meta-Programming capabilities as follows:

```

class StaffMemberUsingMOP {
    private delegate = new Person()
    private hasLocalProperty(name) {
        metaClass.properties.collect{
it.name }.contains(name)
    }
    def salary
    StaffMemberUsingMOP(Map map) {
        map.each{ k, v -> setProperty(k, v)
    }
    }
    void setProperty(String name, value) {
        if (hasLocalProperty(name))
this.@"$name" = value
        else delegate.setProperty(name,
value)
    }
    def getProperty(String name) {
        if (hasLocalProperty(name)) return
this.@"$name"
        else return
delegate.getProperty(name)
    }
}

```

We can use the above classes with this script code:

```
def p1 = new
StaffMemberUsingInheritance(name:'Tom',
age:20, nationality:'French', salary:1000)
def p2 = new
StaffMemberUsingDelegation(name:'Dick',
age:25, nationality:'German', salary:1100)
def p3 = new
StaffMemberUsingMOP(name:'Harry', age:30,
nationality:'Dutch', salary:1200)
describe(p1)
describe(p2)
describe(p3)
```

With the result being:

```
Tom has a salary of 1000
Dick has a salary of 1100
Harry has a salary of 1200
```

#### Method Example

The above example focussed on classes that were very data centric and were implemented using properties. Let's consider the a similar example but this time with classes which are method centric.

First, we define a `Person` class.

```

class Person {
    private name
    private age
    Person(name, age) {
        this.name = name
        this.age = age
    }
    def haveBirthday() { age++ }
    def describe() { "$name is $age years
old" }
}

```

We can use inheritance to define a staff member class as follows:

```

class StaffMemberUsingInheritance extends
Person {
    private salary
    StaffMemberUsingInheritance(name, age,
salary) {
        super(name, age)
        this.salary = salary
    }
    def describe() {
        super.describe() + " and has a
salary of $salary"
    }
}

```

This works well here, but in complex systems the inheritance might start to complicate our system. An alternative way to implement the functionality using traditional delegation is shown here:

```
class StaffMemberUsingDelegation {
    private delegate
    private salary
    StaffMemberUsingDelegation(name, age,
salary) {
        delegate = new Person(name, age)
        this.salary = salary
    }
    def haveBirthday() {
        delegate.haveBirthday()
    }
    def describe() {
        delegate.describe() + " and has a
salary of $salary"
    }
}
```

The pattern here is simple, for each method in the delegate class that we want available in the staff member class, we create a method that explicitly calls the delegate. Simple but this is boiler-plate code that needs to change whenever we change the underlying classes.

So, as an alternative, we can use Groovy's Meta Object Programming facilities to auto delegate methods using `invokeMethod`:

```

class StaffMemberUsingMOP {
    private delegate
    private salary
    StaffMemberUsingMOP(name, age, salary) {
        delegate = new Person(name, age)
        this.salary = salary
    }
    def invokeMethod(String name, args) {
        delegate.invokeMethod(name, args)
    }
    def describe() {
        delegate.describe() + " and has a
salary of $salary"
    }
}

```

The result in this case looks like we didn't save much code at all. If however, there were lots of methods from the delegate that we needed to use, this one wouldn't grow in size, whereas the previous approach would become larger. Also, this version will require less maintenance as we don't need to explicitly change it if the `Person` class changes over time. Note that because the `describe` method didn't follow the *boiler-plate* approach of simple delegation (because it needed extra logic), we are still required to implement that method manually.

This script code shows the various classes described above in action:

```
def p1 = new
StaffMemberUsingInheritance('Tom', 20, 1000)
def p2 = new
StaffMemberUsingDelegation('Dick', 25, 1100)
def p3 = new StaffMemberUsingMOP('Harry',
30, 1200)
p1.haveBirthday()
println p1.describe()
p2.haveBirthday()
println p2.describe()
p3.haveBirthday()
println p3.describe()
```

Which results in the following output:

```
Tom is 21 years old and has a salary of 1000
Dick is 26 years old and has a salary of
1100
Harry is 31 years old and has a salary of
1200
```

Further Information on this Pattern

- [Delegation Pattern](#)
- [Replacing Inheritance with Composition](#)

## Security

Groovy is integrated with the Java security model. Groovy scripts can be compiled and executed in the presence of a `SecurityManager` and a `Policy` that dictates what permissions are granted to the script.

In a typical java environment, permissions are granted to code according to its **codeSource**. A `codeSource` consists of a **codebase** (essentially, the URL the code was loaded from by the class loader) and optionally the certificates used to verify the code (when it is obtained from a signed jar file). Since groovy can produce java `.class` files which can be loaded by existing secure class loaders (e.g. `URLClassLoader`), the traditional mechanisms can be used to enforce security policies without doing anything special. Setting up and running java security can be a little tricky, so consider the following resources for more information:

- [Java Security Tutorial](#)
- [Java Application Security](#)
- [Permissions in the Java 2 SDK](#)
- [Java 1.4 Security](#)
- [Java Security, 2nd Edition – O'Reilly](#)

The last of these is a book which covers the Java security model in detail.

In a typical groovy environment, there are additional considerations – often groovy scripts are loaded dynamically from some filesystem and translated *on the fly* into java class files. In other cases, groovy scripts may be entered via an interactive shell, or retrieved from a database for dynamic translation.

### Filesystem based Groovy scripts

In the case where the script is read from the filesystem, groovy uses a custom class loader [GroovyClassLoader](#) that searches the CLASSPATH for .groovy files and gives them a codeSource constructed from a codebase built from the source file URL. This class loader also supports signed .jar files containing .groovy scripts so that both codebase and certificates can be used to verify the source code. Once the groovy scripts are loaded as classes, they behave just like java classes with respect to security.

### Non-URL based Groovy scripts

In the case where the script has no URL, there is not necessarily any definitive way for groovy to associate an appropriate codeSource with the script. In these cases, groovy allows a codebase to be specified for the script that is being compiled (by specifying a [GroovyCodeSource](#)), subject to the caller having permission to specify that specific codebase. This codebase takes the form of a URL, but need not refer to a physical file location.

To illustrate this more clearly, consider the case where some server system is responsible for fetching and loading scripts that will be executed on behalf of a client. Assume that the server is trusted (i.e. it has permission to do anything) while the client belongs to a class of restricted clients that only (for example) have permission to access the normally restricted property "file.encoding". For this simple example, assume that the security Policy in effect has been specified by the following policy file:

```
grant codeBase
"file:${server.home}/classes/-" {
  permission java.security.AllPermission;
};

grant codeBase
"file:/serverCodeBase/restrictedClient" {
  permission java.util.PropertyPermission
"file.encoding", "read";
};
```

The groovy script to be executed on behalf of the client is:

```
//Do some work... then access the
file.encoding property:
fileEncoding =
System.getProperty("file.encoding");
```

When the client calls the server and passes this script for execution, the server can evaluate it, specifying a specific codebase:

```
new GroovyShell().evaluate(new
GroovyCodeSource(clientscriptStr,
"RestrictedScript",
"/serverCodeBase/restrictedClient")
```

In order for the server to be able to create a `GroovyCodeSource` with a specific `codeBase`, it must be granted permission by the Policy. The specific permission required is a [GroovyCodeSourcePermission](#), which the server has by implication (the policy file grant of `java.security.AllPermission`). The net effect of this is to compile the client script with the `codeBase` `/serverCodeBase/restrictedClient`, and execute the compiled script. When executed, the policy file grant(s) for the `codeBase` `/serverCodeBase/restrictedClient` will be in effect.

### Additional information

For more information, check out the security test cases in the groovy source code distribution. These tests specify a custom policy file `groovy.policy`, which is located in the security directory under the groovy-core CVS module. The class `SecurityTestSupport` (located at `src/test/groovy/security`) activates this policy by specifying it in the system property `"java.security.policy"`. Examining this policy file along with the test cases should detail the concepts discussed here.

Note that in a typical application environment, the policy would be located and activated either by using the default lookup mechanism (`policy.url.<n>` setting in `JAVA_HOME/jre/lib/security/java.security`) or as a VM argument: `-Djava.security.policy=/my/policy/file`.

### Related external articles

- [Groovy scripts and JVM Security](#)

## Writing Domain-Specific Languages

### Book on Domain-Specific Languages with Groovy

You can have a look at our [books section](#) listing Fergal Dearle's book titled "[Groovy for Domain-Specific Languages](#)" published by Packt Publishing.

### Tutorial on DSLs

Guillaume Laforge and Paul King presented a [session at SpringOne2GX on the various techniques for authoring DSLs](#), and integrating them in your applications.

Guillaume Laforge and John Wilson presented a [tutorial on Groovy DSLs](#) at the QCon 2007 conference in London.

Matt Secoske presented a session on [Implementing DSLs in Groovy](#) at the OSCON 2007 conference.

## Groovy features enabling DSLs

Groovy is particularly well suited for writing a DSL: [Domain-Specific Language](#). A DSL is a mini-language aiming at representing constructs for a given domain. Groovy provides various features to let you easily embed DSLs in your Groovy code:

- the [builder](#) concept lets you write tree structured languages
- you can add new methods and properties on arbitrary classes through [categories](#) or custom metaclasses, even numbers: 3.euros, 5.days, etc, as shown in this [article explaining how to write a unit handling mini-DSL](#)
- most [operators can be overloaded](#): 5.days + 6.hours, myAccount += 400.euros
- passing maps to methods makes your code look like methods have named parameters: move( x: 500.meters, y: 1.kilometer )
- you can also create your own control structures by passing [closures](#) as the last argument of a method call: ifOnce( condition )  
{ ... }; inTransaction { ... }
- it is also possible to add dynamic methods or properties (methods or properties which don't really exist but that can be intercepted and acted upon) by implementing [GroovyObject](#) or creating a custom [MetaClass](#)

Guillaume Laforge gave some [thoughts and examples](#) on that topic on his blog. John Wilson has implemented a DSL in his [Google Data Support](#) to make operations on dates easier.

Joachim Baumann wrote an article showing how to [implement a small DSL for measurement calculation](#), which uses some of the techniques like adding properties to numbers, or overloading operators. Guillaume Laforge also wrote an article on [representing units using the JScience library](#).

Andy Glover also plays with internal DSLs in Groovy by producing a [behavior testing DSL](#).

Inspired by an article from Bruce Tate [IBMs Alphaworks](#) a couple of samples were written in groovy.

- [Chris vanBuskirk](#) uses arrays and maps to model the injection of the state transitions.
- [Edward Sumerfield](#) uses closures to inject the state transitions.

Inspired by [RSpec](#) (and also by random episodes of Family Guy) [an example](#) of a unit testing DSL using Groovy

- [Clifton Craig](#) uses Builders in Groovy to create [GSpec](#).

### *When Groovy's not enough*

If you have the need to write your own language completely, consider using a compiler compiler. There are many to choose from, e.g. Antlr, JavaCC, SableCC, Coco/R, Cup/JLex/JFI;ex, BYacc/J, Beaver, etc. See [wikipedia](#) for an interesting list. Some of these can even benefit from Groovy. Here is a groovy example for [JParsec](#).

## Type checking extensions

### Rationale

Since Groovy 2.0, users are allowed to use the optional `@TypeChecked` annotation to activate type checking. In this mode, the compiler becomes more verbose and throws errors for, example, typos, non-existent methods, ... This

comes with a few limitations though, most of them coming from the fact that Groovy remains inherently a dynamic language. For example, you wouldn't be able to use type checking on a markup builder:

```
def builder = new MarkupBuilder(out)
builder.html {
  head { ... }
  body {
    p 'Hello, world!'
  }
}
```

In the previous example, none of the `html`, `head`, `body` or `p` methods exist. This works because Groovy uses a dynamic dispatch and converts those method calls at runtime. There's no limitation about the number of tags that you can use, nor the attributes.

On the other hand, Groovy is also a platform of choice when it comes to implement internal DSLs. The flexible syntax, combined with runtime and compile-time metaprogramming capabilities make Groovy an interesting choice because it allows the programmer to focus on the DSL rather than on tooling or implementation. Since Groovy DSLs are Groovy code, it's easy to have IDE support without having to write a dedicated plugin for example.

In a lot of cases, DSL engines are written in Groovy (or Java) then user code is executed as scripts, meaning that you have some kind of wrapper on top of user logic. The wrapper may consist, for example, in a `GroovyShell` or `GroovyScriptEngine` that performs some tasks transparently before running the script (adding imports, applying AST transforms, extending a base script, ...). Often, user written scripts come to production without testing because the DSL logic comes to a point where **any** user may write code using the DSL syntax. In the end, a user may just ignore that what he writes is actually **code**. This adds some challenges for the DSL implementor, such as securing execution of user code or, in this case, early reporting of errors.

For example, imagine a DSL which goal is to drive a rover on Mars remotely. Sending a message to the rover takes around 15 minutes. If the rover executes the script and fails with an error (say a typo), you have two problems:

- first, feedback comes only after 30 minutes (the time needed for the rover to get the script and the time needed to receive the error)
- second, some portion of the script has been executed and you may have to change the fixed script significantly (implying that you need to know the current state of the rover...)

Type checking extensions is a new mechanism in Groovy 2.1 that will allow the developer of a DSL engine to make those scripts safer by applying the same kind of checks that static type checking allows on regular groovy classes. The principle, here, is to fail early, that is to say fail compilation of scripts as soon as possible, and if possible provide feedback to the user (including nice error messages).

### How does it work?

Since Groovy 2.1.0, the `@TypeChecked` annotation supports an attribute called `extensions`. This parameter takes an array of strings corresponding to a list of type checking extensions scripts. Those scripts are found at **compile time** on classpath. For example, you would write:

```
@TypeChecked(extensions=' /path/to/myextension.groovy' )  
void foo() { ...}
```

In that case, the *foo* methods would be type checked with the rules of the normal type checker completed by those found in the *myextension.groovy* script. Note that while internally the type checker supports multiple mechanisms to implement type checking extensions (including plain old java code), the recommended way is to use those type checking extension scripts.

### A DSL for type checking

The idea behind type checking extensions is to use a DSL to extend the type checker capabilities. This DSL allows you to hook into the compilation process, more specifically the type checking phase, using an "event-driven" API. For example, when the type checker enters a method body, it throws a *beforeVisitMethod* event that you can react to:

```
beforeVisitMethod { methodNode ->  
    println "Entering ${methodNode.name}"  
}
```

#### An example

Imagine that you have this rover DSL at hand. A user would write:

```
robot.move 100
```

If you have a class defined as such:

```
class Robot {  
    Robot move(int qt) { this }  
}
```

The script can be type checked before being executed using the following script:

```
def config = new CompilerConfiguration()
config.addCompilationCustomizers(new
ASTTransformationCustomizer(TypeChecked))
def shell = new GroovyShell(config)
def robot = new Robot()
shell.setVariable('robot', robot)
shell.evaluate(script)
```

Using the compiler configuration above, we can apply `@TypeChecked` transparently to the script. In that case, it would fail at compile time:

```
[Static type checking] - The variable
[robot] is undeclared.
```

Now, we will slightly update the configuration to include the "extensions" parameter:

```
config.addCompilationCustomizers(new
ASTTransformationCustomizer(TypeChecked,
extensions:[ '/path/to/myextension.groovy' ]))
```

Then add the following to the `myextension.groovy` file:

```
unresolvedVariable { var ->
    if ('robot'==var.name) {
        storeType(var, classNodeFor(Robot))
        handled = true
    }
}
```

Here, we're telling the compiler that if an unresolved variable is found and that its name is `robot`, then we can say

that the type of this variable is a *Robot*.

## The API

### AST

The type checking API is a low level API, dealing with the Abstract Syntax Tree. You will have to know your AST well to develop extensions, even if the DSL makes it much easier than just dealing with AST code from plain Java or Groovy.

### Events

The type checker sends the following events, to which an extension script can react:

Event name	Description	
setup	<b>Called When</b>	Called after the type checker finished initialization
	<b>Arguments</b>	
	<b>Usage</b>	<pre>setup {     //     initiali     zation }</pre> <p>Can be used to perform setup of your extension</p>

finish

**Called When**

Called after the type checker completed type checking

**Arguments**

**Usage**

```
finish {  
    //  
    finaliza  
    tion  
    block  
}
```

Can be used to perform additional checks after the type checker has finished its job.

unresolvedVariable

**Called When**

Called when the type checker finds an unresolved variable

**Arguments**

VariableExpression var

**Usage**

```
unresolvedVariable { var  
->  
  if  
  ('foo'==  
var.name  
) {  
  
  storeType  
e(var,  
classNodeFor(List))  
  
  handled  
= true  
  }  
}
```

Allows the developer to help the type checker with user-injected variables.



unresolvedAttribute

**Called When**

Called when the type checker cannot find an attribute on the receiver

**Arguments**

AttributeExpression aex

**Usage**

```
unresolvedAttribute {  
  aex ->  
    if  
      (getType(aex.objectExpression) ==  
       classNodeFor(Foo)) {  
  
        storeType(aex,  
                  classNodeFor(String))  
  
        handled = true  
      }  
}
```

Allows the developer to handle missing attributes

beforeMethodCall

**Called When**

Called before the type checker starts type checking a method call expression

**Arguments**

MethodCall call

**Usage**

```
beforeMethodCall  
{ call  
->  
    if  
    (call  
instance  
of  
StaticMethodCall  
) {  
    //  
    ...  
  
    handled  
= true  
    }  
}
```

Allows you to intercept method calls before the type checker performs its own checks. This is useful if you want to replace the default type checking with a custom one for a limited scope. In that case, you must set the *handled* flag to true, so that the type checker **skips** its own checks.

afterMethodCall

<b>Called When</b>	Called once the type checker has finished type checking a method call
<b>Arguments</b>	MethodCall call
<b>Usage</b>	<pre>afterMethodCall { call -&gt;     //     ... }</pre> <p>Allow you to perform additional checks after the type checker has done its own checks. This is in particular useful if you want to perform the standard type checking tests but also want to ensure additional type safety, for example checking the arguments against each other.</p> <p>Note that <i>afterMethodCall</i> is called even if you did <i>beforeMethodCall</i> and set the <i>handled</i> flag to true.</p>

onMethodSelection

<b>Called When</b>	Called by the type checker when it finds a method appropriate for a method call
<b>Arguments</b>	Expression expr, MethodNode node

## Usage

```
onMethod  
Selectio  
n {  
  expr,  
  node ->  
    if  
    (node.de  
claringC  
lass.nam  
e ==  
'Foo') {  
      //  
calling  
a method  
on  
'Foo',  
let's  
perform  
addition  
al  
checks!  
    }  
}
```

The type checker works by inferring argument types of a method call, then chooses a target method. If it finds one that corresponds, then it triggers this event. It is for example interesting if you want to react on a specific method call, such as entering the scope of a method that takes a closure as argument (as in builders).

Please note that this event may be thrown for various types of expressions, not only method calls (binary expressions for example).

<b>Called When</b>	Called by the type checker when it fails to find an appropriate method for a method call
<b>Arguments</b>	ClassNode receiver, String name, ArgumentListExpression argList, ClassNode[] argTypes, MethodCall call
<b>Usage</b>	

```
methodNotFound {
    receiver
    , name,
    argList,
    argTypes
    , call
    ->
        //
    receiver
    is the
    inferred
    type of
    the
    receiver
        //
    name is
    the name
    of the
    called
    method
        //
    argList
    is the
    list of
    arguments
    the
    method
    was
    called
    with
        //
    argTypes
    is the
    array of
    inferred
    types
    for each
    argument
        //
    call is
    the
    method
    call for
    which we
    couldn't
    find a
    target
    method
}
```

Unlike *onMethodSelection*, this event is sent when the type checker **cannot** find a target method for a method call (instance or static). It gives you the chance to intercept the error before it is sent to the user, but also set the target method.

For this, you need to return a list of `MethodNode`. In most situations, you would either return:

- an empty list, meaning that you didn't find a corresponding method
- a list with exactly one element, saying that there's no doubt about the target method

If you return more than one `MethodNode`, then the compiler would throw an error to the user stating that the method call is ambiguous, listing the possible methods. For convenience, if you want to return only one method, you are allowed to return it directly instead of wrapping it into a list.

## beforeVisitMethod

### Called When

Called by the type checker before type checking a method body

### Arguments

MethodNode node

### Usage

```
beforeVisitMethod {
    methodNode ->

    handled = true
    // tell the type checker we will handle the body by ourselves
}
```

The type checker will call this method before starting to type check a method body. If you want, for example, to perform type checking by yourself instead of letting the type checker do it, you have to set the *handled* flag to true.

This event can also be used to help defining the scope of your extension (for example, applying it only if you are inside method *foo*).

## afterVisitMethod

### Called When

Called by the type checker after type checking a method body

### Arguments

MethodNode node

### Usage

```
afterVisitMethod  
{  
    methodNode ->  
        //  
        ...  
}
```

Gives you the opportunity to perform additional checks after a method body is visited by the type checker. This is useful if you collect information, for example, and want to perform additional checks once everything has been collected.

beforeVisitClass

<b>Called When</b>	Called by the type checker before type checking a class
<b>Arguments</b>	ClassNode node
<b>Usage</b>	<pre>beforeVisitClass {   classNode -&gt;   // ... }</pre> <p>If a class is annotated with <i>@TypeChecked</i>, then before visiting the class, this event will be sent. It is also the case for inner classes defined inside a class annotated with <i>@TypeChecked</i>. It can help you define the scope of your extension, or you can even totally replace the visit of the type checker with a custom type checking implementation. For that, you would have to set the <i>handled</i> flag to true.</p>

afterVisitClass

**Called When**

Called by the type checker after having finished the visit of a type checked class

**Arguments**

ClassNode node

**Usage**

```
afterVisitClass  
{  
  classNode ->  
    //  
    perform  
    additional  
    checks  
}
```

Called for every class being type checked after the type checker finished its work. This includes classes annotated with *@TypeChecked* and any inner/anonymous class defined in the same class with is not skipped.

## incompatibleAssignment

### Called When

Called when the type checker thinks that an assignment is incorrect, meaning that the right hand side of an assignment is incompatible with the left hand side.

### Arguments

ClassNode lhsType,  
ClassNode rhsType,  
Expression assignment

### Usage

```
incompatibleAssignment {  
  lhsType,  
  rhsType,  
  expr ->  
  if  
  (isBinaryExpression(expr)  
  ) &&  
  isAssignment(expr) {  
  ... }  
}
```

Gives the developer the ability to handle incorrect assignments. This is for example useful if a class overrides setProperty, because in that case it is possible that assigning a variable of one type to a property of another type is handled through that runtime mechanism.

In that case, you can help the type checker just by telling it that the assignment is valid (using *handled* set to true).

Of course, an extension script may consist of several blocks, and you can have multiple blocks responding to the same event. This makes the DSL look nicer and easier to write. However, reacting to events is far from sufficient. If you know you can react to events, you also need to deal with the errors, which implies several "helper" methods that will make things easier.

## Working with extensions

### *Support classes*

The DSL relies on a support class called `org.codehaus.groovy.transform.stc.GroovyTypeCheckingExtensionSupport`. This class itself extends `org.codehaus.groovy.transform.stc.TypeCheckingExtension`. Those two classes define a number of "helper" methods that will make working with the AST easier, especially regarding type checking. One interesting thing to know is that you **have access to the type checker**. This means that you can programmatically call methods of the type checker, including those that allow you to **throw compilation errors**.

The extension script delegates to the `GroovyTypeCheckingExtensionSupport` class, meaning that you have direct access to the following variables:

- `context`: the type checker context, of type `org.codehaus.groovy.transform.stc.TypeCheckingContext`
- `typeCheckingVisitor`: the type checker itself, a `org.codehaus.groovy.transform.stc.StaticTypeCheckingVisitor` instance
- `generatedMethods`: a list of "generated methods", which is in fact the list of "dummy" methods that you can create inside a type checking extension using the `newMethod` methods

The type checking context contains a lot of information that is useful in context for the type checker. For example, the current stack of enclosing method calls, binary expressions, closures, ... This information is in particular important if you have to know "where" you are when an error occurs and that you want to handle it.

### *Class nodes*

Handling class nodes is something that needs particular attention when you work with a type checking extension. Compilation works with an abstract syntax tree (AST) and the tree may not be complete when you are type checking a class. This also means that when you refer to types, you must not use class literals such as `String` or `HashSet`, but to class nodes representing those types. This requires a certain level of abstraction and understanding how Groovy deals with class nodes. To make things easier, Groovy supplies several helper methods to deal with class nodes. For example, if you want to say "the type for `String`", you can write:

```
classNodeFor(String)
```

You would also note that there is a variant of `classNodeFor` that takes a `String` as an argument, instead of a `Class`. In general, you should **not** use that one, because it would create a class node for which the name is "String", but without any method, any property, ... defined on it. The first version returns a class node that is *resolved* but the second one returns one that is *not*. So the latter should be reserved for very special cases.

The second problem that you might encounter is referencing a type which is not yet compiled. This may happen more often than you think. For example, when you compile a set of files together. In that case, if you want to say "that variable is of type Foo" but Foo is not yet compiled, you can still refer to the "Foo" class node using `lookupClassNodeFor`.

### *Helping the type checker*

Say that you know that variable *foo* is of type *Foo* and you want to tell the type checker about it. Then you can use the *storeType* method, which takes two arguments: the first one is the node for which you want to store the type and the second one is the type of the node. If you look at the implementation of *storeType*, you would see that it delegates to the type checker equivalent method, which itself does a lot of work to store node metadata. You would also see that storing the type is not limited to variables: you can set the type of any expression.

Likewise, getting the type of an AST node is just a matter of calling *getType* on that node. This would in general be what you want, but there's something that you must understand:

- *getType* returns the **inferred type** of an expression. This means that it will not return, for a variable declared of type *Object* the class node for *Object*, but the inferred type of this variable **at this point of the code** (flow typing)
- if you want to access the origin type of a variable (or field/parameter), then you must call the appropriate method on the AST node

#### *Throwing an error*

**To throw a type checking error, you only have to call the *addStaticTypeError* method which takes two arguments:**

- a *message* which is a string that will be displayed to the end user
- an AST node responsible for the error. It's better to provide the best suiting AST node because it will be used to retrieve the line and column numbers

#### *isXXXExpression*

It is often required to know the type of an AST node. For readability, the DSL provides a special *isXXXExpression* method that will delegate to "x instance of XXXExpression". For example, instead of writing:

```
if (node instanceof BinaryExpression) {  
    ...  
}
```

which requires you to import the *BinaryExpression* class, you can just write:

```
if (isBinaryExpression(node)) {  
    ...  
}
```

#### *Virtual methods*

When you perform type checking of dynamic code, you may often face the case when you know that a method call is valid but there is no "real" method behind it. As an example, take the Grails dynamic finders. You can have a method call consisting of a method named *findByName(...)*. As there's no *findByName* method defined in the bean, the type checker would complain. Yet, you would know that this method wouldn't fail at runtime, and you can even tell what is the return type of this method. For this case, the DSL supports two special constructs that consist of

"virtual methods". This means that you will return a method node that doesn't really exist but is defined in the context of type checking. Three methods exist:

- `newMethod(String name, Class returnType)`
- `newMethod(String name, ClassNode returnType)`
- `newMethod(String name, Callable<ClassNode> return Type)`

**All three variants do the same: they create a new method node which name is the supplied name and define the return type of this method. Moreover, the type checker would add those methods in the *generatedMethods* list (see *isGenerated* below). The reason why we only set a name and a return type is that it is only what you need in 90% of the cases. For example, in the *findByName* example upper, the only thing you need to know is that *findByName* wouldn't fail at runtime, and that it returns a domain class. The *Callable* version of return type is interesting because it defers the computation of the return type when the type checker actually needs it. This is interesting because in some circumstances, you may not know the actual return type when the type checker demands it, so you can use a closure that will be called each time *getReturnType* is called by the type checker on this method node. If you combine this with deferred checks, you can achieve pretty complex type checking including handling of forward references.**

```
newMethod('foo') {
    // each time getReturnType on this method
    node will be called, this closure will be
    called!
    println 'Type checker called me!'
    classNodeFor(Foo) // return type
}
```

Should you need more than the name and return type, you can always create a new *MethodNode* by yourself.

### Scoping

Scoping is very important in DSL type checking and is one of the reasons why we couldn't use a *pointcut* based approach to DSL type checking. Basically, you must be able to define very precisely when your extension applies and when it does not. Moreover, you must be able to handle situations that a regular type checker would not be able to handle, such as forward references:

```
point a(1,1)
line a,b // b is referenced afterwards!
point b(5,2)
```

Say for example that you want to handle a builder:

```
builder.foo {
    bar
    baz(bar)
}
```

Your extension, then, should only be active once you've entered the *foo* method, and inactive outside of this scope. But you could have complex situations like multiple builders in the same file or embedded builders (builders in builders). While you should not try to fix all this from start (you must accept limitations to type checking), the type checker does offer a nice mechanism to handle this: a scoping stack, using the *scopeEnter* and *scopeExit* methods.

- *scopeEnter* creates a new scope and puts it on top of the stack
- *scopeExit* pops a scope from the stack

### A scope consists of:

- a parent scope
- a map of custom data

If you want to look at the implementation, it's simply a *LinkedHashMap* (*org.codehaus.groovy.transform.stc.GroovyTypeCheckingExtensionSupport.TypeCheckingScope*), but it's quite powerful. For example, you can use such a scope to store a list of closures to be executed when you exit the scope. This is how you would handle forward references:

```

def scope = scopeEnter()
...
scope.secondPassChecks << { println
'executed later' }
...
scopeExit {
    secondPassChecks*.run() // execute
deferred checks
}

```

That is to say, that if at some point you are not able to determine the type of an expression, or that you are not able to check at this point that an assignment is valid or not, you can still make the check later... This is a very powerful feature. Now, *scopeEnter* and *scopeExit* provide some interesting syntactic sugar:

```

scopeEnter { // create a new scope
    secondPassChecks = [] // initialize
custom data in this scope (here, a list of
closures to be executed when scopeExit is
called)
}

```

At anytime in the DSL, you can access the current scope using *getCurrentScope()* or more simply *currentScope*. The general schema would be, then:

- determine a "pointcut" where you push a new scope on stack and initialize custom variables within this scope
- using the various events, you can use the information stored in your custom scope to perform checks, defer checks,...
- determine a "pointcut" where you exit the scope, call *scopeExit* and eventually perform additional checks

#### **Other useful methods**

For the complete list of helper methods, please refer to the *org.codehaus.groovy.transform.stc.GroovyTypeCheckingExtensionSupport* and *org.codehaus.groovy.transform.stc.TypeCheckingExtension* classes. However, take special attention to those methods:

- *isDynamic*: takes a *VariableExpression* as argument and returns true if the variable is a *DynamicExpression*, which means, in a script, that it wasn't defined using a type or def.
- *isGenerated*: takes a *MethodNode* as an argument and tells if the method is one that was generated by the

type checker extension using the *newMethod* method

- *isAnnotatedBy*: takes an AST node and a Class (or ClassNode), and tells if the node is annotated with this class. For example: *isAnnotatedBy(node, NotNull)*
- *getTargetMethod*: takes a method call as argument and returns the *MethodNode* that the type checker has determined for it
- *delegatesTo*: emulates the behaviour of the *@DelegatesTo* annotation. It allows you to tell that the argument will delegate to a specific type (you can also specify the delegation strategy)

## F.A.Q

### Is it possible to use compiled scripts?

At the moment, no. The way the DSL is currently implemented disallows it.

### Can I use @Grab in a type checking extension?

Yes, it works perfectly, so you can include libraries that would only be available at compile time. In that case, you must understand that you would increase the time of compilation significantly (at least, the first time it grabs the dependencies).

### How can I share extensions?

A type checking extension is just a script that need to be on classpath. You can share it as is, or bundle it in a jar file (uncompiled).

### Is it possible to activate extensions by default, without having to include them in @TypeChecked parameters?

No. While we thought this could be useful, we want to wait for user feedback before implementing such a feature. Especially, having such a script on classpath could lead to difficult to solve conflicts.

### Can I use an extension to statically compile code that doesn't work with @CompileStatic?

This is an interesting question and the short answer is no. Technically, even if you tell the type checker what is the type of a dynamic variable, for example, it would not know how to compile it. Is it *getBinding('foo')*, *getProperty('foo')*, *delegate.getFoo()*, ... There is absolutely no way of telling the static compiler how to compile such code even if you use a type checking extension (that would, again, only give hints about the type). The preferred way to solve static compilation of this code is to use the good old AST transforms, which are compatible with both *@TypeChecked* and *@CompileStatic*.

### If extensions are not used for static compilation, why does @CompileStatic support extensions too?

Type checking extensions allow you to help the type checker where it fails, but it also allow you to fail where it doesn't. In that context, it makes sense to support extensions for *@CompileStatic* too. Imagine an extension that is capable of type checking SQL queries. In that case, the extension would be valid in both dynamic and static context, because without the extension, the code would still pass.

### Can I include other extensions into an extension?

Currently, no.

Wait, it's a bit painful to use *@TypeChecked(extensions='/path/to/my/extension/in/long/classpath.groovy')*!

Take a look at Groovy 2.1 annotation aliases.

Where can I find examples?

You can download the source code for Groovy and take a look at the [TypeCheckingExtensionsTest](#) class which is linked to [various extension scripts](#).

### Is it possible to update the AST using an extension?

As you have access to the AST, there is nothing in theory that prevents you from modifying the AST. However, we do not recommend you to do so. First of all, you would explicitly break the contract of type checking, which is to annotate, and only annotate the AST. Type checking should **not** modify the AST tree because you wouldn't be able to guarantee anymore that code without the `@TypeChecked` annotation behaves the same without the annotation. Now, if your extension is meant to work with `@CompileStatic`, then you **can** modify the AST because this is indeed what `@CompileStatic` will eventually do. Static compilation doesn't guarantee the same semantics at dynamic Groovy so there is effectively a difference between code compiled with `@CompileStatic` and code compiled with `@TypeChecked`. It's up to you to choose whatever strategy you want to update the AST, but probably using an AST transformation that runs before type checking is easier (because the compiler does some smart bits before reaching bytecode generation).

## Testing Guide

Welcome to the Groovy Testing Guide. It contains information of relevance to those writing developer tests or systems and acceptance tests.

- [Unit Testing](#) — shows how Groovy may be used to simplify unit testing
  - [Using JUnit 4 with Groovy](#)
- [Using Other Testing Frameworks](#) — Examples of integration with a number of Java testing frameworks
  - [Using EasyMock with Groovy](#)
  - [Using Instinct with Groovy](#)
  - [Using JBehave with Groovy](#)
  - [Using JDummy with Groovy](#)
  - [Using JMockit with Groovy](#)
  - [Using JMock with Groovy](#)
  - [Using Popper with Groovy](#)
  - [Using RMock with Groovy](#)
  - [Using TestNG with Groovy](#)
- [Groovy Mocks](#)
  - [Developer Testing using Closures instead of Mocks](#)
  - [Developer Testing using Maps and Expandos instead of Mocks](#)
  - [Mocking Static Methods using Groovy](#)
  - [Using MockFor and StubFor](#) — used to assist (typically unit) testing of classes in isolation
- [GMock](#)
- [Integrating TPTP](#) — some hints for using the Eclipse TPTP facilities with Groovy
- [Model-based testing using ModelJUnit](#)
- [Test Combinations](#) — some hints for using Groovy to assist generate test data in particular all combinations and all pair combinations
  - [Effectiveness of testing combinations with all pairs](#)
- [Test Coverage](#) — is a useful measure of the effectiveness of unit tests and can be derived for Groovy tests
  - [Code Coverage with Cobertura](#)
- [Testing Web Applications](#) — how to use NekoHTML, HtmlUnit, Watij and WebTest to test web applications
- [Testing Web Services](#) — how to test Web Services using Groovy directly and in conjunction with WebTest and SoapUI

- [Using Testing Frameworks with Groovy](#) — explores testing features of Groovy and some other testing frameworks
  - [Using GSpec with Groovy](#)

## Unit Testing

### The Groovy Advantage

Groovy simplifies JUnit testing, making it more Groovy, in several ways, including:

- JUnit is built into the groovy runtime, so you can script JUnit tests for your Groovy and Java classes using Groovy syntax.
- Groovy provides many additional JUnit assertion statements (see below)
- Groovy unit tests are easily scriptable with Ant / Maven (see below)
- Groovy provides [Groovy Mocks](#)

See also:

- [Unit test your java code with groovy](#)
- [testngroove project page](#)
- [Using JUnit 4 with Groovy](#)

### Example

To write unit tests in Groovy, you have to create a class extending `groovy.util.GroovyTestCase`.

```
import groovy.util.GroovyTestCase  
  
class MyTest extends GroovyTestCase {  
    void testSomething() {  
        assert 1 == 1  
        assert 2 + 2 == 4 : "We're in  
trouble, arithmetic is broken"  
    }  
}
```

### Groovy Test Assertions

Apart from the [default assertion methods inherited](#) from the JUnit framework's `TestCase` class, `GroovyTestCase` also offers additional test assertions:

- `assertArrayEquals(Object[] expected, Object[] value)`
- `assertLength(int length, char[] array)`
- `assertLength(int length, int[] array)`
- `assertLength(int length, Object[] array)`
- `assertContains(char expected, char[] array)`
- `assertContains(int expected, int[] array)`

- `assertToString(Object value, String expected)`
- `assertInspect(Object value, String expected)`
- `assertScript(final String script) // assert that a script runs without exceptions`
- `shouldFail(Closure code) // assert that an exception was thrown in that closure`
- `shouldFail(Class clazz, Closure code) // the same but for a class`

## Details

By default Groovy unit test cases generate java bytecode and so are just the same as any other Java unit test cases. One thing to watch is often Ant / Maven look for \*.java files to find unit tests with pattern matching, rather than \*.class files.

There's an option in Maven to ensure you search for classes (and so find any Groovy unit test cases) via this property

```
maven.test.search.classdir = true
```

Once you've got this enabled you can use Maven goals to run individual test cases like this

```
maven test:single  
-Dtestcase=foo.MyGroovyTest
```

## Running GroovyTestCases on the command-line

Since beta-6, you can also run your groovy tests (extending `GroovyTestCase`) on the command-line. It has simple as launching any other Groovy script or class:

```
groovy MyTest.groovy
```

## Running GroovyTestCases in IDEs

Most IDEs support JUnit but maybe don't yet handle Groovy shame! 😊 .

Firstly if you compile the groovy code to bytecode, then it'll just work in any JUnit IDE just fine.

Sometimes though you want to just hack the unit test script and run from in your IDE without doing a build.

If your IDE doesn't automatically recompile Groovy for you then there's a utility to help you run Groovy unit test cases inside any JUnit IDE without needing to run your Ant / Maven build.

The [GroovyTestSuite](#) class is a JUnit TestSuite which will compile and run a GroovyUnit test case from a command line argument (when run as an application) or from the `_test_` system property when run as a JUnit test suite.

To run the `GroovyUnitTest` as an application, just do the equivalent of this in your IDE

```
java groovy.util.GroovyTestSuite  
src/test/Foo.groovy
```

Or to run the test suite inside your IDE, just run the GroovyTestSuite test with this system property defined

```
-Dtest=src/test/Foo.groovy
```

Either of the above can really help improve the development experience of writing Groovy unit test cases in IDEs that don't yet support Groovy natively.

### **Running a TestSuite containing GroovyTestCase scripts directly in Eclipse**

You can take advantage of [GroovyTestSuite](#)'s ability to compile GroovyTestCase scripts into classes to build a TestSuite which can be run from Eclipse's JUnit runner directly.

The suite() method of TestSuite creates and returns a Test. Within your TestSuite's suite() method, you can create a GroovyTestSuite and use it to compile groovy scripts into Class instances, and add them to a TestSuite that you are building using TestSuite.addSuite(Class).

Here's a TestSuite that contains some GroovyTestCase scripts:

```

public class MyTestSuite extends TestSuite {
    // Since Eclipse launches tests
    relative to the project root,
    // declare the relative path to the
    test scripts for convenience
    private static final String TEST_ROOT =
"src/test/com/foo/bar/";
    public static TestSuite suite() throws
Exception {
        TestSuite suite = new TestSuite();
        GroovyTestSuite gsuite = new
GroovyTestSuite();
        suite.addTestSuite(FooTest.class);
// non-groovy test cases welcome, too.

suite.addTestSuite(gsuite.compile(TEST_ROOT
+ "BarTest.groovy"));

suite.addTestSuite(gsuite.compile(TEST_ROOT
+ "FooFactoryTest.groovy"));

suite.addTestSuite(gsuite.compile(TEST_ROOT
+ "BaazTest.groovy"));
        return suite;
    }
}

```

This TestSuite subclass can then be launched as a normal TestSuite in Eclipse. For example, right-click, Run As -> JUnit Test.

From there, the behavior of the JUnit test runner is the same; hierarchy view of all tests and individual methods, their

results, etc.

### Using normal scripts as unit test cases

You can write scripts like this

```
x = [1, 2, 3]  
assert x.size() == 3
```

and use these scripts as unit test cases if you use the `GroovyTestSuite` class to run them as described below.

When the above script is compiled, it doesn't actually implement JUnit's `TestCase` and so needs a special runner so that it can be used inside a JUnit test framework. This is what `GroovyTestSuite` does, it detects scripts like the above and wraps them in a JUnit Test adapter so you can run scripts like the above as a unit test case inside your IDE.

Example of a test suite to test Groovy scripts

```
// RunAllScriptsTests.groovy
import groovy.util.GroovyTestSuite
import junit.framework.Test
import junit.textui.TestRunner
import
org.codehaus.groovy.runtime.ScriptTestAdapte
r

class AllTests {
    static Test suite() {
        def allTests = new GroovyTestSuite()

        allTests.addTest(new
ScriptTestAdapter(allTests.compile("GroovyOK
.groovy"), [] as String[]))

        allTests.addTest(new
ScriptTestAdapter(allTests.compile("GroovyOK
2.groovy"), [] as String[]))

        return allTests
    }
}

TestRunner.run(AllTests.suite())
```

Sample output

```
groovy RunAllScriptsTests.groovy
```

```
..
```

```
Time: 0,015
```

```
OK (2 tests)
```

### New AllTestSuite

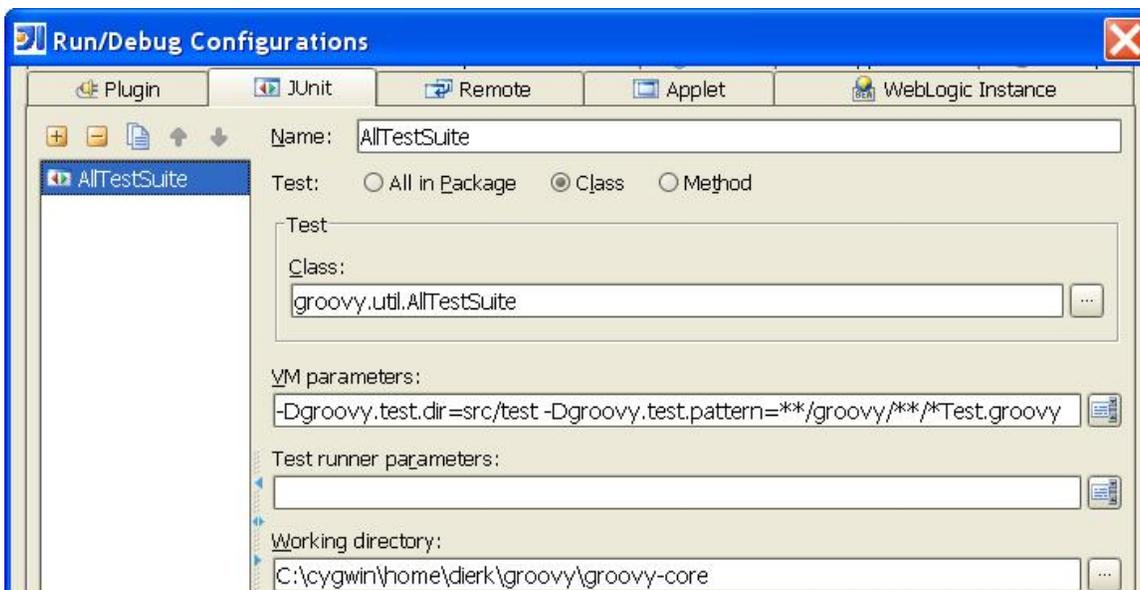
There is a new class `groovy.util.AllTestSuite` that transparently cares for all the above.

Simply make a Run Configuration in your IDE for this class, providing the following System Properties:

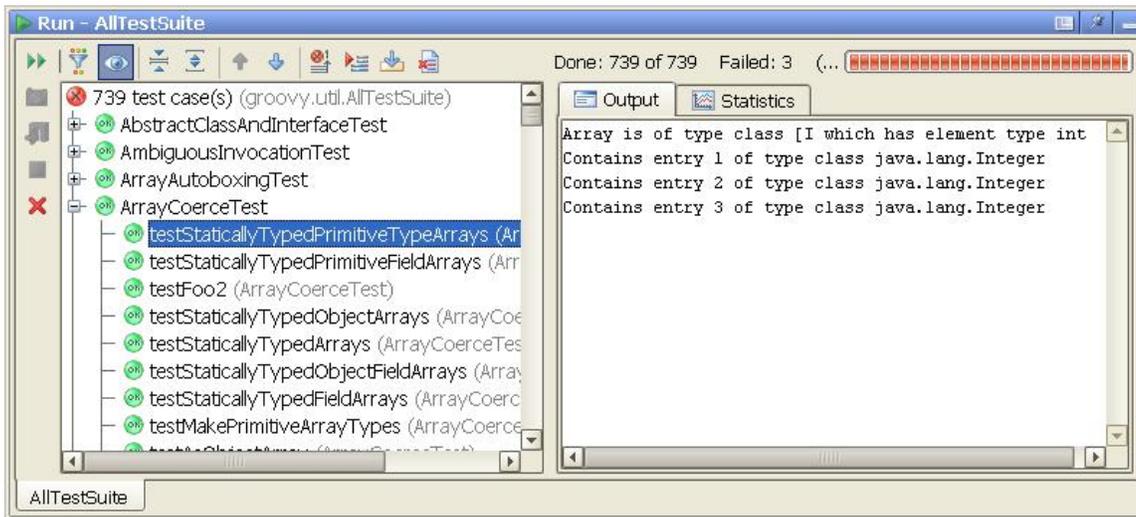
Property name	meaning	default
<code>groovy.test.dir</code>	the directory to search for groovy tests	<code>./test/</code>
<code>groovy.test.pattern</code>	the ant fileset pattern to search below the dir	<code>**/*Test.groovy</code>

See the API documentation of `groovy.util.AllTestSuite` for more details.

Here is the run config for JetBrains IDEA:



Here is how it looks like when running the `AllTestSuite` for the the groovy Unit tests of the Groovy project itself:



## Using JUnit 4 with Groovy

Groovy has excellent support for Testing with Unit Testing (GroovyTestCase adds extra capabilities to TestCase from JUnit 3.8.2) and Mocking capabilities built right in. Currently, there are no special Groovy extensions for JUnit 4 but it's easy to use so long as you are using Groovy 1.5+ and Java 5+ (a requirement for annotations/Junit 4.x). Here are some examples.

Make sure you are using at least Groovy 1.5 (we used 1.5.6) and JUnit 4.x (we used 4.4).

### Simple Example

Let's test some of the built-in arithmetic operators in Groovy.

```

import org.junit.Test
import static org.junit.Assert.assertEquals

class ArithmeticTest {
    @Test
    void additionIsWorking() {
        assertEquals 4, 2+2
    }

    @Test(expected=ArithmeticException)
    void divideByZero() {
        println 1/0
    }
}

```

Alternatively, one could use the `shouldFail` method as follows:

```

class ArithmeticTest {
    final shouldFail = new
GroovyTestCase().&shouldFail

    @Test
    void divideByZero() {
        shouldFail(ArithmeticException) {
            println 1/0
        }
    }
}

```

Our test class includes two tests *additionIsWorking* and *divideByZero*. The second of these is expected to fail with

an *ArithmeticException* exception.

Running the test gives:

```
JUnit version 4.4
```

```
..
```

```
Time: 0.078
```

```
OK (2 tests)
```

#### **Hamcrest Matching**

You can use the Hamcrest matchers that come with JUnit 4.4 and above like this:

```

import static org.junit.Assert.assertThat
import static org.hamcrest.CoreMatchers.*
import static
org.junit.matchers.JUnitMatchers.*
import org.junit.Test

class LanguageTest {
    def languages = [tom:['Java', 'Groovy'],
dick:['C#', 'Ruby']]

    @Test void
tomKnowsJavaAndGroovyHamcrest() {
        assertThat languages['tom'],
is(["Java", "Groovy"])
        assertThat languages['tom'][0],
containsString("v")
        assertThat languages['tom'][1],
containsString("v")
    }

    @Test void tomKnowsJavaAndGroovyNative()
{
        assert languages['tom'] == ["Java",
"Groovy"]
        assert languages['tom'].every{
it.contains("v") }
    }
}

```

The first test uses the matchers. You can see by the second test, that native Groovy assertions are usually going to

do the job just as well if not better in the Groovy world.

#### Parameterized Testing

JUnit 4.x includes the ability to have parameterized tests. Suppose we want to test the following program:

```
class GroovyMultiplier {
    int triple(int val) {
        return val * 3
    }
}
```

Here is what your code might look like. This example uses parameterization.

```
import org.junit.Test
import org.junit.Before
import org.junit.runner.RunWith
import org.junit.runners.Parameterized
import
org.junit.runners.Parameterized.Parameters

@RunWith(Parameterized)
class MultiplierTest {
    def testee
    def param
    def expectedResult

    @Parameters static data() {
        return (2..4).collect{ [it, it *
3] as Integer[] }
    }

    MultiplierTest(a, b) {
```

```
        param = a
        expectedResult = b
    }

    @Before void setUp() {
        testee = new GroovyMultiplier()
    }

    @Test void positivesFixed() {
        assert testee.triple(1) == 3: "+ve
multiplier error"
    }

    @Test void positivesParameterized() {
        assert testee.triple(param) ==
expectedResult
    }

    @Test void negativesParameterized() {
        assert testee.triple(-param) ==
```

```
-expectedResult  
    }  
}
```

The output will look something like this:

```
JUnit version 4.4  
.....  
Time: 0.062  
  
OK (9 tests)
```

#### Theory Tests

You can use the experimental Theory tests from JUnit 4.4 and above as follows:

```
import org.junit.runner.*  
import org.junit.experimental.theories.*  
import static org.junit.Assume.assumeTrue as  
assume  
  
@RunWith(Theories)  
class LanguageTheoryTest {  
    @DataPoint public static String java =  
'Java'  
    @DataPoint public static String ruby =  
'JRuby'  
    @DataPoint public static String python =  
'Jython'  
    @DataPoint public static String  
javascript = 'Rhino'  
    @DataPoint public static String groovy =
```

```

'Groovy'
    @DataPoint public static String scala =
'Scala'
    @DataPoint public static String csharp =
'C#'

    def jvmLanguages = [java, ruby, python,
groovy, scala, javascript]

    def teamSkills = [
        tom:    [java, groovy, ruby],
        dick:   [csharp, scala, java,
python],

        harry: [javascript, groovy, java]
    ]

    @Theory void everyone_knows_java() {
        teamSkills.each { person, skills ->
            assert java in skills
        }
    }

    @Theory void
someone_knows_each_jvm_language(String
language) {
        assume language in jvmLanguages
        assert teamSkills.any { person,
skills ->
            language in skills
        }
    }

```

```
}
```

```
    @Theory void  
    tom_knows_all_languages_ending_with_y(String  
    language) {  
        assume language.endsWith('y')
```

```
        assert language in teamSkills.tom
    }
}
```

When run, this gives:

```
JUnit version 4.4
...
Time: 0.063

OK (3 tests)
```

## Using Other Testing Frameworks

### Using Other Java Testing Frameworks

Sometimes you may wish to consider complementing Groovy's built-in capabilities with one or more of the following frameworks:

- [Using EasyMock with Groovy](#)
- [Using Instinct with Groovy](#)
- [Using JBehave with Groovy](#)
- [Using JDummy with Groovy](#)
- [Using JMockit with Groovy](#)
- [Using JMock with Groovy](#)
- [Using Popper with Groovy](#)
- [Using RMock with Groovy](#)
- [Using TestNG with Groovy](#)

### Using EasyMock with Groovy

[EasyMock](#) is a mocking framework for Java. Here we look at EasyMock 2.2 which requires Java 5 and has the following benefits:

- Hand-writing classes for Mock Objects is not needed.
- Supports refactoring-safe Mock Objects: test code will not break at runtime when renaming methods or reordering method parameters
- Supports return values and exceptions.
- Supports checking the order of method calls, for one or more Mock Objects.

The sections below illustrate using EasyMock for the mocking parts of [Using Testing Frameworks with Groovy](#).

#### *The Item Storer Example*

We are going to consider how you might use EasyMock as part of testing the [Item Storer Example](#).

Here is how we can test JavaStorer:

```
// require(groupId: 'easymock',
artifactId: 'easymock', version='2.2')
import org.easymock.EasyMock

mockControl = EasyMock.createStrictControl()
mockReverser =
mockControl.createMock(Reverser.class)
storer = new JavaStorer(mockReverser)
testStorage()

def testStorage() {
    expectReverse(123.456, -123.456)
    expectReverse('hello', 'olleh')
    mockControl.replay()
    checkReverse(123.456, -123.456)
    checkReverse('hello', 'olleh')
    mockControl.verify()
}

def expectReverse(input, output) {
    // it's a pity mockControl doesn't have
    an expect() method

    EasyMock.expect(mockReverser.reverse(input))
    .andReturn(output)
}

def checkReverse(value, reverseValue) {
    storer.put(value)
```

```
assert value == storer.get()
```

```
    assert reverseValue ==
    storer.getReverse()
}
```

## Using Instinct with Groovy

[Instinct](#) is a Behaviour Driven Development ([BDD](#)) framework for Java. Inspired by [RSpec](#), Instinct provides:

- flexible annotation of contexts, specifications, test actors, etc. (via Java 1.5 annotations, marker interfaces or naming conventions)
- automatic creation of test doubles (dummies, mocks and stubs) and test subjects
- state and behavioural (mocks) expectation API
- JUnit test runner integration
- Ant tasks

The sections below illustrate using Instinct for the examples from [Using Testing Frameworks with Groovy](#).

### *The Stack Example*

Here is how you might use Instinct to test the [Stack Example](#):

```
//
require(url:'http://code.google.com/p/instinct', jar:'instinct-0.1.5.jar')
//
require(url:'http://geekscape.org/static/boost.html', jar:'boost-982.jar')
// require(groupId:'org.jmock',
artifactId:'jmock', version:'2.2.0')
import
com.googlecode.instinct.marker.annotate.*
import
com.googlecode.instinct.runner.TextRunner

class AlmostEmptyFixedStackContext {
    private stack
```

```

@BeforeSpecification
void initially() {
    stack = new FixedStack()
    stack.push 'anything'
    assert !stack.isEmpty()
}

@Specification
void shouldRemainNotEmptyAfterPeek() {
    stack.peek()
    assert !stack.isEmpty()
}

@Specification
void shouldBecomeEmptyAfterPop() {
    stack.pop()
    assert stack.isEmpty()
}
}

class AlmostFullFixedStackContext {
    private stack

    @BeforeSpecification
    void initially() {
        stack = new FixedStack()
        (1..<FixedStack.MAXSIZE).each{ x ->
stack.push x }
        assert !stack.isFull()
    }
}

```

```

    @Specification
    void shouldBeFullAfterPush() {
        stack.push 'anything'
        assert stack.isFull()
    }
}

class EmptyFixedStackContext extends
GroovyTestCase {
    private stack = new FixedStack()

    @BeforeSpecification
    void preConditions() {
        assert stack.isEmpty()
    }

    @Specification
    void shouldNoLongerBeEmptyAfterPush() {
        stack.push 'anything'
        assert !stack.isEmpty()
    }

    @Specification
    void shouldComplainWhenSentPeek() {
        shouldFail(StackUnderflowException)
    {
        stack.peek()
    }
}

    @Specification

```

```
void shouldComplainWhenSentPop() {
    shouldFail(StackUnderflowException)
{
    stack.pop()
    }
}
}
```

```
class FullFixedStackContext extends
GroovyTestCase {
    private stack

    @BeforeSpecification
    void initially() {
        stack = new FixedStack()
        (1..FixedStack.MAXSIZE).each{ x ->
stack.push x }
        assert stack.isFull()
    }

    @Specification
    void shouldRemainFullAfterPeek() {
        stack.peek()
        assert stack.isFull()
    }

    @Specification
    void shouldNoLongerBeFullAfterPop() {
        stack.pop()
        assert !stack.isFull()
    }
}
```

```
@Specification
void shouldComplainOnPush() {
    shouldFail(StackOverflowException) {
        stack.push 'anything'
    }
}
}
```

```
class NonEmptyFixedStackContext {
    private stack
```

```
@BeforeSpecification
```

```
void setUp() {
    stack = new FixedStack()
    ('a'..'c').each{ x -> stack.push x }
    assert !stack.isEmpty()
}
}
```

```
@Specification
```

```
void shouldAddToTheTopWhenSentPush() {
    stack.push 'd'
    assert stack.peek() == 'd'
}
}
```

```
@Specification
```

```
void
shouldBeUnchangedWhenSentPushThenPop() {
    stack.push 'anything'
    stack.pop()
    assert stack.peek() == 'c'
}
```

```
    }

    @Specification
    void
    shouldReturnTheTopItemWhenSentPeek() {
        assert stack.peek() == 'c'
    }

    @Specification
    void
    shouldNotRemoveTheTopItemWhenSentPeek() {
        assert stack.peek() == 'c'
        assert stack.peek() == 'c'
    }

    @Specification
    void shouldReturnTheTopItemWhenSentPop()
    {
        assert stack.pop() == 'c'
    }

    @Specification
    void shouldRemoveTheTopItemWhenSentPop()
    {
        assert stack.pop() == 'c'
        assert stack.pop() == 'b'
    }
}

TextRunner.runContexts(
    AlmostEmptyFixedStackContext,
```

**AlmostFullFixedStackContext,**  
**EmptyFixedStackContext,**

```
FullFixedStackContext,  
NonEmptyFixedStackContext  
)
```

which outputs:

```
AlmostEmptyFixedStackContext  
- shouldBeEmptyAfterPop  
- shouldRemainNotEmptyAfterPeek  
AlmostFullFixedStackContext  
- shouldBeFullAfterPush  
EmptyFixedStackContext  
- shouldComplainWhenSentPeek  
- shouldNoLongerBeEmptyAfterPush  
- shouldComplainWhenSentPop  
FullFixedStackContext  
- shouldComplainOnPush  
- shouldNoLongerBeFullAfterPop  
- shouldRemainFullAfterPeek  
NonEmptyFixedStackContext  
- shouldAddToTheTopWhenSentPush  
- shouldBeUnchangedWhenSentPushThenPop  
- shouldReturnTheTopItemWhenSentPop  
- shouldReturnTheTopItemWhenSentPeek  
- shouldNotRemoveTheTopItemWhenSentPeek  
- shouldRemoveTheTopItemWhenSentPop
```

#### *The Item Storer Example*

Here is how you might use Instinct to integration test the [Item Storer Example](#):

```
//
require(url:'http://code.google.com/p/instinct', jar:'instinct-0.1.5.jar')
//
require(url:'http://geekscape.org/static/boost.html', jar:'boost-982.jar')
// require(groupId:'org.jmock',
artifactId:'jmock', version:'2.2.0')
import
com.googlecode.instinct.marker.annotate.BeforeSpecification as initially
import
com.googlecode.instinct.marker.annotate.Specification as spec
import static
com.googlecode.instinct.runner.TextRunner.runContexts as check_specs_for

class a_default_storer {
    def storer

    @initially void create_new_storer() {
        storer = new Storer()
    }

    private check_persist_and_reverse(value,
expectedReverse) {
        storer.put(value)

        def persisted = storer.get()
        assert persisted == value
    }
}
```

```
    def reversed = storer.reverse
    assert reversed == expectedReverse
  }

  @spec def should_reverse_numbers() {
    check_persist_and_reverse 123.456,
-123.456
  }

  @spec def should_reverse_strings() {
    check_persist_and_reverse 'hello',
'olleh'
  }

  @spec def should_reverse_lists() {
    check_persist_and_reverse([1, 3, 5],
[5, 3, 1])
  }
```

```
}
```

```
check_specs_for a_default_storer
```

We have used some BDD-flavoured method naming conventions here, but they are not compulsory. Here is the output:

```
a_default_storer
- should_reverse_lists
- should_reverse_strings
- should_reverse_numbers
```

### Using JBehave with Groovy

[JBehave](#) is a Behaviour Driven Development ([BDD](#)) framework for Java.

The sections below illustrate using JBehave for the examples from [Using Testing Frameworks with Groovy](#).

#### *The Stack Example*

Here is how you might use JBehave to test the [Stack Example](#):

```
// require(url:'http://jbehave.org/',
jar='jbehave-1.0.1.jar')
import org.jbehave.core.Run
import org.jbehave.core.behaviour.Behaviours

class AlmostEmptyFixedStackBehavior {
    private stack

    void setUp() {
        stack = new FixedStack()
        stack.push 'anything'
        assert !stack.isEmpty()
    }
}
```

```

    void shouldRemainNotEmptyAfterPeek() {
        stack.peek()
        assert !stack.isEmpty()
    }

    void shouldBecomeEmptyAfterPop() {
        stack.pop()
        assert stack.isEmpty()
    }
}

class AlmostFullFixedStackBehavior {
    private stack

    void setUp() {
        stack = new FixedStack()
        (1..<FixedStack.MAXSIZE).each{ x ->
stack.push x }
        assert !stack.isFull()
    }

    void shouldBecomeFullAfterPush() {
        stack.push 'anything'
        assert stack.isFull()
    }
}

class EmptyFixedStackBehavior extends
GroovyTestCase {
    private stack = new FixedStack()

```

```
void shouldInitiallyBeEmpty() {
    assert stack.isEmpty()
}

void shouldNoLongerBeEmptyAfterPush() {
    stack.push 'anything'
    assert !stack.isEmpty()
}

void shouldComplainWhenSentPeek() {
    shouldFail(StackUnderflowException)
{
    stack.peek()
}
}

void shouldComplainWhenSentPop() {
    shouldFail(StackUnderflowException)
{
    stack.pop()
}
}

class FullFixedStackBehavior extends
GroovyTestCase {
    private stack

    void setUp() {
        stack = new FixedStack()
```

```
        (1..FixedStack.MAXSIZE).each{ x ->
stack.push x }
        assert stack.isFull()
    }

void shouldRemainFullAfterPeek() {
    stack.peek()
    assert stack.isFull()
}

void shouldNoLongerBeFullAfterPop() {
    stack.pop()
    assert !stack.isFull()
}

void shouldComplainOnPush() {
    shouldFail(StackOverflowException) {
        stack.push 'anything'
    }
}
}
```

```
class NonEmptyFixedStackBehavior {
    private stack

    void setUp() {
        stack = new FixedStack()
        ('a'..'c').each{ x -> stack.push x }
        assert !stack.isEmpty()
    }
}
```

```
void shouldAddToTheTopWhenSentPush() {
    stack.push 'd'
    assert stack.peek() == 'd'
}

void
shouldBeUnchangedWhenSentPushThenPop() {
    stack.push 'anything'
    stack.pop()
    assert stack.peek() == 'c'
}

void
shouldReturnTheTopItemWhenSentPeek() {
    assert stack.peek() == 'c'
}

void
shouldNotRemoveTheTopItemWhenSentPeek() {
    assert stack.peek() == 'c'
    assert stack.peek() == 'c'
}

void shouldReturnTheTopItemWhenSentPop()
{
    assert stack.pop() == 'c'
}

void shouldRemoveTheTopItemWhenSentPop()
{
    assert stack.pop() == 'c'
}
```

```
        assert stack.pop() == 'b'
    }
}

class AllBehaviours implements Behaviours {
    Class[] getBehaviours() {
        return [
            AlmostEmptyFixedStackBehavior,
            AlmostFullFixedStackBehavior,
            EmptyFixedStackBehavior,
            FullFixedStackBehavior,
            NonEmptyFixedStackBehavior
        ]
    }
}
```

```
}
```

```
Run.main('AllBehaviours')
```

#### *The Item Storer Example*

Here is how you might use JBehave to test the [Item Storer Example](#):

```
// require(url:'http://jbehave.org/',
jar='jbehave-1.0.1.jar')
import org.jbehave.core.Run

class JBehaveStorerBehavior {
    def storer = new Storer()

    def static checkPersistAndReverse(cut,
value, reverseValue) {
        cut.put(value)
        assert value == cut.get()
        assert reverseValue ==
cut.getReverse()
    }

    void shouldReverseStrings() {
        checkPersistAndReverse storer,
'hello', 'olleh'
    }

    void shouldReverseNumbers() {
        checkPersistAndReverse storer,
123.456, -123.456
    }
}
```

```
void shouldReverseLists() {  
    checkPersistAndReverse storer, [1,  
3, 5], [5, 3, 1]  
}
```

```
}
```

```
Run.main('JBehaveStorerBehavior')
```

## Using JDummy with Groovy

[JDummy](#) is a thin API which sits above JMock. It allows very succinct expectation setting code when the expectation code would normally involve many stubs.

The sections below illustrate using JDummy for the mocking parts of [Using Testing Frameworks with Groovy](#).

### *The Item Storer Example*

We are going to consider how you might use JDummy as part of testing the [Item Storer Example](#).

Here is how we can test `JavaStorer`:

```
// require(groupId:'jmock',
artifactId:'jmock', version='1.2.0')
// require(groupId:'jmock',
artifactId:'jmock-cglib', version='1.2.0')
// require(groupId:'junit',
artifactId:'junit', version='3.8.2')
// require(groupId:'cglib',
artifactId:'cglib-nodep',
version='2.2_beta1')
// require(url:'jdummy.sf.net',
jar:'jdummy-1.3.3.jar')
import net.sf.jdummy.JDummyTestCase

class JDummyTest extends JDummyTestCase {
    def mockReverser, storer

    protected void setUp() throws Exception
{
```

```
        mockReverser =
mimicWithDummyValues(Reverser.class)
        storer = new
JavaStorer(mockReverser)
    }

    void testStorage() {
        expectReverse(123.456, -123.456)
        expectReverse('hello', 'olleh')
        checkReverse(123.456, -123.456)
        checkReverse('hello', 'olleh')
    }

    def expectReverse(input, output) {
        // with is a keyword in Groovy so we
quote it

assertBehavior(mockReverser).expects(once())
.method('reverse').'with'(eq(input)).will(re
turnValue(output))
    }

    def checkReverse(value, reverseValue) {
        storer.put(value)
        assert value == storer.get()
        assert reverseValue ==
storer.getReverse()
    }
}
```

```
def suite = new junit.framework.TestSuite()
suite.addTestSuite(JDummyTest.class)
junit.textui.TestRunner.run(suite)
```

Note: Our example is so simple, that JDummy's power is not really illustrated here.

## Using JMockit with Groovy

[JMockit](#) is a single class with a small set of static methods, which allow arbitrary methods and constructors of any other class to be replaced by mock implementations at runtime. It has the following features:

- no particular design must be followed by code under test, e.g.:
  - you don't need to have interfaces everywhere
  - you don't need to avoid static method calls
  - you don't need to use dependency injection, i.e. you can have `new SomeClass()` calls throughout your code
  - you don't need to worry about `final` classes
- legacy code can be unit tested without the need for any adaptation

Since JMockit depends on the JVM class redefinition mechanism exposed by `java.lang.instrumentation`, Groovy, JUnit or TestNG tests that use it must be run under a Java SE 5 VM. However, application and test code can still be compiled to older versions of the language.

The sections below illustrate using JMockit for the mocking parts of [Using Testing Frameworks with Groovy](#).

### *The Item Storer Example*

We are going to consider how you might use JMockit as part of testing the [Item Storer Example](#).

First we define the following class in a file called `MockReverser.groovy`:

```
class MockReverser implements Reverser {
    private testData = [123.456:-123.456,
hello:'olleh']
    def reverse(value) { testData[value] }
}
```

We place this in a separate file rather than straight in the same script as the instrumentation classes in the JVM (which JMockit relies upon) don't know about classes compiled in memory by Groovy. They will look for replaced classes on the classpath.

Now, here is how we can test `Storer`:

```

//
require(url:'https://jmockit.dev.java.net',
jar='jmockit.jar')
//
require(url:'https://jmockit.dev.java.net',
jar='jmockit-asm2.jar')
// needs to be run with
"-javaagent:jmockit.jar"
import mockit.*

def checkReverse(storer, value,
reverseValue) {
    storer.put(value)
    assert value == storer.get()
    assert reverseValue ==
storer.getReverse()
}

Mockit.redefineMethods(GroovyReverser,
MockReverser)
def storer = new Storer()
checkReverse(storer, 123.456, -123.456)
checkReverse(storer, 'hello', 'olleh')

```

If you recall from the example, `Storer` does a new `GroovyReverser()` call, so we use `Mockit.redefineMethods()` to replace the original version with our mock version.

### Using JMock with Groovy

[JMock](#) is a popular mocking framework for Java. Several versions are available.

The sections below illustrate using various versions of JMock for the mocking parts of [Using Testing Frameworks with Groovy](#).

### The Item Storer Example

We are going to consider how you might use JMock as part of testing the [Item Storer Example](#).

Here is how we can test `JavaStorer` using version 1.x of JMock using its JUnit 3 integration:

```
// require(groupId: 'jmock',
artifactId: 'jmock-core', version='1.2.0')
// require(groupId: 'junit',
artifactId: 'junit', version='3.8.2')
import org.jmock.MockObjectTestCase

class JMock1Test extends MockObjectTestCase
{
    def mockControl, mockReverser, storer

    protected void setUp() throws Exception
    {
        mockControl = mock(Reverser.class)
        mockReverser = mockControl.proxy()
        storer = new
JavaStorer(mockReverser)
    }

    void testStorage() {
        expectReverse(123.456, -123.456)
        expectReverse('hello', 'olleh')
        checkReverse(123.456, -123.456)
        checkReverse('hello', 'olleh')
    }

    def expectReverse(input, output) {
        // with is a keyword in Groovy so we
```

quote it

```
mockControl.expects(once()).method('reverse')
    .with(eq(input)).will(returnValue(output))
}

def checkReverse(value, reverseValue) {
    storer.put(value)
    assert value == storer.get()
    assert reverseValue ==
storer.getReverse()
}
}
```

```
def suite = new junit.framework.TestSuite()
suite.addTestSuite(JMock1Test.class)
junit.textui.TestRunner.run(suite)
```

Here is how we can test `JavaStorer` using version 2.x of JMock using its JUnit 4 integration:

```
// require(groupId:'junit',
artifactId:'junit4', version='4.3.1')
// require(groupId:'org.jmock',
artifactId:'jmock', version='2.1.0')
// require(groupId:'org.jmock',
artifactId:'jmock-junit4', version='2.1.0')
import org.jmock.integration.junit4.JMock
import org.jmock.Mockery
import org.junit.Test
import org.junit.Before
import org.junit.runner.RunWith
import org.junit.runner.JUnit4Core

@RunWith(JMock)
class JMock2Test {
    Mockery context = new
JUnit4GroovyMockery()
    def mockReverser, storer

    @Before void setUp() throws Exception {
        mockReverser =
context.mock(Reverser.class)
        storer = new
JavaStorer(mockReverser)
    }
}
```

```
@Test void testStorage() {
    expectReverse(123.456, -123.456)
    expectReverse('hello', 'olleh')
    checkReverse(123.456, -123.456)
    checkReverse('hello', 'olleh')
}

def expectReverse(input, output) {
    context.checking{

one(mockReverser).reverse(input);
will(returnValue(output))
    }
}

def checkReverse(value, reverseValue) {
    storer.put(value)
    assert value == storer.get()
    assert reverseValue ==
storer.getReverse()
}
```

```
}
```

```
JUnitCore.main( ' JMock2Test ' )
```

To make our tests a little more DSL-like, we used the following helper class with JMock 2:

```
import groovy.lang.Closure;  
import org.jmock.Expectations;  
import  
org.jmock.integration.junit4.JUnit4Mockery;  
  
public class JUnit4GroovyMockery extends  
JUnit4Mockery {  
    class ClosureExpectations extends  
Expectations {  
        void closureInit(Closure cl, Object  
delegate) {  
            cl.setDelegate(delegate);  
            cl.call();  
        }  
    }  
  
    public void checking(Closure c) {  
        ClosureExpectations expectations =  
new ClosureExpectations();  
        expectations.closureInit(c,  
expectations);  
        super.checking(expectations);  
    }  
}
```

## Using Popper with Groovy

[Popper](#) extends [JUnit](#) to allow you to specify *theories*. Theories are assertions about your code's behavior that may be true over (potentially) infinite sets of input values. You might find it useful to pose theories about your Groovy code too.

### Example

Let's consider how we might test the following class (example taken from the Popper web site):

```
class Dollar {
    int amount
    Dollar(int amount) { this.amount =
amount }
    Dollar times(int value) { amount *=
value; return this }
    Dollar divideBy(int value) { amount /=
value; return this }
}
```

With traditional JUnit code, we might test it as follows:

```
import org.junit.Test
import org.junit.runner.JUnitCore

class StandardTest {
    @Test void multiplyThenDivide() {
        assert new
Dollar(10).times(6).divideBy(6).amount == 10
    }
}

JUnitCore.main('StandardTest')
```

This tests the method for one `amount` value and one `m` value. Next steps might be to triangulate so that additional

values are also tested. In general though, it might be difficult to know when you have done enough values (when to stop) and also what invariants of your class may hold if you simply keep adding more tests without sufficient refactoring. With these factors in mind, Popper provides facilities to make invariants and preconditions of your classes obvious as well as providing an extensible framework for adding new test values.

Here is how you might use Popper to test the above class. First, we have avoided using [Hamcrest](#) style assertions in our Groovy code. Groovy's built-in `assert` method usually allows such assertions to be expressed very elegantly without any additional framework. We'll create a small helper class to allow Groovy-style assertions to be used for method pre-conditions:

```
import static
net.saff.theories.assertion.api.Requirements
.*
import
net.saff.theories.assertion.api.InvalidTheoryParameterException
import
net.saff.theories.runner.api.TheoryContainer

class GroovyTheoryContainer extends
TheoryContainer {
    def assume(condition) {
        try {
            assert condition
        } catch (AssertionError ae) {
            throw new
InvalidTheoryParameterException(condition,
is(condition))
        }
    }
    def
assumeMayFailForIllegalArguments(Closure c)
{
        try {
```

```
        c.call()
    } catch (IllegalArgumentException e)
{
    throw new
InvalidTheoryParameterException(e, isNull())
```

```
}  
  }  
    }
```

Now, our test becomes:

```
import org.junit.*
import org.junit.runner.*
import net.saff.theories.methods.api.Theory
import net.saff.theories.runner.api.*

@RunWith(Theories)
class PopperTest extends
GroovyTheoryContainer {
    private log = [] // for explanatory
purposes only
    public static int VAL1 = 0
    public static int VAL2 = 1
    public static int VAL3 = 2
    public static int VAL4 = 5

    @Theory void
multiplyIsInverseOfDivide(int amount, int m)
{
    assume m != 0
    assert new
Dollar(amount).times(m).divideBy(m).amount
== amount
    log << [amount, m]
}

    @After void dumpLog() {
        println log
    }
}

JUnitCore.main('PopperTest')
```

We have added an additional log variable to this example to explain how Popper works. By default, Popper will use any public fields in our test as test data values `VAL1` through `VAL4` in our example. It will determine all combinations of the available variables and call the `multiplyIsInverseOfDivide()` for each combination. This is a very crude way to select test instance values but works for simple tests like this one.

You should also note the `assume` statement. In our example, we haven't catered for `m` being 0 which would result in a divide by zero error. The `assume` statement allows this method precondition to be made explicit. When Popper calls the test method, it will silently ignore any test data combinations which fail the method preconditions. This keeps the preconditions obvious and simplifies creating test data sets.

Here is the output from running this test:

```
JUnit version 4.3.1
```

```
.[[0, 1], [0, 2], [0, 5], [1, 1], [1, 2],  
[1, 5], [2, 1], [2, 2], [2, 5], [5, 1], [5,  
2], [5, 5]]
```

```
Time: 0.297
```

```
OK (1 test)
```

We wouldn't normally recommend sending this kind of information to standard out when running your test, but here it is very illustrative. Note that all four test values have been used for the `amount` variable but only three values have been used for `m`. This is exactly what we want here.

Popper supports an extensible framework for specifying more elaborate algorithms for selecting test data. Instead of the public variables, we can define our own parameter supplier. Here is one which supplies data between a `first` value and a `last` value. First the annotation definition (coded in Java):

```

// Java
import
net.saff.theories.methods.api.ParametersSupp
liedBy;
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@ParametersSuppliedBy(BetweenSupplier.class)
public @interface Between {
    int first();
    int last();
}

```

And the backing supplier (coded in Groovy):

```

import net.saff.theories.methods.api.*
import java.util.*

public class BetweenSupplier extends
ParameterSupplier {
    public List getValues(test,
ParameterSignature sig) {
        def annotation =
sig.supplierAnnotation

annotation.first()..annotation.last()
    }
}

```

Now our Groovy test example could become:

```

import org.junit.*
import org.junit.runner.*
import net.saff.theories.methods.api.Theory
import net.saff.theories.runner.api.*

@RunWith(Theories)
class PopperBetweenTest extends
GroovyTheoryContainer {
    private int test, total // for
explanatory purposes only

    @Theory void multiplyIsInverseOfDivide(
        @Between(first = -4, last = 2)
int amount,
        @Between(first = -2, last = 5)
int m
    ) {
        total++
        assume m != 0
        assert new
Dollar(amount).times(m).divideBy(m).amount
== amount
        test++
    }

    @After void dumpLog() {
        println "$test tests performed out
of $total combinations"
    }
}

```

```
}
```

```
JUnitCore.main( ' PopperBetweenTest ' )
```

When run, this yields:

```
JUnit version 4.3.1  
.49 tests performed out of 56 combinations  
  
Time: 0.234  
  
OK (1 test)
```

The supplied test values for the test method are  $(-4, -2)$ ,  $(-4, -1)$ ,  $(-4, 0)$ , ...,  $(2, 5)$ . The data where  $m$  is equal to 0 will be skipped as soon as the `assume` statement is reached.

#### Bowling Example

We can also Groovy to make the [bowling example](#) a little more succinct:

```
import net.saff.theories.methods.api.*  
import net.saff.theories.runner.api.*  
import org.junit.runner.*  
  
@RunWith(Theories.class)  
class BowlingTests extends  
GroovyTheoryContainer {  
    public static Game STARTING_GAME = new  
Game()  
    public static Game NULL_GAME = null  
  
    public static Bowl THREE = new Bowl(3)  
    public static Bowl FOUR = new Bowl(4)
```

```
public static Bowl NULL_BOWL = null
    @DataPoint public Bowl oneHundredBowl()
{ new Bowl(100) }
```

```
public static int ONE_HUNDRED = 100
public static int ZERO = 0
```

```
@Theory
public void
shouldBeTenFramesWithTwoRollsInEach(Game
game, Bowl first, Bowl second) {
    assume game && first && second
    assume game.isAtBeginning()
    assume !first.isStrike()
    assume
!second.completesSpareAfter(first)
    10.times {
        game.bowl(first)
        game.bowl(second)
    }
    assert game.isGameOver()
}
```

```
@Theory
public void maximumPinCountIsTen(Bowl
bowl) {
    assume bowl
    assert bowl.pinCount() <= 10
}
```

```
@Theory
```

```
public void
pinCountMatchesConstructorParameter(int
pinCount) {
    assumeMayFailForIllegalArguments {
        assert new
Bowl(pinCount).pinCount() == pinCount
    }
}
```

```
}
```

```
JUnitCore.main( ' BowlingTests ' )
```

## Using RMock with Groovy

[RMock](#) is a Java mock object framework typically used with JUnit 3.x. RMock has support for a setup-modify-run-verify workflow when writing JUnit tests. It integrates better with IDE refactoring support than some other popular mocking frameworks and allows designing classes and interfaces in a true test-first fashion.

The sections below illustrate using RMock for the mocking parts of [Using Testing Frameworks with Groovy](#).

### *The Item Storer Example*

We are going to consider how you might use RMock as part of testing the [Item Storer Example](#).

Here is how we can test `JavaStorer`:

```
// require(groupId: 'com.agical.rmock',  
artifactId: 'rmock', version='2.0.2')  
// require(groupId: 'junit',  
artifactId: 'junit', version='3.8.2')  
// require(groupId: 'cglib',  
artifactId: 'cglib-nodep',  
version='2.2_beta1')  
import  
com.agical.rmock.extension.junit.RMockTestCa  
se  
  
class RmockTest extends RMockTestCase {  
    def mockReverser, storer  
  
    protected void setUp() throws Exception  
    {  
        mockReverser = mock(Reverser.class,  
        'mockReverser')
```

```
        storer = new
JavaStorer(mockReverser)
    }

    void testStorage() {
        expectReverse(123.456, -123.456)
        expectReverse('hello', 'olleh')
        startVerification()
        checkReverse(123.456, -123.456)
        checkReverse('hello', 'olleh')
    }

    def expectReverse(input, output) {
        mockReverser.reverse(input)
        modify().returnValue(output)
    }

    def checkReverse(value, reverseValue) {
        storer.put(value)
        assert value == storer.get()
        assert reverseValue ==
storer.getReverse()
    }
}
```

```
def suite = new junit.framework.TestSuite()
suite.addTestSuite(RmockTest.class)
junit.textui.TestRunner.run(suite)
```

## Using TestNG with Groovy

[TestNG](#) is a testing framework inspired from JUnit and NUnit but with new functionality to make it more powerful and easier to use. Features include:

- JDK 5 Annotations (JDK 1.4 is also supported with JavaDoc annotations)
- Flexible test configuration
- Support for data-driven testing (with @DataProvider)
- Support for parameters
- Allows distribution of tests on slave machines
- Powerful execution model (no more TestSuite)
- Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...)
- Embeds BeanShell for further flexibility
- Default JDK functions for runtime and logging (no dependencies)
- Dependent methods for application server testing

TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc...

The sections below illustrate using TestNG for the examples from [Using Testing Frameworks with Groovy](#).

### *The Item Storer Example*

Here is how you might use TestNG to integration test the [Item Storer Example](#):

```
// require(groupId: 'org.testng',
artifactId: 'testng', version='5.6')
import org.testng.annotations.*
import org.testng.TestNG
import org.testng.TestListenerAdapter

class StorerIntegrationTest {

    private storer

    @BeforeClass
    def setUp() {
```

```

        storer = new Storer()
    }

    private checkPersistAndReverse(value,
reverseValue) {
        storer.put(value)
        assert value == storer.get()
        assert reverseValue ==
storer.getReverse()
    }

    @Test
    void shouldPersistAndReverseStrings() {
        checkPersistAndReverse 'hello',
'olleh'
    }

    @Test
    void shouldPersistAndReverseNumbers() {
        checkPersistAndReverse 123.456,
-123.456
    }

    @Test
    void shouldPersistAndReverseLists() {
        checkPersistAndReverse([1, 3, 5],
[5, 3, 1])
    }
}

def testng = new TestNG()

```

```
testng.setTestClasses(StorerIntegrationTest)
```

```
testng.addListener(new
TestListenerAdapter())
testng.run()
```

You might also like to consider the special Groovy integration from the [Test'N'Groove](#) project which provides command-line runners and ant tasks for using TestNG with Groovy.

## Groovy Mocks

### Groovy Mocks

Groovy has excellent built-in support for a range of mocking alternatives. Before considering those, let's review some relevant terms.

#### Terms

##### *Collaborator*

An ordinary Groovy or Java class that's instance or class methods are to be called. Calling them can be time consuming or produce side effects that are unwanted when testing (e.g. database operations).

##### *Caller*

A Groovy Object that calls methods on the Collaborator, i.e. collaborates with it.

##### *Mock*

An object that can be used to augment the Collaborator. Method calls to the Collaborator will be handled by the Mock, showing a *demanded* behavior. Method calls are *expected* to occur *strictly* in the *demanded* sequence with a given *range* of cardinality. The use of a Mock implicitly ends with *verifying* the expectations.

##### *Stub*

Much like a Mock but the *expectation* about sequences of method calls on the Collaborator is *loose*, i.e. calls may occur out of the *demanded* order as long as the *ranges* of cardinality are met. The use of a Stub does not end with an implicit verification since the stubbing effect is typically asserted on the Caller. An explicit call to *verify* can be issued to assert all *demanded* method calls have been effected with the specified cardinality.

## An extended example

### *System under test*

We will explore a system under test inspired from the [JBehave](#) currency example.

Our system makes use of a base currency class used to represent the currency of a particular country:

```
class Currency {
    public static final Currency USD = new
Currency("USD")
    public static final Currency EUR = new
Currency("EUR")
    private String currencyCode
    private Currency(String currencyCode) {
        this.currencyCode = currencyCode
    }
    public String toString() {
        return currencyCode
    }
}
```

and a base exchange rate class which encapsulates buying and selling rates for a currency:

```
class ExchangeRate {
    final double fromRate
    final double toRate
    public ExchangeRate(double fromRate,
double toRate) {
        this.fromRate = fromRate
        this.toRate = toRate
    }
}
```

We will make use of an exchange rate service collaborator to retrieve the exchange rates for a particular country:

```
interface ExchangeRateService {  
    ExchangeRate retrieveRate(Currency c)  
}
```

Our class under test is a currency converter. It makes use of the following exception:

```
class InvalidAmountException extends  
Exception {  
    public InvalidAmountException(String  
message) {super(message);}  
}
```

and conforms to the following interface:

```
interface CurrencyConverter {  
    double convertFromSterling(double  
amount, Currency toCurrency) throws  
InvalidAmountException  
    double convertToSterling(double amount,  
Currency fromCurrency) throws  
InvalidAmountException  
}
```

Here is our class under test.

```
class SterlingCurrencyConverter implements  
CurrencyConverter {  
    private ExchangeRateService  
exchangeRateService
```

```
public
SterlingCurrencyConverter(ExchangeRateService
exchangeRateService) {
    this.exchangeRateService =
exchangeRateService;
}
```

```
private double convert(double amount,
double rate) throws InvalidAmountException {
    if (amount < 0) {
        throw new
InvalidAmountException("amount must be
non-negative")
    }
    return amount * rate
}
```

```
public double convertFromSterling(double
amount, finance.Currency toCurrency) throws
InvalidAmountException {
    return convert(amount,
exchangeRateService.retrieveRate(toCurrency)
.fromRate)
}
```

```
public double convertToSterling(double
amount, finance.Currency fromCurrency)
throws InvalidAmountException {
    return convert(amount,
exchangeRateService.retrieveRate(fromCurrenc
```

**y) .toRate)**

```
    }  
}
```

### ***Mocking using Map coercion***

When using Java, Dynamic mocking frameworks are very popular. A key reason for this is that it is hard work creating custom hand-crafted mocks using Java. Such frameworks can be used easily with Groovy if you choose (as shown in this [extended example](#)) but creating custom mocks is much easier in Groovy. You can often get away with simple maps or closures to build your custom mocks.

Let's consider maps first.

By using maps or expandos, we can incorporate desired behaviour of a collaborator very easily as shown here:

```
def service = [retrieveRate:{ new  
ExchangeRate(1.45, 0.57) }] as  
ExchangeRateService  
def sterlingConverter = new  
SterlingCurrencyConverter(service)  
double convertedAmount =  
sterlingConverter.convertFromSterling(10.0,  
Currency.USD);  
assert convertedAmount == 14.50
```

For more details, see [Developer Testing using Maps and Expandos instead of Mocks](#).

### ***Mocking using Closure coercion***

Alternatively, we can use closures:

```
service = { new ExchangeRate(1.55, 0.56) }
as ExchangeRateService
sterlingConverter = new
SterlingCurrencyConverter(service)
convertedAmount =
sterlingConverter.convertFromSterling(10.0,
Currency.USD);
assert convertedAmount == 15.50
```

For more details, see [Developer Testing using Closures instead of Mocks](#).

#### ***Mocking using MockFor and StubFor***

If we need the full power of a dynamic mocking framework, Groovy has a built-in framework which makes use of meta-programming to define the behaviour of the collaborator. An example is shown here:

```

import groovy.mock.interceptor.*

def mockContextClass = new
MockFor(DummyExchangeRateService)
mockContextClass.demand.retrieveRate { new
ExchangeRate(1.65, 0.55) }
class DummyExchangeRateService implements
ExchangeRateService {
    ExchangeRate retrieveRate(Currency
currency){}
}
mockContextClass.use {
    def dummyService = new
DummyExchangeRateService()
    sterlingConverter = new
SterlingCurrencyConverter(dummyService)
    convertedAmount =
sterlingConverter.convertFromSterling(10.0,
Currency.USD)
    assert convertedAmount == 16.50
}

```

This approach works well for testing Groovy classes. In the current versions of Groovy (Groovy 1.5 and 1.6 beta1 at the time of writing this page), the behavior that you define with `demand` clauses represents the behavior of all of the instances of the mocked class type. For more details, see [Using MockFor and StubFor](#).

#### *Instance-style MockFor and StubFor*

You can also use `MockFor` and `StubFor` in a more traditional style by creating instances as follows:

```
mockContext1 = new
MockFor(ExchangeRateService)
mockContext1.demand.retrieveRate { new
ExchangeRate(1.75, 0.54) }
def dummyService1 =
mockContext1.proxyInstance()
def sterlingConverter1 = new
SterlingCurrencyConverter(dummyService1)
convertedAmount1 =
sterlingConverter1.convertFromSterling(10.0,
Currency.USD)
mockContext1.verify dummyService1
assert convertedAmount1 == 17.50
```

```
mockContext2 = new
MockFor(ExchangeRateService)
mockContext2.demand.retrieveRate(1){ new
ExchangeRate(1.85, 0.53) }
def dummyService2 =
mockContext2.proxyInstance()
def sterlingConverter2 = new
SterlingCurrencyConverter(dummyService2)
convertedAmount2 =
sterlingConverter2.convertFromSterling(10.0,
Currency.USD)
mockContext2.verify dummyService2
assert convertedAmount2 == 18.50
```

This approach lets you have multiple instances of the same class all with different behaviors. Also, note that you have to explicitly call the `verify` method here if you want to check that the *demanded* behavior was in fact

observed. Also, you can use this technique for testing Java classes but you need to call `proxyDelegateInstance()` instead of `proxyInstance()`.

### Developer Testing using Closures instead of Mocks

Groovy's 'as' operator can be used with closures in a neat way which is great for developer testing in simple scenarios. We haven't found this technique to be so powerful that we want to do away with dynamic mocking, but it can be very useful in simple cases none-the-less.

Suppose we are using [Interface Oriented Design](#) and as sometimes advocated we have defined a number of short interfaces as per below. (Note: we ignore the discussion about whether interfaces are as valuable a design approach when using dynamic languages that support duck-typing.)

```
interface Logger { def log(message) }  
interface Helper { def doSomething(param) }  
interface Factory { Helper getInstance() }
```

Now, using a coding style typically used with [dependency injection](#) (as you might use with [Spring](#)), we might code up an application class as follows:

```
class MyApp {  
    private factory  
    private logger  
    MyApp(Factory factory, Logger logger) {  
        this.logger = logger  
        this.factory = factory  
    }  
    def doMyLogic(param) {  
  
        factory.getInstance().doSomething(param)  
        logger.log('Something done with: ' +  
param)  
    }  
}
```

To testing this, we could use [Groovy's built-in mocking](#) or some other Java-based dynamic mocking framework. Alternatively, we could write our own static mocks. But no one does that these days I hear you say! Well, they never

had the ease of using closures, which bring dynamic power to static mocks, so here we go:

```
def param = 'DUMMY STRING'
def logger = { message -> assert message ==
  'Something done with: ' + param}
def helper = { assert it == param }
def factory = { helper as Helper }
def myApp = new MyApp(factory as Factory,
  logger as Logger)
myApp.doMyLogic(param)
```

That was easy. Behind the scenes, Groovy creates a proxy object for us that implements the interface and is backed by the closure.

Easy yes, however, the technique as described above assumes our interfaces all have one method. What about more complex examples? Well, the 'as' method works with Maps of closures too. Suppose our helper interface was defined as follows:

```
interface Helper {
    def doSomething(param)
    def doSomethingElse(param)
}
```

And our application modified to use both methods:

```

...
    def doMyLogic(param) {
        def helper = factory.getInstance()
        helper.doSomething(param)
        helper.doSomethingElse(param)
        logger.log('Something done with: ' +
param)
    }
...

```

We simply use a map of closures with the key used being the same name as the methods of the interface, like so:

```

...
def helperMethod = { assert it == param }
def helper = [doSomething:helperMethod,
doSomethingElse:helperMethod]
// as before
def factory = { helper as Helper }
...

```

Still easy!

For this simple example, where we wanted each method to be the same (i.e. implementing the same code) we could have done away with the map altogether, e.g. the following would work, making each method be backed by the closure:

```

def factory = { helperMethod as Helper }

```

#### *More Information*

See also:

- [Developer Testing using Maps and Expandos instead of Mocks](#)
- [Groovy way to implement interfaces](#)

## Developer Testing using Maps and Expandos instead of Mocks

Suppose we are trying to test the following application:

```
class MyApp {  
    def factory  
    def logger  
    def doBusinessLogic(param) {  
        def myObj = factory.instance  
        myObj.doSomething(param)  
        myObj.doSomethingElse(param)  
        logger.log('Something done with: ' +  
param)  
    }  
}
```

We might be tempted to replace logger and factory with Groovy's [built-in mocks](#), but it turns out that Maps or Expandos are often sufficiently powerful enough in Groovy that we don't need full blown dynamic mocks for this example.

Instead of a mock for logger, we can use an Expando as follows:

```
...  
    def logger = new Expando()  
    logger.log = { msg -> assert msg ==  
'Something done with: ' + param }  
...  

```

Here the expected behaviour for our production code is captured within a Closure. (When using TDD, this closure would force the production code we saw in our original code to be created.)

Instead of a mock for factory, we can use a simple map as follows:

```
...  
    def factory = [instance: businessObj]  
...
```

Here, `businessObj` is the object we want the factory to return, though in general this could be a Closure similar to what we did for the `Expando` above.

Putting this altogether yields (after some refactoring) the following complete test:

```
class MyAppTest extends GroovyTestCase {  
    void testDoesBusinessLogic() {  
        // triangulate  
        checkDoesBusinessLogic "case1"  
        checkDoesBusinessLogic "case2"  
    }  
    private checkDoesBusinessLogic(param) {  
        def logger =  
setUpLoggingExpectations(param)  
        def businessObj =  
setUpBusinessObjectExpectations(param)  
        def factory = [instance:  
businessObj]  
        def cut = new MyApp(logger:logger,  
factory:factory)  
        cut.doBusinessLogic(param)  
    }  
    private setUpLoggingExpectations(param)  
{  
        def shouldProduceCorrectLogMessage =  
            { msg -> assert msg ==  
'Something done with: ' + param }  
    }  
}
```

```
        def logger = new Expando()
        logger.log =
shouldProduceCorrectLogMessage
        return logger
    }
    private
setUpBusinessObjectExpectations(param) {
    def shouldBeCalledWithInputParam = {
assert it == param }
        def myObj = new Expando()
        myObj.doSomething =
shouldBeCalledWithInputParam
        myObj.doSomethingElse =
shouldBeCalledWithInputParam
```

```
        return myObj
    }
}
```

See also: [Developer Testing using Closures instead of Mocks](#)

## Mocking Static Methods using Groovy

Suppose you wish to test a class which is dependent on a static call. Is there a way to mock out that call?

Yes, there are two suggested approaches:

- Using EMC
- Using JMockit (requires Java 6)

### Using EMC

Here we are calling `Arrays.sort()` directly - normally that would be the problematic code within your class under test.

```
// non-mocked first to prove it works
normally
String[] things = [ 'dog', 'ant', 'bee',
'cat' ]
Arrays.sort(things)
println things // => {"ant", "bee", "cat",
"dog"}

Arrays.metaClass.'static'.sort = { a ->
    a[0] = 'ewe'
    a[1] = 'fox'
}

things = [ 'dog', 'ant', 'bee', 'cat' ]
Arrays.sort(things)
println things // => {"ewe", "fox", "bee",
"cat"}
```

More details about this approach: [ExpandoMetaClass - Adding static methods](#)

#### Using JMockit

If you are in a position to use Java 6, you should also consider using JMockit.



```
//
require(url:'https://jmockit.dev.java.net',
jar='jmockit.jar')
//
require(url:'https://jmockit.dev.java.net',
jar='jmockit-asm2.jar')
// needs to be run with
"-javaagent:jmockit.jar"
// and
"-Xbootclasspath/a:jmockit.jar;.;jmockit-asm
2.jar"
// The bootclasspath option is only required
because we are mocking
// a class from the java.* package (part of
the bootclasspath for Java)

import mockit.Mockit

// non-mocked first to show normal case
String[] things = [ 'dog', 'ant', 'bee',
'cat' ]
Arrays.sort(things)
println things // => {"ant", "bee", "cat",
"dog"}

Mockit.redefineMethods(Arrays, MockArrays)

things = [ 'dog', 'ant', 'bee', 'cat' ]
Arrays.sort(things)
println things // => {"dog", "elk", "ape",
"cat"}
```

Where MockArrays is the following Java class:

```
// Java
public class MockArrays {
    public static void sort(Object[] a) {
        a[1] = "elk";
        a[2] = "ape";
    }
}
```

We use a Java class here because otherwise JMockit tries to replace other GroovyObject methods (e.g. getMetaClass, invokeMethod, ...) and won't find them inside the `java.util.Arrays` class. Obviously, if you're redefining a Groovy class, you can use another Groovy class.

More details: [Using JMockit with Groovy](#).

## Using MockFor and StubFor

### Groovy Mocks

Mock objects are used to assist (typically unit) testing of classes in isolation.

The Groovy Mock resides in the `groovy.mock.interceptor` package. It is an all-groovy mock testing library.

In principle, it is used like this:

```

import groovy.mock.interceptor.MockFor

def mocker = new MockFor(Collaborator.class)
// create the Mock support
mocker.demand.one(1..2) { 1 }
// demand the 'one' method one or two times,
returning 1
mocker.demand.two()      { 2 }
// demand the 'two' method exactly once,
returning 2
mocker.use {
// start using the Mock
    def caller = new Caller()
// caller will call Collaborator
    assertEquals 1, caller.collaborateOne()
// will call Collaborator.one
    assertEquals 1, caller.collaborateOne()
// will call Collaborator.one
    assertEquals 2, caller.collaborateTwo()
// will call Collaborator.two
}
// implicit verify for strict expectation
here

```

Groovy Mocks were inspired by [EasyMock](#).

Find background information about Mocks and endo-testing under [XP2000 conference paper](#).

For an extended example, see [Using Testing Frameworks with Groovy](#)

#### Features

- typical mock style of *failing early*
- mocks instance and class methods
- mocks final methods and final Collaborators

- mocks Groovy and Java Collaborators (but Caller must be groovy)
- can mock all objects of a given class (or a single Groovy object)
- mocks even if Collaborator cannot be injected into the Caller
- mocks even if Collaborator is not accessible on the Caller (no getter)
- demanded calls specified via recording calls on the Demand object (EasyMock style).
- cardinality specified as Ranges, default is 1..1; 'optional' can be achieved with 0..1
- behavior specified via Closures, allowing static or calculated return values, throwing exceptions, asserting argument values, etc. (even tricky sequence constraints by sharing state in the testMethod scope between the behavior Closures)
- matching parameter list specified via Closure's parameter list, supporting typed or untyped params, default params, and varargs.
- not dependent on any external mock library

For an extensive list of usages see the [unit tests that show how to use the mock package](#).

#### MockFor Tests

Some tests to demonstrate how to use the MockFor class.

```

import grails.test.*
import groovy.mock.interceptor.*

class MockForTests extends GroovyTestCase {

    void
    test_mock_single_method_and_use_as_category(
    ) {
        def mock = new
MockFor(MockForTestsClass)
        mock.demand.ameethod { "from mock" }
        mock.use {
            assertEquals "from mock", new
MockForTestsClass().ameethod()
        }
    }

    void

```

```

test_mock_single_method_using_demand_with_an
d_use_as_category() {
    def mock = new
MockFor(MockForTestsClass)
    mock.demand.with {
        amethod() { "from mock"}
    }

    mock.use {
        assertEquals "from mock", new
MockForTestsClass().amethod()
    }
}

void
test_mock_single_static_method_and_use_as_ca
tegory() {
    def mock = new
MockFor(MockForTestsClass)
    mock.demand.astaticmethod { "from static
mock"}
    mock.use {
        assertEquals "from static mock",
MockForTestsClass.astaticmethod()
    }
}

void test_mock_method_with_parameters() {
    def mock = new
MockFor(MockForTestsClass)
    mock.demand.amethodwithparameters { a ->

```

```

"from mock with parameters ${a}" }

    mock.use {
        assertEquals "from mock with
parameters 123",
            new
MockForTestsClass().amethodwithparameters(12
3)
    }
}

// This is a special case problem related
to a bug. This proves the work around.
// The bug is that you can not mock the
"find" method unless you pass in the
// cardinality as well.
void
test_mocking_a_method_called_find_is_a_speci
al_case() {
    def mock = new
MockFor(MockForTestsClass)
    mock.demand.find(1) { "cardinality 1" }
    mock.use {
        assertEquals "cardinality 1",
MockForTestsClass.find()
    }
}

// This illustrates a down side of
MockFor. Despite the fact that the MockFor
constructor

```

```
// is passed the type that it is replacing
it does no verification that the demands
actually
```

```
// match the signatures of the methods in
the real implementation.
```

```
void
test_methods_do_not_have_to_exist_on_a_class
_for_them_to_be_mocked() {
```

```
    def mock = new MockFor(Object)
    mock.demand.aMethodThatDoesNotExist {
"SEE IT WORKS ANYWAY" }
```

```
        mock.use {
            assert "SEE IT WORKS ANYWAY" == new
Object().aMethodThatDoesNotExist()
        }
    }
```

```
// This illustrates a down side of
MockFor. Despite the fact that the MockFor
constructor
```

```
// is passed the type that it is replacing
it does no verification that the demands
actually
```

```
// match the signatures of the methods in
the real implementation.
```

```
void
test_demand_the_wrong_number_of_arguments_is
_allowed() {
    def mock = new
```

```

MockFor(MockForTestsClass)
    mock.demand.amethodwithparameters { a,
extraArg -> "blindly called, changes in
implementation ignored" }

    mock.use {
        assertEquals "blindly called, changes
in implementation ignored",
            new
MockForTestsClass().amethodwithparameters(12
3, 123)
    }
}

void
test_mock_single_method_and_use_proxy() {
    def mock = new
MockFor(MockForTestsClass)
        mock.demand.amethod { "from mock with
proxy" }

        def proxy = mock.proxyInstance()
        assertEquals "from mock with proxy",
proxy.amethod()
        mock.verify proxy
    }

void
test_create_proxy_instance_with_constructor_
arguments() {
    def mock = new

```

```

MockFor(MockForTestClassWithConstructorArgs)
    mock.demand.amethod { "from mock with
proxy" }

    def proxy =
mock.proxyInstance(["value1", "value2"]as
Object[])
    assertEquals "from mock with proxy",
proxy.amethod()
    mock.verify proxy
}

void test_mocking_a_constructor_call() {
    def mock = new
MockFor(MockForTestsClass, true)
    mock.demand.with {
        MockForTestsClass() { throw new
Exception("YES IT WORKED") }
    }

    mock.use {
        shouldFail Exception, { new
MockForTestsClass() }
    }
}

// Constructor mocking requires that an
instance of a class is returned
// and that instance can be cast to the
mocked type.
void

```

```

test_mock_a_constructor_that_gets_used() {
    def mock = new
MockFor(MockForTestsClass, true)
    mock.demand.with {
        MockForTestsClass() { new Object() as
GroovyObject }
    }

    mock.use { new MockForTestsClass() }
}

// If you return the wrong type from this
mocked constructor you will get
// an error like this which is a little
hard to work out:
//
//     java.lang.ClassCastException:
java.lang.String cannot be cast to
groovy.lang.GroovyObject
//
void
test_mock_a_constructor_that_returns_the_wro
ng_type() {
    def mock = new
MockFor(MockForTestsClass, true)
    mock.demand.with {
        MockForTestsClass() { "THIS IS THE
WRONG TYPE BEING RETURNED" }
    }

    shouldFail ClassCastException, {

```

```

        mock.use { new MockForTestsClass() }
    }
}

// To demand more interactions with the
object beyond the constructor the mocked
// constructor must return the proxy
instance instead.
//
// NOTE: MockFor constructs an instance of
the real class when creating the proxy
// instance. This will require that the
required parameters are passed in to the
// proxyInstance call for forwarding to
the real constructor.
void
test_mock_a_constructor_that_returns_a_proxy
_instance() {
    def mock = new
MockFor(MockForTestsClass, true)

    def theProxyInstance =
mock.proxyInstance()

    mock.demand.with {
        MockForTestsClass() { theProxyInstance
    }

    amethod() { }
}

mock.use {

```

```
        def testClass= new MockForTestsClass()  
        testClass.amethod()  
    }  
}  
}
```

```
class MockForTestsClass {
```

```
    void amethod() {  
        println "original method"  
    }
```

```
    static void astaticmethod() {  
        println "original static method"  
    }
```

```
    void amethodwithparameters(message) {  
        println "original method with  
parameters"  
    }  
}
```

```
class MockForTestClassWithConstructorArgs  
extends MockForTestsClass {  
    def arg1  
    def arg2  
    MockForTestClassWithConstructorArgs(arg1,  
arg2) {  
        this.arg1 = arg1
```

```
        this.arg2 = arg2
    }
}
```

#### StubFor Tests

Some tests to demonstrate how to use the StubFor class.

```
import groovy.mock.interceptor.*

class StubForTest extends GroovyTestCase {

    void
    test_stub_single_method_no_different_from_mock() {
        def mock = new
        StubFor(StubForTestsClass)
        mock.demand.amethod { "from stub" }
        mock.use {
            assertEquals "from stub", new
            StubForTestsClass().amethod()
        }
    }

    // While we can call stubbed methods many
    // times we still have to specify
    // a cardinality higher than our expected
    // call count.
    void
    test_stub_single_method_but_call_it_many_times() {
        def mock = new
```

```

StubFor(StubForTestsClass)
    mock.demand.amethod(9999) { "from stub"
}

    mock.use {
        assertEquals "from stub", new
StubForTestsClass().amethod()
        assertEquals "from stub", new
StubForTestsClass().amethod()
    }
}

void
test_stub_single_method_dont_call_it_and_exp
ect_no_errors() {
    def mock = new
StubFor(StubForTestsClass)
        mock.demand.amethod { "not called" }
        mock.use { new StubForTestsClass() }
    }
}

class StubForTestsClass {
    void amethod() {

```

```
        println "original method"
    }
}
```

## GMock

### GMock

**GMock is a mocking framework for the Groovy language.**

[GMock](#) is all about simple syntax and readability of your tests so you spend less time learning the framework and more writing code. To use GMock just drop the gmock jar file in your classpath.

### Getting Started

First extend the `org.gmock.GMockTestCase`. Create mock object using the `mock()` method. You setup expectation simply by calling method on your mock.

```
def mockLoader = mock()
mockLoader.load("fruit").returns("apple")
```

The code you are testing should be executed within the `play` closure.

```
void testBasic(){
    // create mock and setup expectation
    play {
        // run your code
    }
}
```

### In a nutshell

- **Method mocking:** `mockLoader.load("fruit").returns("apple")`
- **Exception mocking:** `mockLoader.load("unknown").raises(new RuntimeException())`
- **Stub mocking:** `mockLoader.load("fruit").returns("apple").stub()`
- **Static method mocking:** `mockMath.static.random().returns(0.5)`
- **Property mocking:** `mockLoader.name.returns("loader")`
- **Constructor mocking:** `def mockFile = mock(File, constructor('/a/path/file.txt'))`
- **Partial mocking:** `mock(controller).params.returns(id: 3)`

- **Times expectation:** mockLoader.load("fruit").returns("apple").atLeastOnce()
- **Custom matcher:** mockLoader.load(match{ it.startsWith("fru") })
- **Strict ordering:** ordered { ... }
- **Optional support for Hamcrest matcher:** mockLoader.put("test", is(not(lessThan(5))))
- **GMockController** if you can't extend GMockTestCase in your test

Visit the website for a full documentation: [GMock](#)

## Integrating TPTP

This page provides some hints for using the Eclipse TPTP facilities with Groovy.

The Eclipse Testing and Performance Tools Platform ([TPTP](#)) project addresses the entire test and performance life cycle, from early testing to production application monitoring, including test editing and execution, monitoring, tracing and profiling, and log analysis. It is primarily aimed at the Java-aware tester but can also be used with Groovy.

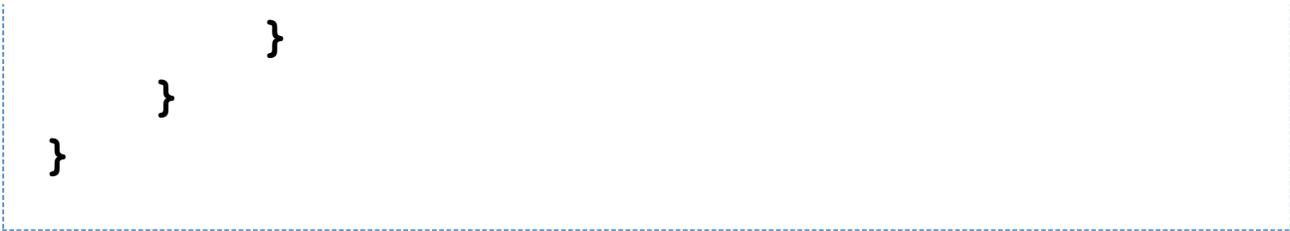
As one example, if you follow the introductory tutorial included in the TPTP documentation called: [Creating a datapool driven JUnit test application](#), you will create a datapool application which lets you invoke data-driven JUnit tests. If you simply replace the Java code with Groovy code you can have data-driven Groovy tests. Here is what your test might look like:

```
import
org.eclipse.hyades.models.common.datapool.im
pl.Common_DatapoolFactoryImpl;

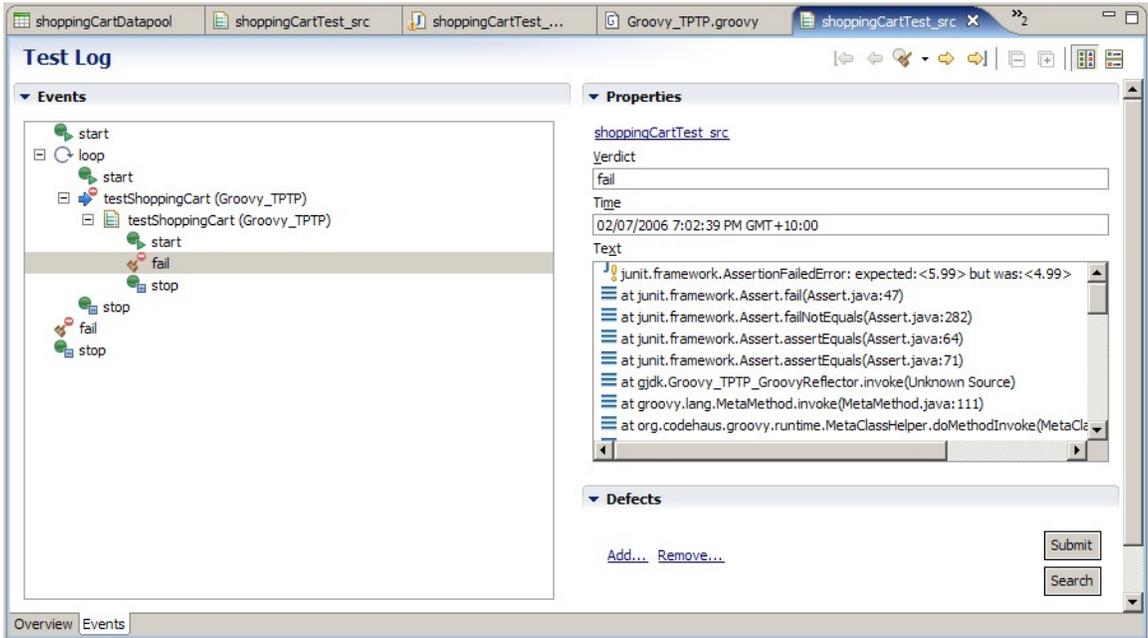
class Groovy_TPTP extends GroovyTestCase {
    def dpIterator

    void setUp() throws Exception {
        def dpFactory = new
Common_DatapoolFactoryImpl()
        def datapool = dpFactory.load(new
java.io.File("GIA_TPTP\\shoppingCartDatapool
.datapool"), false)
        dpIterator =
dpFactory.open(datapool, "org.eclipse.hyades.
datapool.iterator.DatapoolIteratorSequential
Private")
        dpIterator.dpInitialize(datapool, 0)
    }
}
```

```
}  
  
void testShoppingCartConstructor() {  
    def cart = new MyShoppingCart()  
    while(!dpIterator.dpDone()) {  
        def description =  
dpIterator.dpCurrent().getCell("Description"  
).stringValue  
        def expectedPrice =  
dpIterator.dpCurrent().getCell("Price").doub  
leValue  
        def actualPrice =  
cart.myFlowers[description]  
        assertNotNull(actualPrice)  
        assertEquals(expectedPrice,  
actualPrice.doubleValue())  
        dpIterator.dpNext()  
    }  
}
```



Here is what your output might look like:



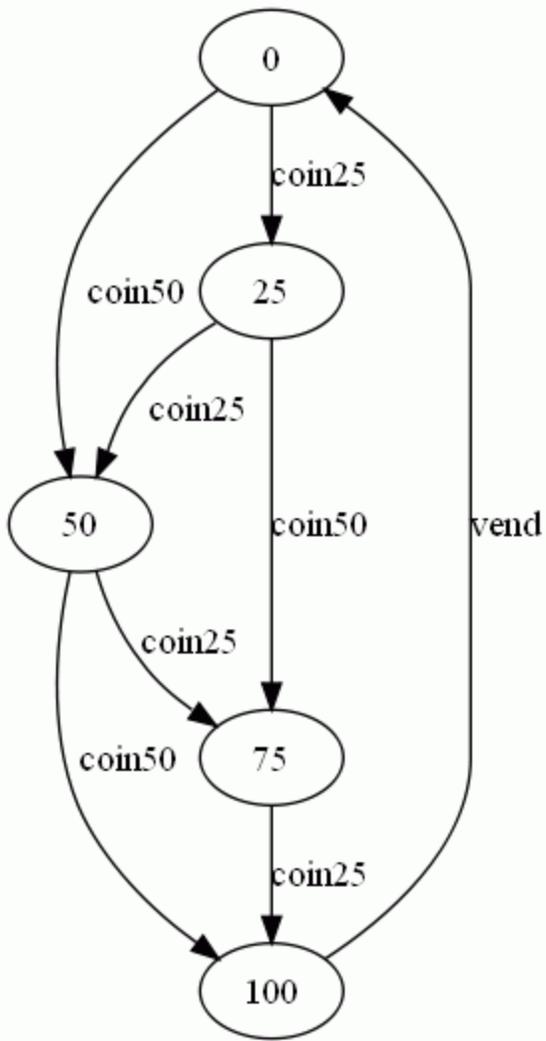
In addition to allowing you to write datapool-aware Groovy tests, by making use of `GroovyTestSuite` and `AllTestSuite` you can visually select Groovy test cases to run within TPTP using its looping and selection features. This lets non-Groovy aware team members create new test cases from primitives which Groovy-aware team members can create.

For more details, see [GINA](#) or the [TPTP](#) web site.

## Model-based testing using ModelJUnit

[ModelJUnit](#) is a Java library that extends JUnit to support [model-based testing](#).

In this example, we will consider how to model and test a simple vending machine. The system under test is illustrated by the following state diagram:



The `VendingMachineModel` class models this system. Our script makes use of the `RandomTester` class to generate some tests for this model. The total code looks like this:

```

// require junit.jar
// require modeljunit.jar

import net.sourceforge.czt.modeljunit.*
import
net.sourceforge.czt.modeljunit.coverage.*

class VendingMachineModel implements
FsmModel {
    def state = 0 // 0,25,50,75,100

    void reset(boolean testing) {state = 0}

    boolean vendGuard() {state == 100}
    @Action void vend() {state = 0}

    boolean coin25Guard() {state <= 75}
    @Action void coin25() {state += 25}

    boolean coin50Guard() {state <= 50}
    @Action void coin50() {state += 50}
}

def tester = new RandomTester(new
VendingMachineModel())
tester.buildGraph()
tester.addListener "verbose"
tester.generate 20

```

When we run this script, we will see the randomly generated test transitions. The output will be:

```
done (0, coin50, 50)
done (50, coin25, 75)
done (75, coin25, 100)
done (100, vend, 0)
done Random reset(true)
done (0, coin25, 25)
done (25, coin50, 75)
done (75, coin25, 100)
done (100, vend, 0)
done (0, coin25, 25)
done (25, coin50, 75)
done (75, coin25, 100)
done (100, vend, 0)
done (0, coin50, 50)
done (50, coin50, 100)
done (100, vend, 0)
done (0, coin25, 25)
done (25, coin50, 75)
done (75, coin25, 100)
done (100, vend, 0)
```

We can optionally obtain additional metrics about the test:

```

// ... as before to create the tester

def metrics = [
  new ActionCoverage(),
  new StateCoverage(),
  new TransitionCoverage(),
  new TransitionPairCoverage()
]
metrics.each {
  tester.addCoverageMetric it
}

tester.addListener "verbose"
tester.generate 20

println 'Metrics Summary:'
metrics.each {
  tester.model.printMessage "$it.name was
$it"
}

```

This generates the following:

```

Metrics Summary:
Action Coverage was 3/3
State Coverage was 5/5
Transition Coverage was 7/8
Transition-Pair Coverage was 9/12

```

We can also add a graph listener with the following code:

```
def graphListener =
tester.model.getListener("graph")
graphListener.printGraphDot "vending.dot"
println "Graph contains " +
graphListener.graph.numVertices() +
    " states and " +
graphListener.graph.numEdges() + "
transitions."
```

When run, a [graphviz](#) dot file is produced (corresponding to the image above) and the output is:

```
Graph contains 5 states and 8 transitions.
```

See also: [State Pattern](#)

## Test Combinations

This page provides some hints for using Groovy to assist generate test data in particular all combinations and all pair combinations.

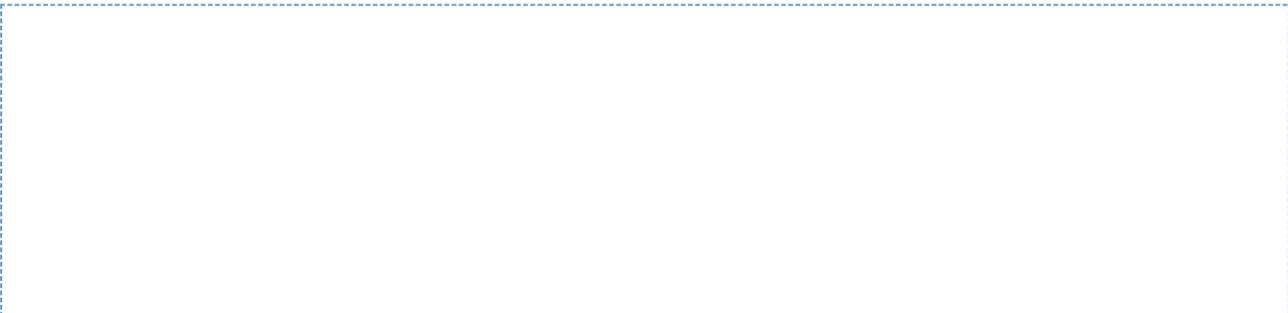
Frequently you may have to test combinations, e.g. a method has several enumerations for its arguments or a web page has several dropdowns with multiple values, or you have assorted hardware combinations (as in the example below). You could manually work out the combinations, or you can let Groovy help you. We are going to look at two ways groovy can help you:

- By generating your test cases in Groovy
- By reading and invoking XML data produced by a specialist tool called Whitch

The most ([effective](#)) way to test large numbers of combinations is called "all pairs".

## Combinations algorithms in Groovy

Here is a script which calculates all combinations for a particular example:



```

results = new HashSet()
def buildCombinations(Map
partialCombinations, inputsLeft) {
  def first =
inputsLeft.entrySet().toList().get(0)
  def partialResults = [ ]
  first.value.each{
    def next = [(first.key):it]
    next.putAll(partialCombinations)
    partialResults << next
  }
  if (inputsLeft.size() == 1) {
    results.addAll(partialResults)
  } else {
    partialResults.each{
      rest = inputsLeft.clone()
      rest.remove(first.key)
      buildCombinations(it, rest)
    }
  }
}

def configurations = [memory:['256M',
'512M', '1G', '2G'],
                      disk:['5G', '10G'],
                      os:['MacOS',
'Windows', 'Linux']]

buildCombinations([:], configurations)
println results.size() + " combinations:"
results.each{ println it }

```

Running this script yields the following results:

**24 combinations:**

```
[ "memory": "512M", "os": "MacOS", "disk": "5G" ]
[ "memory": "2G", "os": "Linux", "disk": "5G" ]
[ "memory": "1G", "os": "Linux", "disk": "10G" ]
[ "memory": "512M", "os": "Linux", "disk": "5G" ]
[ "memory": "512M", "os": "Windows",
  "disk": "10G" ]
[ "memory": "2G", "os": "MacOS", "disk": "10G" ]
[ "memory": "1G", "os": "Windows", "disk": "5G" ]
[ "memory": "256M", "os": "MacOS", "disk": "5G" ]
[ "memory": "1G", "os": "Linux", "disk": "5G" ]
[ "memory": "2G", "os": "Windows",
  "disk": "10G" ]
[ "memory": "512M", "os": "MacOS",
  "disk": "10G" ]
[ "memory": "256M", "os": "Windows",
  "disk": "5G" ]
[ "memory": "256M", "os": "Windows",
  "disk": "10G" ]
[ "memory": "1G", "os": "MacOS", "disk": "10G" ]
[ "memory": "1G", "os": "Windows",
  "disk": "10G" ]
[ "memory": "512M", "os": "Windows",
  "disk": "5G" ]
[ "memory": "256M", "os": "Linux",
  "disk": "10G" ]
[ "memory": "2G", "os": "Windows", "disk": "5G" ]
[ "memory": "2G", "os": "MacOS", "disk": "5G" ]
```

```
["memory": "2G", "os": "Linux", "disk": "10G"]  
["memory": "256M", "os": "Linux", "disk": "5G"]  
["memory": "1G", "os": "MacOS", "disk": "5G"]  
["memory": "256M", "os": "MacOS",
```

```
"disk": "10G"]  
["memory": "512M", "os": "Linux",  
"disk": "10G"]
```

Note: you would normally invoke your test method rather than just printing out the test case as we have done here.

You could then use this information as the input for a data-driven test.

It turns out though, that running all of these combinations is often overkill. If for instance, some bug occurs when memory is low on Windows, then both of the following test cases will illustrate the bug:

```
["memory": "256M", "os": "Windows",  
"disk": "5G"]  
["memory": "256M", "os": "Windows",  
"disk": "10G"]
```

A technique known as [all pairs](#) or orthogonal array testing suggests using just a subset of the input data combinations with high likelihood of finding all bugs and greatly reduced test execution time.

To calculate all pairs for the above example, you could use the following script:

```
initialResults = new HashSet()  
results = new HashSet()  
  
def buildPairs(Map partialCombinations,  
inputsLeft) {  
  def first = getFirstEntry(inputsLeft)  
  def partialResults = [ ]  
  first.value.each{  
    def next = [(first.key):it]  
    def nextEntry = getFirstEntry(next)  
    next.putAll(partialCombinations)  
    partialResults << next  
  }  
}
```

```

    if (inputsLeft.size() == 1) {

initialResults.addAll(partialResults)
    } else {
        partialResults.each{
            rest = inputsLeft.clone()
            rest.remove(first.key)
            buildPairs(it, rest)
        }
    }
}

def adjustPairs() {
    results = initialResults.clone()
    initialResults.each {
        def rest = results.clone()
        rest.remove(it)
        if (allPairsCovered(it, rest)) {
            results.remove(it)
        }
    }
}

def getFirstEntry(Map map) {
    return map.entrySet().toList().get(0)
}

def getAllPairsFromMap(map) {
    if (map.size() <= 1) return null
    def allPairs = new HashSet()
    def first = getFirstEntry(map)

```

```

def rest = map.clone()
rest.remove(first.key)
rest.each{
    def nextPair = new HashSet()
    nextPair << first
    nextPair << it
    allPairs << nextPair
}
def restPairs = getAllPairsFromMap(rest)
if (restPairs != null) {
    allPairs.addAll(restPairs)
}
return allPairs
}

```

```

boolean allPairsCovered(candidate,
remaining) {
    def totalCount = 0
    def pairCombos =
getAllPairsFromMap(candidate)
    pairCombos.each { candidatePair ->
        def pairFound = false
        def pairs = candidatePair.toList()
        for (it in remaining) {
            def entries = it.entrySet()
            if (!pairFound &&
entries.contains(pairs[0]) &&
entries.contains(pairs[1])) {
                pairFound = true
                totalCount++
            }
}
}

```

```
        }
    }
    return (totalCount == pairCombos.size())
}

def updateUsedPairs(map) {
    getAllPairsFromMap(map).each{ usedPairs
<< it }
}

def configurations = [memory:['256M',
'512M', '1G', '2G'],
                    disk:['5G', '10G'],
                    os:['MacOS',
'Windows', 'Linux']]

buildPairs([:], configurations)
```

```
adjustPairs()  
println results.size() + " pairs:"  
results.each{ println it }
```

This code is not optimised. It builds all combinations and then removes unneeded pairs. We could greatly reduce the amount of code by restructuring our all combinations example and then calling that - but we wanted to make each example standalone. Here is the result of running this script:

```
12 pairs:  
["memory":"1G", "os":"Linux", "disk":"5G"]  
["memory":"512M", "os":"MacOS",  
"disk":"10G"]  
["memory":"256M", "os":"Windows",  
"disk":"10G"]  
["memory":"1G", "os":"Windows",  
"disk":"10G"]  
["memory":"512M", "os":"Windows",  
"disk":"5G"]  
["memory":"2G", "os":"MacOS", "disk":"5G"]  
["memory":"2G", "os":"Windows", "disk":"5G"]  
["memory":"2G", "os":"Linux", "disk":"10G"]  
["memory":"1G", "os":"MacOS", "disk":"5G"]  
["memory":"256M", "os":"Linux", "disk":"5G"]  
["memory":"512M", "os":"Linux",  
"disk":"10G"]  
["memory":"256M", "os":"MacOS",  
"disk":"10G"]
```

We saved half of the combinations. This might not seem like much but when the number of input items is large or the number of alternatives for each input data is large, the saving can be substantial.

If this is still too many combinations, we have one more additional algorithm that is sometimes useful. We call it minimal pairs. Rather than making sure each possible pair combination is covered, minimal pairs only adds a new

test data item into the results if it introduces a new pair combination not previously seen. This doesn't guarantee all pairs are covered but tends to produce a minimal selection of interesting pairs across the possible combinations. Here is the script:

```
results = new HashSet()
usedPairs = new HashSet()

def buildUniquePairs(Map
partialCombinations, inputsLeft) {
  def first = getFirstEntry(inputsLeft)
  def partialResults = [ ]
  first.value.each{
    def next = [(first.key):it]
    def nextEntry = getFirstEntry(next)
    next.putAll(partialCombinations)
    partialResults << next
  }
  if (inputsLeft.size() == 1) {
    partialResults.each {
      if (!containsUsedPairs(it)) {
        updateUsedPairs(it)
        results << it
      }
    }
  } else {
    partialResults.each{
      rest = inputsLeft.clone()
      rest.remove(first.key)
      buildUniquePairs(it, rest)
    }
  }
}
```

```
def getFirstEntry(map) {  
    return map.entrySet().toList().get(0)  
}
```

```
def getAllPairsFromMap(map) {  
    if (map.size() <= 1) return null  
    def allPairs = new HashSet()  
    def first = getFirstEntry(map)  
    def rest = map.clone()  
    rest.remove(first.key)  
    rest.each{  
        def nextPair = new HashSet()  
        nextPair << first  
        nextPair << it  
        allPairs << nextPair  
    }  
    def restPairs = getAllPairsFromMap(rest)  
    if (restPairs != null) {  
        allPairs.addAll(restPairs)  
    }  
    return allPairs  
}
```

```
boolean containsUsedPairs(map) {  
    if (map.size() <= 1) return false  
    def unpaired = true  
    getAllPairsFromMap(map).each {  
        if (unpaired) {  
            if (usedPairs.contains(it))  
                unpaired = false  
        }  
    }  
}
```

```
        }
    }
    return !unpaired
}

def updateUsedPairs(map) {
    getAllPairsFromMap(map).each{ usedPairs
<< it }
}

def configurations = [memory:['256M',
'512M', '1G', '2G'],
                    disk:['5G', '10G'],
                    os:['MacOS',
'Windows', 'Linux']]
```

```
buildUniquePairs([:], configurations)
println results.size() + " pairs:"
results.each{ println it }
```

Here is the result of running the script:

```
6 pairs:
["memory": "512M", "os": "Windows",
"disk": "5G"]
["memory": "256M", "os": "MacOS", "disk": "5G"]
["memory": "2G", "os": "Linux", "disk": "10G"]
["memory": "1G", "os": "Linux", "disk": "5G"]
["memory": "512M", "os": "MacOS",
"disk": "10G"]
["memory": "256M", "os": "Windows",
"disk": "10G"]
```

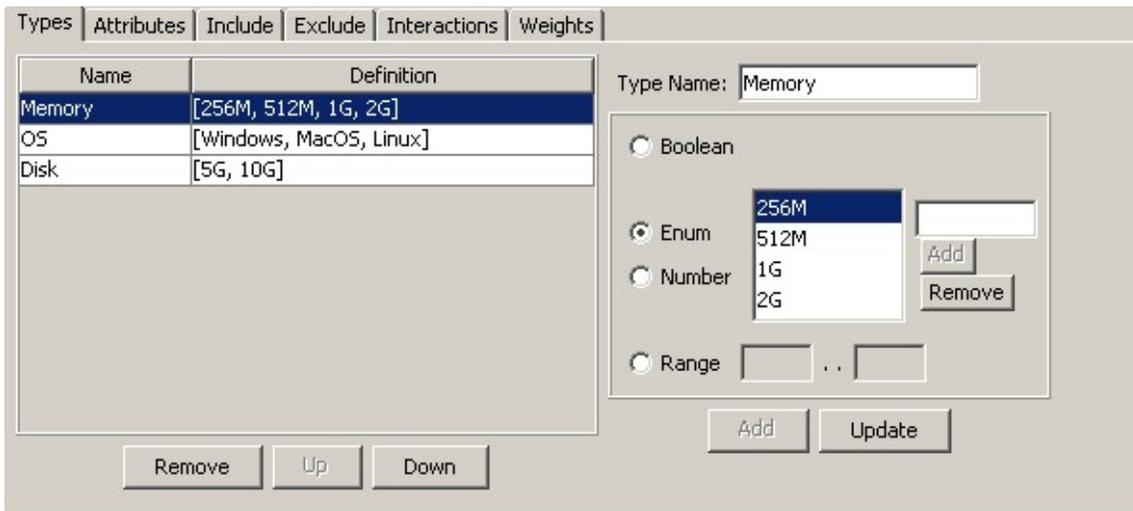
### Combinations using Whitch

The IBM alphaworks site hosts the Intelligent Test Case Handler ([WHITCH](#)) project. From their website: *This technology is an Eclipse plug-in for generation and manipulation of test input data or configurations. It can be used to minimize the amount of testing while maintaining complete coverage of interacting variables. Intelligent Test Case Handler enables the user to generate small test suites with strong coverage properties, choose regression suites, and perform other useful operations for the creation of systematic software test plans.*

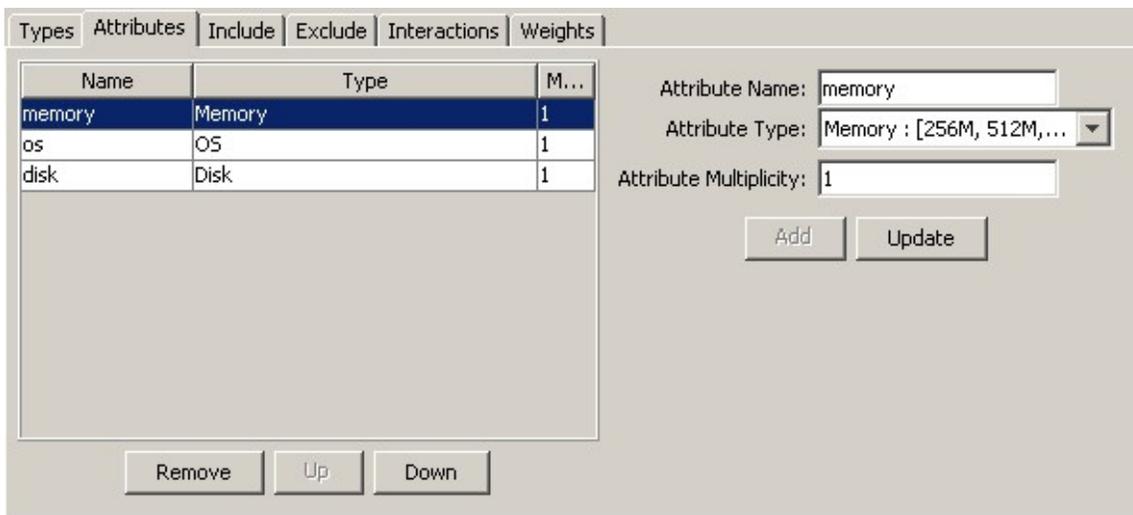
You should follow the instructions on the WHITCH site for installing the plug-in into your Eclipse directory (we used Eclipse 3.2). The user guide is then part of Eclipse help and details the features and instructions for using the tool. We will simply highlight how you might decide to use it.

First create a new Whitch file. File -> New -> Other... -> Whitch file. You must select the project and provide a file name. We used 'groovy.whitch'.

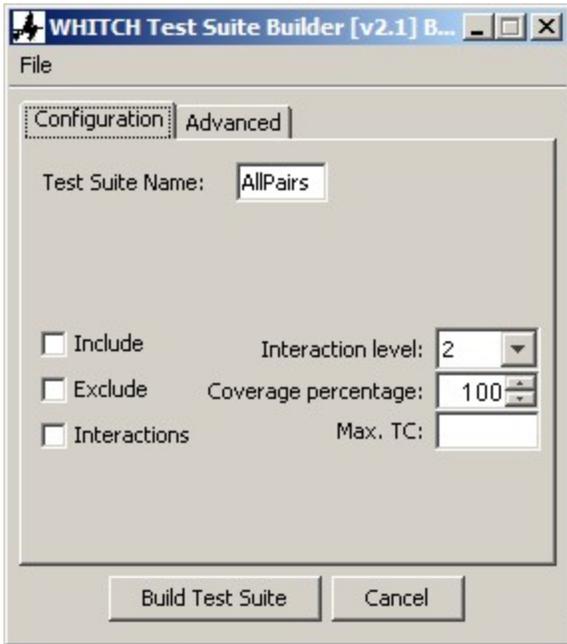
Now add your types similar to the earlier example. You should end up with something like:



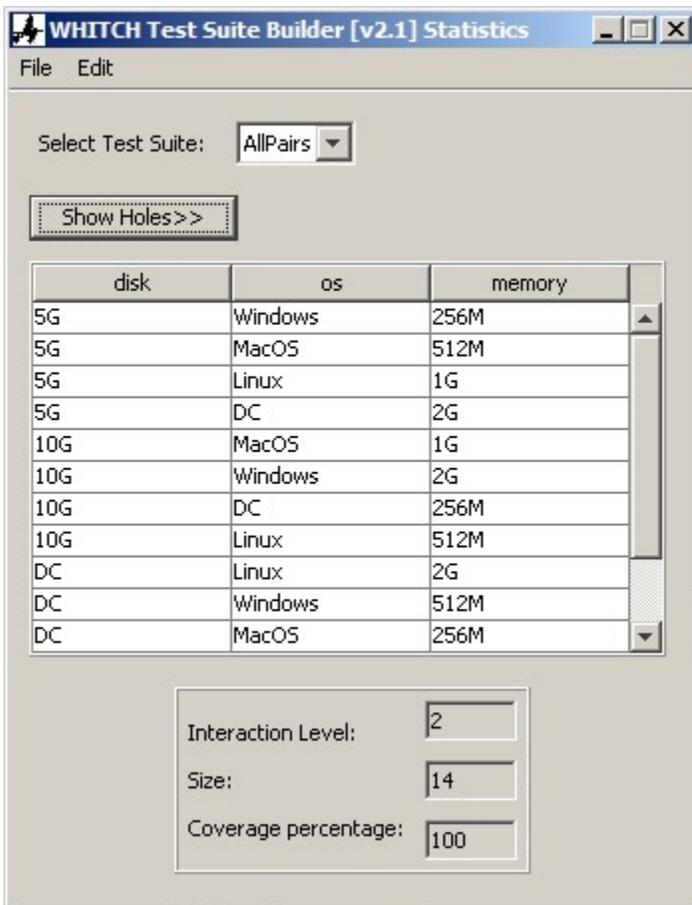
Now add some attributes corresponding to the types you have just added. The result will be something like:



Now select Which -> Build. Type in a test suite name. We used 'AllPairs' and select an interaction level of 2 (for pairs) as follows:



Now click 'Build Test Suite' to obtain your results:



Now save your Whitch file. The result will be that the test cases are now stored into an XML file.

Note: We have not shown any advanced Whitch features, e.g. it lets you add test cases in the Include tab which must always be added into the test suite and test cases which are not possible into the Exclude tab. It also lets you

try to reduce your test case size, give weightings, choose different algorithms for test case generation and more. See the Whitch user guide for more details.

```
def testsuite = new XmlSlurper().parse(new
File('groovy.whitch'))
def attributes =
testsuite.Model.Profile.Attribute
def testcases = testsuite.TestCases.TestCase
println testcases.size() + ' pairs:'
testcases.each{ testcase ->
    def map = [:]
    (0..2).each{
        def key =
attributes[it]. '@name'.toString()
        def value =
testcase.Value[it]. '@val'.toString()
        map.put(key, value)
    }
    println map
}
```

Running this script yields the following results:

14 pairs:

```
[ "memory": "256M", "os": "Windows",  
  "disk": "5G" ]  
[ "memory": "512M", "os": "MacOS", "disk": "5G" ]  
[ "memory": "1G", "os": "Linux", "disk": "5G" ]  
[ "memory": "2G", "os": "DC.DC", "disk": "5G" ]  
[ "memory": "1G", "os": "MacOS", "disk": "10G" ]  
[ "memory": "2G", "os": "Windows",  
  "disk": "10G" ]  
[ "memory": "256M", "os": "DC.DC",  
  "disk": "10G" ]  
[ "memory": "512M", "os": "Linux",  
  "disk": "10G" ]  
[ "memory": "2G", "os": "Linux",  
  "disk": "DC.DC" ]  
[ "memory": "512M", "os": "Windows",  
  "disk": "DC.DC" ]  
[ "memory": "256M", "os": "MacOS",  
  "disk": "DC.DC" ]  
[ "memory": "256M", "os": "Linux",  
  "disk": "DC.DC" ]  
[ "memory": "2G", "os": "MacOS",  
  "disk": "DC.DC" ]  
[ "memory": "1G", "os": "Windows",  
  "disk": "DC.DC" ]
```

Note: the value "DC.DC" indicates a don't care value and can be replaced with any value for that field.

### Effectiveness of testing combinations with all pairs

Rather than testing every possible combination of things, all pairs simplifies the exercise to testing every pair of things which reduces the complexity significantly, for example instead of 700,000 possible combinations, all pairs

would be about 500 combinations.

While this reduces the number of tests, does it help find bugs? All pairs works because when things break, they have a tendency to break because of the faulty interaction of two things rather than 3 or more. A long term study of medical device failures found a strong correlation for this. For all the failures reported, a quarter of them would have been found with all pairs testing. The true result is probably much better than this, because a lot of the failure reports did not have enough detail to allow proper analysis. Of the detailed reports, 98% of the failures would have been found with all pairs testing! The paper, "Failure Modes in Medical Devices", is at [csrc.ncsi.nist.gov/staff/kuhn/final-rqse.pdf](http://csrc.ncsi.nist.gov/staff/kuhn/final-rqse.pdf)

If you do all pairs testing, you could still use minimal pairs to get better test effectiveness. By starting your tests with all the minimal pairs first, this would give a good broad coverage of combinations. The remaining all pairs combinations could then be used to finish the exercise.

## Test Coverage

Code coverage is a useful measure of the effectiveness of unit tests and can be derived for Groovy tests.

Consider the following Groovy code:

```
class BiggestPairCalc
{
    int sumBiggestPair(int a, int b, int c)
    {
        def op1 = a
        def op2 = b
        if (c > a) {
            op1 = c
        } else if (c > b) {
            op2 = c
        }
        return op1 + op2
    }
}
```

And the following test:

```

class BiggestPairCalcTest extends
GroovyTestCase
{
    void testSumBiggestPair() {
        def calc = new BiggestPairCalc()
        assertEquals(9,
calc.sumBiggestPair(5, 4, 1))
    }
}

```

If you use [Cobertura](#) to perform your coverage, the resulting report might look like:

#### Coverage Report - BiggestPairCalc

Classes in this File	Line Coverage	Branch Coverage	Complexity
<a href="#">BiggestPairCalc</a>	71% <span style="background-color: #90EE90;">5/7</span>	100% <span style="background-color: #90EE90;">2/2</span>	0

1	class BiggestPairCalc
2	{
3	int sumBiggestPair(int a, int b, int c) {
4	1 def op1 = a
5	1 def op2 = b
6	1 if (c > a) {
7	0 op1 = c
8	1 } else if (c > b) {
9	0 op2 = c
10	}
11	1 return op1 + op2
12	}
13	}

Your [Ant](#) build file to make all this happen might look like:

```

<?xml version="1.0"?>
<project name="sample"
default="coverage-report" basedir=".">
  <!-- set up properties, paths, taskdefs,
prepare targets -->
  [details deleted]

  <!-- compile java (if you have any) and

```

```

groovy source -->
  <target name="compile" depends="prepare">
    <javac srcdir="${dir.src}"
destdir="${dir.build}" debug="true">
      <classpath refid="project.classpath"/>
    </javac>
    <groovyc srcdir="${dir.src}"
destdir="${dir.build}" stacktrace="true">
      <classpath refid="project.classpath"/>
    </groovyc>
  </target>

  <!-- instrument already compiled class
files -->
  <target name="instrument"
depends="compile" >
    <cobertura-instrument
todir="target/instrumented-classes">
      <fileset dir="${dir.build}">
        <include name="**/*.class"/>
      </fileset>
    </cobertura-instrument>
  </target>

  <!-- run all junit tests using the
instrumented classes -->
  <target name="cover-test"
depends="instrument">
    <mkdir dir="${dir.report}/cobertura" />
    <junit printsummary="yes"
haltonerror="no" haltonfailure="no"

```

```
fork="yes">
  <formatter type="plain"
usefile="false"/>
  <batchtest>
    <fileset
dir="target/instrumented-classes"
includes="**/*Test.class" />
  </batchtest>
  <classpath
refid="cover-test.classpath"/>
  </junit>
</target>

<!-- create the html reports -->
<target name="coverage-report"
depends="cover-test">
  <cobertura-report srcdir="${dir.src}"
```

```
    destDir="${dir.report}/cobertura" />
  </target>
</project>
```

For more details, see [GINA](#) or the [Cobertura](#) web site or [Code Coverage with Cobertura](#).

### Code Coverage with Cobertura

Starting with v1.8, Cobertura will create detailed reports for Groovy source. It can be download from <http://cobertura.sourceforge.net/download.html>

Below is an example build file for using Cobertura in your project.

Notice that you must compile your groovy source into .class files so that Cobertura can add instrumentation to those .class files.

---

build.xml

---

```
<?xml version="1.0"?>
<project name="sample"
default="coverage-report" basedir=".">
  <!-- properties for project directory
structure -->
  <property name="dir.src" value="." />
  <property name="dir.build" value="build" />
  <property name="dir.dist" value="dist" />
  <property name="dir.lib" value="lib" />
  <property name="dir.report"
value="report" />

  <!-- setup a project classpath that
includes the external libs -->
  <path id="project.classpath">
    <!-- include the classes in this project
-->
    <pathelement location="${dir.build}" />
```

```
    <!-- include external libraries -->
    <fileset dir="{dir.lib}"
includes="**/*.jar"/>
  </path>

  <!-- add external tasks -->
  <taskdef name="groovyc"
classpathref="project.classpath"
classname="org.codehaus.groovy.ant.Groovyc" /
>
  <taskdef name="groovy"
classpathref="project.classpath"
classname="org.codehaus.groovy.ant.Groovy" />

  <!-- create output directories -->
  <target name="prepare" depends="clean">
    <mkdir dir="{dir.build}"/>
  </target>

  <!-- clean -->
  <target name="clean" description="Remove
all generated files.">
    <delete dir="{dir.build}"/>
    <delete dir="{dir.report}"/>
    <delete
dir="target/instrumented-classes"/>
    <delete file="cobertura.ser"/>
  </target>

  <!-- compile java (if you have any) and
groovy source -->
```

```
<target name="runGroovyC"
depends="prepare">
    <groovyc srcdir="${dir.src}"
destdir="${dir.build}" stacktrace="true">
        <classpath refid="project.classpath"/>
    <javac debug="on" />
    </groovyc>

</target>
```

```
<!--=====
=====-->
```

```
<!-- Cobertura Test Coverage Tool -->
```

```
<!--=====
=====-->
```

```
<path id="cobertura.classpath">
    <fileset dir="${dir.lib}/cobertura"
includes="**/*.jar"/>
    <pathelement
location="target/instrumented-classes"/>
    <pathelement location="${dir.src}"/>
</path>
```

```
<taskdef
classpath="${dir.lib}/cobertura/cobertura.jar"
resource="tasks.properties"
classpathref="cobertura.classpath"/>
```

```
<!-- adds the logging code to the already
```

```
compiled class files -->
  <target name="instrument"
depends="runGroovyC">
  <delete quiet="false"
failonerror="false">
    <fileset
dir="target/instrumented-classes"
includes="**/*.class"/>
  </delete>
  <cobertura-instrument
todir="target/instrumented-classes">
    <fileset dir="${dir.build}">
      <include name="**/*.class"/>
      <exclude name="**/*Test.class"/>
    </fileset>
  </cobertura-instrument>
</target>

<!-- setup class path to include
instrumented classes before non-instrumented
ones -->
  <path id="cover-test.classpath">
    <fileset dir="${dir.lib}"
includes="**/*.jar"/>
    <pathelement
location="target/instrumented-classes"/>
    <pathelement location="${dir.build}"/>
  </path>

  <!-- run all my junit tests using the
instrumented classes -->
```

```
<target name="cover-test"
depends="instrument">
    <mkdir dir="${dir.report}/cobertura" />
    <junit printsummary="yes"
haltonerror="no" haltonfailure="no"
fork="yes">
        <formatter type="plain"
usefile="true"/>
        <batchtest>
            <fileset dir="${dir.build}"
includes="**/*Test.class" />
        </batchtest>
        <classpath
refid="cover-test.classpath"/>
        </junit>
    </target>

<!-- create the html reports -->
<target name="coverage-report"
depends="cover-test">
    <cobertura-report srcdir="${dir.src}"
```

```
    destdir="${dir.report}" />
    </target>
</project>
```

See also: [Test Coverage](#)

## Testing Web Applications

This page discusses how to use NekoHTML, HtmlUnit, Watij and WebTest to test web applications. There are many ways to test Web Applications with Groovy:

- use Groovy (potentially in conjunction with a specialist HTML parser) to parse HTML pages as if they were XML
- use Groovy to simplify the code required to drive a Java API browser simulator, e.g. HtmlUnit or HttpUnit
- use Groovy to simplify the code required to drive a Java API for manually driving a real browser, e.g. IE or Firefox
- use Groovy to interact with a higher-level testing library which uses one of the above two approaches, e.g. Watij (for the equivalent of Watir in the Ruby world) or WebTest (to open up the possibility of testing more than just web applications)

We examine a few approaches below.

### Groovy with CyberNeko HTML Parser

[NekoHTML](#) is a library which allows you to parse HTML documents (which may not be well-formed) and treat them as XML documents (i.e. XHTML). NekoHTML automatically inserts missing closing tags and does various other things to clean up the HTML if required - just as browsers do - and then makes the result available for use by normal XML parsing techniques.

Here is an example of using NekoHTML with XmlParser to find '.html' hyperlinks on the groovy homepage:

```
def parser = new
org.cyberneko.html.parsers.SAXParser()
parser.setFeature('http://xml.org/sax/features/namespaces', false)
def page = new
XmlParser(parser).parse('http://groovy.codehaus.org/')
def data = page.depthFirst().A.'@href'.grep{
it != null && it.endsWith('.html') }
data.each { println it }
```

We turned off namespace processing which lets us select nodes using 'A' with no namespace.

Here is one way to do the same example with XmlSlurper:

```
def page = new XmlSlurper(new
org.cyberneko.html.parsers.SAXParser()).parse('http://groovy.codehaus.org/')
def data = page.depthFirst().grep{ it.name()
== 'A' &&
it.@href.toString().endsWith('.html')
}.'@href'
data.each { println it }
```

We didn't turn off namespace processing but do the selection using just the local name, i.e. 'name()'.

Here is the output in both cases:

```
http://groovy.codehaus.org/apidocs/index.html
1
/faq.html
/groovy-jdk.html
http://groovy.codehaus.org/team-list.html
http://groovy.codehaus.org/xref/index.html
http://www.javamagazin.de/itr/ausgaben/psecocom,id,317,nodeid,20.html
http://www.weiqigao.com/blog/2006/09/14/gruby_on_grails_tonight_at_630.html
http://www.oreillynet.com/onjava/blog/2006/09/charles_nutter_responds_our_fu.html
```

Now that we have the links we could do various kinds of assertions, e.g. check the number of links, check that a particular link was always on the page, or check that there are no broken links.

Groovy and HtmlUnit

The following example tests the Google search engine using [HtmlUnit](#):

```
import
com.gargoylesoftware.htmlunit.WebClient

def webClient = new WebClient()
def page =
webClient.getPage('http://www.google.com')
// check page title
assert 'Google' == page.titleText
// fill in form and submit it
def form = page.getFormByName('f')
def field = form.getInputByName('q')
field.setValueAttribute('Groovy')
def button = form.getInputByName('btnG')
def result = button.click()
// check groovy home page appears in list
(assumes it's on page 1)
assert result.anchors.any{ a ->
a.hrefAttribute ==
'http://groovy.codehaus.org/' }
```

Note that to use HtmlUnit with Groovy you should not include the xml-apis-\*.jar included with HtmlUnit in your CLASSPATH.

### Groovy and Watij

The following example tests the Google search engine using [Watij](#):

```
import watij.runtime.ie.IE
import watij.finders.SymbolFactory

def ie = new IE()
ie.start('http://www.google.com')
// check page title
assert ie.title() == 'Google'
// fill in query form and submit it
ie.textField(SymbolFactory.@name,
'q').set('Groovy')
ie.button(SymbolFactory.@name,
'btnG').click()
// check groovy home page appears in list by
trying to flash() it
ie.link(SymbolFactory.url,
'http://groovy.codehaus.org/').flash()
ie.close()
```

You can use Watij from within groovysh or groovyconsole if you want to have an interactive (irb-like in ruby terms) experience.

### Groovy and WebTest

The following example tests the Google search engine using [WebTest](#):

```
<webtest name="google test">
  <steps>
    <invoke url="http://google.com"/>
    <verifyTitle text="Google"/>
    <setInputField name="q"
value="Groovy"/>
    <clickButton name="btnG"/>
    <verifyXPath
xpath="//a[@href='http://groovy.codehaus.org
/']" />
  </steps>
</webtest>
```

The above fragment can be inserted into any [Ant](#) build script where we have defined the WebTest tasks.

The above example didn't use any Groovy but we could have just as easily used some Groovy for the last line if we didn't like the XPath expression, for example:

```

<webtest name="google test">
  <steps>
    <invoke url="http://google.com"/>
    <verifyTitle text="Google"/>
    <setInputField name="q"
value="Groovy"/>
    <clickButton name="btnG"/>
    <groovy>
      assert
      step.context.currentResponse.anchors.any{ a
-> a.hrefAttribute ==
      'http://groovy.codehaus.org/' }
    </groovy>
  </steps>
</webtest>

```

Depending on your setup, you could produce the following report for this test:

Result	#	%	Graph	Secs	#	%	Histogram
✓	5	100		0 - 1	1	50	
✗	0	0		1 - 5	1	50	
○	0	0		5 - 10	0	0	
Sum	5	100		10 - 30	0	0	
				> 30	0	0	
				Sum	2	100	
				Avg		2	

Test Scenario Overview (4844)

No	Result	Name	# Steps	Duration	%	Timing profile	Graph	Failing step
1	✓	google test	5 / 5	4844	100			

✓ google test

Test started at Wed Aug 02 15:53:28 EST 2006, lasting 4844 ms.  
Source: C:\Downloads\webtest.canoo.com\play\googlemacrodef.xml:49:  
Base URL (used by invoke steps with a relative URL): http://null/

No	Result	Name	Parameter	Duration
1	✓	invoke	method GET url http://google.com	2594
2	✓	verifyTitle	text Google	0
3	✓	setInputField	name q value Groovy	16
4	✓	clickButton	name btnG	656
5	✓	groovy		1578

Alternatively, we could have written the whole test in Groovy using AntBuilder as follows:

```
def webtest_home =
System.properties.'webtest.home'
def ant = new AntBuilder()

ant.taskdef(resource: 'webtest.taskdef') {
  classpath() {

pathelement(location: "$webtest_home/lib")
          fileset(dir: "$webtest_home/lib",
includes: "**/*.jar")
  }
}

ant.webtest(name: 'Test Google with Groovy,
AntBuilder and WebTest') {
  steps() {
    invoke(url: 'http://www.google.com')
    verifyTitle(text: 'Google')
    setTextField(name: 'q',
value: 'Groovy')
    clickButton(name: 'btnG')

verifyXPath(xpath: "//a[@href='http://groovy.
codehaus.org/']")
  }
}
```

Grails can automatically create this style of test for your generated CRUD applications, see [Grails Functional Testing](#) for more details.

## Testing Web Services

This page discusses how to test Web Services using Groovy directly and in conjunction with WebTest and SoapUI.

Testing Web Services can be done in several ways. Here are three:

- act like a normal web services client and perform asserts on the returned result
- use [WebTest](#) (with either the XML or Groovy syntax)
- use [SoapUI](#) (for functional and load testing)

We are going to use the Web Service example at:

<http://groovy.codehaus.org/Groovy+SOAP>

### Being a client

You can be a normal client web service client and perform asserts on the returned results:

```
import groovy.net.soap.SoapClient  
  
def proxy = new  
SoapClient("http://localhost:6980/MathServiceInterface?wsdl")  
  
def result = proxy.add(1.0, 2.0)  
assert (result == 3.0)  
  
result = proxy.square(3.0)  
assert (result == 9.0)
```

### Using WebTest

Using the WebTest variations makes sense if you are combining your tests into an acceptance test suite.

Here is how you would test it using traditional WebTest:

```
<steps>
  <invoke method="POST"
contentFile="addreq.xml" soapAction=""

url="http://localhost:6980/MathServiceInterf
ace"/>
  <verifyXPath xpath="//addResponse/out[
text()='3.0' ]"/>
  <invoke method="POST"
contentFile="suarereq.xml" soapAction=""

url="http://localhost:6980/MathServiceInterf
ace"/>
  <verifyXPath
xpath="//squareResponse/out[ text()='9.0'
]"/>
</steps>
```

Where addreq.xml would look something like:

```
<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope

xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <soap:Body>
    <add
xmlns="http://DefaultNamespace">
      <in0
xmlns="http://DefaultNamespace">1.0</in0>
      <in1>2.0</in1>
    </add>
  </soap:Body>
</soap:Envelope>
```

and squarereq.xml would look something like:

```
<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope

xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <soap:Body>
    <square
xmlns="http://DefaultNamespace">
      <in0
xmlns="http://DefaultNamespace">3.0</in0>
    </square>
  </soap:Body>
</soap:Envelope>
```

Alternatively, testing using groovy within WebTest would look like:

```
<steps>
  <groovy>
    import groovy.net.soap.SoapClient

    def proxy = new
SoapClient("http://localhost:6980/MathServiceInterface?wsdl")

    def result = proxy.add(1.0, 2.0)
    assert (result == 3.0)

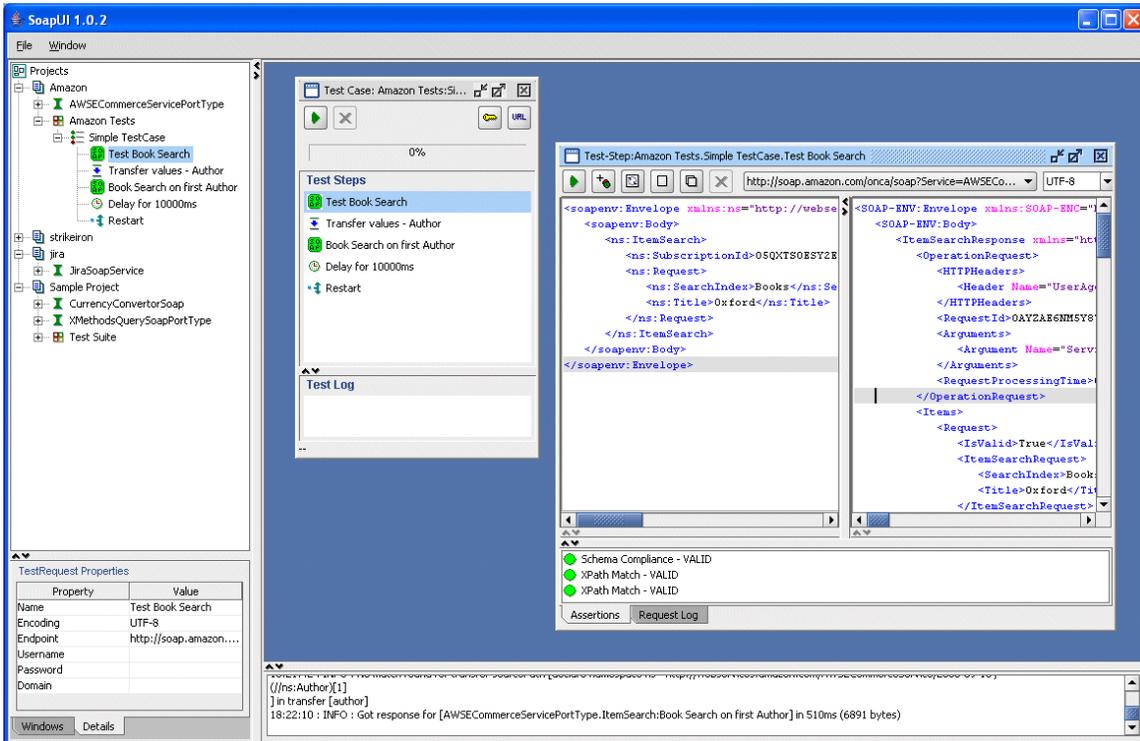
    result = proxy.square(3.0)
    assert (result == 9.0)
  </groovy>
</steps>
```

Note: you will need to place the jars mentioned on that page in your webtest lib directory (i.e. groovysoap, stax, jaf and mail jars) when using this variation.

The first approach (traditional webtest) produces more information in the test summary reporting but requires you to do more work (i.e. keep the requests around as XML). It depends if you already have those XML files around for other purposes, e.g. manual testing.

### Using SOAPUI

[soapui](#) is a SOAP functional and load testing tool. It can use Groovy steps within its testcases. For further details, see the soapui documentation for the [Groovy Step](#). This step supports data-driven tests, allows control of test execution and allows customised reporting.



## Using Testing Frameworks with Groovy

Groovy is great for Agile development in general and testing in particular because:

- it has built-in support for the JUnit testing framework
- it has built-in mocking capabilities
- it provides a very expressive language in which to write tests which can utilise Closure syntax and Groovy's other features which support the creation of testing domain specific languages (DSLs)
- it can easily leverage Java testing frameworks and IDE support
- it's built-in AntBuilder support makes it easy to set up integration tests

This page explores testing features of Groovy and some other testing frameworks that you may sometimes wish to use to complement Groovy's built-in capabilities.

Two main examples are used:

- A Stack Example
- An Item Storer Example

⚠ Some of the examples on this page rely on annotations: a feature in Groovy 1.5+ when used with Java 5 and above.

### A Stack Example

Inspired by the [RSpec](#) stack example, suppose we want to test the following Groovy classes:

```
class StackOverflowException extends  
RuntimeException {}
```

```
class StackUnderflowException extends  
RuntimeException {}
```

```
class FixedStack {  
    public static MAXSIZE = 10  
    private items = []  
  
    def push(object) {  
        if (items.size() == MAXSIZE) throw  
new StackOverflowException()  
        items << object  
    }  
  
    def pop() {  
        if (!items) throw new  
StackUnderflowException()  
        items.pop()  
    }  
  
    def peek() {  
        if (!items) throw new  
StackUnderflowException()  
        items[-1]  
    }  
  
    boolean isEmpty() {  
        items.isEmpty()  
    }  
  
    boolean isFull() {
```

```
items.size() == MAXSIZE
```

```
    }  
}
```

We can test this with vanilla Groovy using the following tests:

```
class NonEmptyFixedStackTest extends  
GroovyTestCase {  
    private stack  
  
    void setUp() {  
        stack = new FixedStack()  
        [ "a", "b", "c" ].each{ x ->  
stack.push x }  
    }  
  
    void testPreConditions() {  
        assert !stack.isEmpty()  
    }  
  
    void testShouldAddToTheTopWhenSentPush()  
{  
        stack.push "d"  
        assert stack.peek() == "d"  
    }  
  
    void  
testShouldBeUnchangedWhenSentPushThenPop() {  
        stack.push "anything"  
        stack.pop()  
        assert stack.peek() == "c"  
    }  
}
```

```

    }

    void
testShouldReturnTheTopItemWhenSentPeek() {
    assert stack.peek() == "c"
}

    void
testShouldNotRemoveTheTopItemWhenSentPeek()
{
    assert stack.peek() == "c"
    assert stack.peek() == "c"
}

    void
testShouldReturnTheTopItemWhenSentPop() {
    assert stack.pop() == "c"
}

    void
testShouldRemoveTheTopItemWhenSentPop() {
    assert stack.pop() == "c"
    assert stack.pop() == "b"
}
}

class EmptyFixedStackTest extends
GroovyTestCase {
    private stack = new FixedStack()

```

```
void testPreConditions() {
    assert stack.isEmpty()
}

void
testShouldNoLongerBeEmptyAfterPush() {
    stack.push "anything"
    assert !stack.isEmpty()
}

void testShouldComplainWhenSentPeek() {
    shouldFail(StackUnderflowException)
{
    stack.peek()
}
}

void testShouldComplainWhenSentPop() {
    shouldFail(StackUnderflowException)
{
    stack.pop()
}
}

}

class FullFixedStackTest extends
GroovyTestCase {
    private stack

    void setUp() {
        stack = new FixedStack()
```

```

        (1..FixedStack.MAXSIZE).each{ x ->
stack.push x }
    }

    void testPreConditions() {
        assert stack.isFull()
    }

    void testShouldRemainFullAfterPeek() {
        stack.peek()
        assert stack.isFull()
    }

    void testShouldNoLongerBeFullAfterPop()
{
        stack.pop()
        assert !stack.isFull()
    }

    void testShouldComplainOnPush() {
        shouldFail(StackOverflowException) {
            stack.push "anything"
        }
    }
}

class AlmostFullFixedStackTest extends
GroovyTestCase {
    private stack

    void setUp() {

```

```
        stack = new FixedStack()
        (1..<FixedStack.MAXSIZE).each{ x ->
stack.push x }
    }
```

```
void testPreConditions() {
    assert !stack.isFull()
}
```

```
void testShouldBecomeFullAfterPush() {
    stack.push "anything"
    assert stack.isFull()
}
```

```
}
```

```
class AlmostEmptyFixedStackTest extends
GroovyTestCase {
    private stack
```

```
void setUp() {
    stack = new FixedStack()
    stack.push "anything"
}
```

```
void testPreConditions() {
    assert !stack.isEmpty()
}
```

```
void testShouldRemainNotEmptyAfterPeek()
{
    stack.peek()
```

```
    testPreConditions()  
}  
  
void testShouldBecomeEmptyAfterPop() {  
    stack.pop()  
}
```

```
        assert stack.isEmpty()  
    }  
}
```

Of course, even within vanilla Groovy we have a few options. E.g., we could use no test class, just a script, in which case, the naming conventions required by JUnit 3.x (used above) wouldn't apply. Or, we could have used JUnit 4 and annotations to allow us to use alternate naming conventions for the methods. Alternatively, we could use a single test class with different test fixtures, e.g. `fullStack` and `emptyStack`. The other frameworks [mentioned below](#) allow additional possibilities for organising and naming our tests.

### An Item Storer Example

Suppose we have the following Java interface (Groovy supports interface-oriented programming as well as dynamic programming using duck-typing and here we want to illustrate Java test/mock libraries later):

```
// Java  
public interface Reverser {  
    Object reverse(Object item);  
}
```

Now suppose we have an implementation method as follows:

```
class GroovyReverser implements Reverser {  
    def reverse(item) {  
        if (item instanceof Number) return  
-item  
        return item.reverse()  
    }  
}
```

For numbers, the `reverse()` method will negate them. Thanks to duck-typing, other objects that we try to reverse will just call their respective types `reverse()` method if it exists, e.g. so it will work for `String` and `List` objects.

Now suppose we make use of a reverser in some code we are trying to test.

```
class Storer {
    def stored
    Reverser reverser = new GroovyReverser()
    def put(item) {
        stored = item
    }
    def get() {
        return stored
    }
    def getReverse() {
        return reverser.reverse(stored)
    }
}
```

### *Integration Testing*

We can integration test this class as follows:

```
def checkPersistAndReverse(cut, value,
reverseValue) {
    cut.put(value)
    assert value == cut.get()
    assert reverseValue == cut.getReverse()
}

def testBehavior(cut) {
    checkPersistAndReverse cut, 123.456,
-123.456
    checkPersistAndReverse cut, 'hello',
'olleh'
    checkPersistAndReverse cut, [1, 3, 5],
[5, 3, 1]
}

testBehavior(new Storer())
```

### *Mocking Groovy Classes*

The above integration tests exercise our class under test with the production `reverser` in place. For this particular example, we might argue that such a test is appropriate and sufficient. However, in more complicated scenarios, the dependent collaboration class (`GroovyReverser` in this case) might be difficult or expensive to construct. In those cases, we would want to replace it with a mock or stub. See [Groovy Mocks](#) for a definition of terms.

Here is how we would use Groovy's built-in mock support to test our class under test in isolation:

```

import groovy.mock.interceptor.MockFor

def mockReverser = new
MockFor(GroovyReverser.class)
mockReverser.demand.reverse{ arg -> -arg }
mockReverser.demand.reverse{ arg ->
arg.reverse() }
mockReverser.use {
    testBehavior(new Storer())
}

```

Note that we didn't need to do anything to inject our mocks into the class under test. Inside the use method, all attempts to create a `GroovyReverser` object will be replaced by a mock object. This also works for Java objects created by Groovy objects, as the following shows:

```

mockReverser = new
MockFor(JavaReverser.class)
mockReverser.demand.reverse{ arg -> -arg }
mockReverser.demand.reverse{ arg ->
arg.reverse() }
mockReverser.use {
    testBehavior(new Storer(reverser:new
JavaReverser()))
}

```

Here `JavaReverser` is a class that we have defined as follows:

```

import java.util.List;
import java.util.ArrayList;
import java.util.ListIterator;

```

```

public class JavaReverser implements
Reverser {
    public Object reverse(Object item) {
        if (item instanceof Number) {
            Number n = (Number) item;
            return new
Double(-n.doubleValue());
        }
        if (item instanceof String) {
            String s = (String) item;
            int size = s.length();
            StringBuffer sb = new
StringBuffer(size);
            for (int i = size - 1; i >= 0;
i--) {
                sb.append(s.charAt(i));
            }
            return sb.toString();
        }
        if (item instanceof List) {
            List l = (List) item;
            int size = l.size();
            List result = new
ArrayList(size);
            ListIterator iter =
l.listIterator(size);
            while (iter.hasPrevious()) {
                result.add(iter.previous());
            }
            return result;
        }
    }
}

```

```
}  
throw new
```

```
UnsupportedOperationException();  
    }  
}
```

Hmmm. Quite a bit longer than the Groovy version. But that's another story.

#### *Testing Java classes which use other Java classes*

Now, consider now the following Java class:

```
public class JavaStorer {  
    Object stored;  
    Reverser reverser;  
  
    public JavaStorer(Reverser reverser) {  
        this.reverser = reverser;  
    }  
  
    void put(Object item) {  
        stored = item;  
    }  
    Object get() {  
        return stored;  
    }  
    Object getReverse() {  
        return reverser.reverse(stored);  
    }  
}
```

We now want to test this too and we want to use Groovy to write the tests. Unfortunately, Groovy's built-in mock support won't help us here. It doesn't allow us to test Java classes like this one as it relies on hooking into Groovy's object lifecycle mechanisms. Instead, we can use any of the available Java mocking packages such as the ones [mentioned here](#).

GSpec is a testing framework written in Groovy, which this example is based on. See [this page](#) for a full explanation of using GSpec for BDD style testing.

## Using GSpec with Groovy

[GSpec](#) is an evolving framework to allow you to apply a [BDD](#) style of programming when using Groovy. The sections below illustrate using GSpec for the examples from [Using Testing Frameworks with Groovy](#).

### *The Stack Example*

Here is how you might test the [Stack Example](#) with GSpec:

```
import com.craig.gspec.GSpecBuilderRunner

def the = new GSpecBuilderRunner()

the.context('A non-empty stack') {
  initially {
    the.stack = new FixedStack()
    ('a'..'c').each { x ->
the.stack.push x }
    the.stack.should_not_be_empty
  }

  specify('should return the top item when
sent #peek') {
    the.stack.peek().should_equal 'c'
  }

  specify('should NOT remove the top item
when sent #peek') {
    the.stack.peek().should_equal 'c'
    the.stack.peek().should_equal 'c'
  }
}
```

```
    specify('should be unchanged when sent
#push then #pop') {
        the.stack.push 'Anything'
        the.stack.pop()
        the.stack.peek().should_equal 'c'
    }
}
```

```
    specify('should return the top item when
sent #pop') {
        the.stack.pop().should_equal 'c'
        the.stack.push 'c' // put it back
the way it was
    }
}
```

```
    specify('should remove the top item when
sent #pop') {
        the.stack.pop().should_equal 'c'
        the.stack.pop().should_equal 'b'
    }
}
```

```
    specify('should add to the top when sent
#push') {
        the.stack.push 'd'
        the.stack.peek().should_equal 'd'
    }
}
```

```
the.context('An empty stack') {
    initially {
        the.stack = new FixedStack()
        the.stack.should_be_empty
    }
}
```

```
    }

    specify('should no longer be empty after
#push') {
        the.stack.push 'anything'
        the.stack.should_not_be_empty
    }

    specify('should complain when sent
#peek') {
//          the.stack.peek().should_fail_with
StackUnderflowException
    }

    specify('should complain when sent
#pop') {
//          the.stack.pop().should_fail_with
StackUnderflowException
    }
}

the.context('A stack with one item') {
    initially {
        the.stack = new FixedStack()
        the.stack.push 'anything'
        the.stack.should_not_be_empty
    }

    specify('should remain not empty after
#peek') {
        the.stack.peek()
```

```

        the.stack.should_not_be_empty
    }

    specify('should become empty after
#pop') {
        the.stack.pop()
        the.stack.should_be_empty
    }
}

the.context('A stack with one item less than
capacity') {
    initially {
        the.stack = new FixedStack()
        (1..<FixedStack.MAXSIZE).each { x ->
the.stack.push x }
        the.stack.should_not_be_full
    }

    specify('should become full after
#push') {
        the.stack.push 'Anything'
        the.stack.should_be_full
    }
}

the.context('A full stack') {
    initially {
        the.stack = new FixedStack()
        (1..FixedStack.MAXSIZE).each { x ->
the.stack.push x }

```

```
        the.stack.should_be_full
    }

    specify('should remain full after
#peek') {
        the.stack.peek()
        the.stack.should_be_full
    }

    specify('should no longer be full after
#pop') {
        the.stack.pop()
        the.stack.should_not_be_full
    }

    specify('should complain on #push') {
//
the.stack.push('Anything').should_fail_with
```

## StackOverflowException

```
    }  
}
```

### *The Item Storer Example*

Here is how you might integration test the [Item Storer Example](#) with GSpec:

```
import com.craig.gspec.GSpecBuilderRunner  
  
def checkPersistAndReverse(storer, orig,  
reversed){  
    storer.put orig  
    storer.get().should_equal orig  
    storer.getReverse().should_equal  
reversed  
}  
  
def the = new GSpecBuilderRunner()  
  
the.context('A new storer') {  
    initially() {  
        the.storer = new Storer()  
    }  
    specify('Should persist and reverse  
strings') {  
        checkPersistAndReverse the.storer,  
'hello', 'olleh'  
    }  
    specify('Should persist and reverse  
numbers') {  
        checkPersistAndReverse the.storer,
```

```
123.456, -123.456
    }
    specify('Should persist and reverse
lists') {
        checkPersistAndReverse the.storer,
```

```
[ 1, 3, 5 ], [ 5, 3, 1 ]
}
}
```

## Developer Guide

The developer guide contains information mainly of interest to the developers involved in creating Groovy and its supporting modules and tools.

Useful starting points:

- [JavaDoc](#) for Groovy internal classes written in Java
- [GroovyDoc](#) for Groovy internal classes written in Groovy
- Get started with the sources with [Git](#)
  - details about the Git [source code repository](#)
- binary distribution repositories: [Main](#), Snapshots ([Canoo](#), [Codehaus Bamboo](#))
- local Maven 2 repositories: [Main](#) (may be empty once synced with global repositories), [Snapshot](#)
- details about the [Continuous Integration](#) servers which build Groovy around the clock
- the root [Wiki](#) page

Other topics:

- [Building Groovy from Source](#)
  - [Setting up IntelliJ IDEA to build Groovy](#)
- [Building Groovy from Source using Ant](#)
- [Continuous Integration](#)
- [From source code to bytecode](#)
- [Git](#)
- [Groovy Backstage](#)
  - [Groovy Method Invokation](#)
- [Groovy Internals](#)
- [Release Process](#)
- [Setup Groovy Development Environment](#)

## Building Groovy from Source

Please follow the instructions on the groovy github page: <https://github.com/groovy/groovy-core#building>

### Setting up IntelliJ IDEA to build Groovy

Please follow the instructions on the groovy github page: <https://github.com/groovy/groovy-core#building>

## Building Groovy from Source using Ant

### Warning

**Starting from Groovy 2.0-0-rc-1, the Ant build is no longer supported. [Gradle](#) is used instead.**

This is a legacy page that describes how Groovy was built from sources using ant prior to the adoption of Gradle.

## Prerequisites

### *Java Developer Kit (JDK)*

You will need a [JDK 1.5+ \(J2SE 1.5+\)](#) or compatible JDK to build Groovy.

### *Apache Ant*

To execute the build process you need to have [Apache Ant](#) version 1.7.0 (or newer) installed.

 The optional TraX taskdefs are required, and some linux distributions, such as openSUSE, may require you to install them separately from the ant packages.

### *Git Client*

To fetch the source code for the server, you will need to have a [Git](#) client version 1.2 (or newer, 1.4 is recommended) installed.

 If you are using a source archive from the [distribution site](#) site, then you do not need to have a Git client installed.

## Fetch the Groovy Source Code

### *From Git*

Have a look at our [Git page](#) for further details on how to get the sources, either from the Codehaus Git repository, or the GitHub mirror.

### *From a Source Archive*

Select one of the \*-src.\* archives from the [distribution site](#) site and unzip it.

For example, to use the source from the Groovy 2.0.0-beta-2 release:

```
wget
http://dist.groovy.codehaus.org/distributions/groovy-src-2.0.0-beta-2.zip
unzip groovy-src-2.0.0-beta-2.zip
```

## Preparing to Build for the First Time

If you are behind a proxy firewall, you may need to set up proxy information in a `~/.ant/settings.xml` or `~/m2/settings.xml` file:

```
<?xml version="1.0"?>
<settings>
  <proxies>
    <proxy>
      <protocol>http</protocol>
      <host>proxy.host.net</host>
      <port>8080</port>
    </proxy>
  </proxies>
  <nonProxyHosts>localhost</nonProxyHosts>
</settings>
```

If you will need to publish artifacts then you will need to have appropriate permissions with codehaus and ensure that you have stored your credentials in a `~/.ant/settings.xml` or `~/.m2/settings.xml` file of the form:

```
<?xml version="1.0"?>
<settings>
  <servers>
    <server>
      <id>codehaus.org</id>
      <username>YOUR_USER_NAME</username>
      <password>YOUR_PASSWORD</password>
    </server>
  </servers>
</settings>
```

## Building

To build everything, run tests and create a complete installation in `target/install`, run:

```
ant install
```

**i Useful Information**

In case of `java.lang.OutOfMemoryError` try increasing the size of the heap space by setting the `ANT_OPTS` variable as follows:

```
ANT_OPTS=-Xmx512M
```

Increase the size of the heap size if required.

**i** If the unit tests hang indefinitely while building, try adding the following to your ant command:

```
-Djava.awt.headless=true
```

After this finished successfully, then you can try out the distribution that was just built. For example to play with `groovysh` you can run:

```
./target/install/bin/groovysh
```

Sometimes its desirable to skip the test execution for faster turn around. To turn off tests set the `skipTests` property, as in:

```
ant install -DskipTests=true
```

### Distribution Archives

To build a Groovy distribution archive:

```
ant dist
```

This will build everything, generate documentation and create distribution archives under `target/dist`:

- `groovy-binary-VERSION.zip`
- `groovy-docs-VERSION.zip`
- `groovy-src-VERSION.zip`

**⚠** As you might have guessed *VERSION* will be the version of the project you are building, like 2.0.0-beta-2-SNAPSHOT for pre-releases or 2.0.0-beta-2 for a released version.

## Publishing Artifacts

To publish artifacts to the Maven 2 repository, run:

```
ant deploy
```

This will either publish to the [release repository](#) or the [snapshot repository](#) based on whether the POM version contains SNAPSHOT or not.

If deploy is failing, then check [Installing the UserTrust CA into the Java JDK](#).

## Building from IntelliJ IDEA

Please follow the instructions [found here](#).

## Continuous Integration

The Groovy project is extremely lucky to have several [continuous integration](#) environments available to us to give us immediate feedback whenever changes are made to Groovy. As Groovy evolves, this helps us ensure that it remains working on various platforms (including various operating systems and Java versions running on [Bamboo](#) and [TeamCity](#)).

At the moment the main CI server is hosted by [The Codehaus](#) but another version is hosted by [JetBrains](#). Details are covered below.

## Server Information

URL	<a href="http://bamboo.ci.codehaus.org/browse/GROOVY">http://bamboo.ci.codehaus.org/browse/GROOVY</a>
Build Server	Bamboo
Operating System	Fedora
Java Version	1.5, 1.6, 1.7
Mail Data Feed	subscribe to SCM <a href="#">Mailing Lists</a>



URL	<a href="http://teamcity.jetbrains.com/project.html?projectId=project12">http://teamcity.jetbrains.com/project.html?projectId=project12</a> (login as guest when prompted)
Build Server	TeamCity
Operating System	Linux, Windows, MacOS available
Java Version	1.4, 1.5, 1.6 available



## From source code to bytecode

Here are some tips on how to debug what is happening on the transition from groovy source code into the generated bytecode.

artifact	transformation	artifact
source ( <a href="#">hello.groovy</a> )	GroovyLexer	antlr tokens
antlr tokens	GroovyRecognizer	antlr ast
antlr ast	AntlrParserPlugin	groovy ast
groovy ast	AsmClassGenerator	bytecode ( <a href="#">hello.class</a> )

*Note1* [groovy.g](#) is used to generate [GroovyLexer](#) and [GroovyRecognizer](#)

*Note2* [GroovyRecognizer](#) is sometimes easier to understand in its [syntax diagram](#) form

*Note3* [AntlrParserPlugin](#) source available.

## Example

For these examples let's assume the file [hello.groovy](#) contains

```
class Hello {
    static void main(args) {
        println "hello world"
    }
}
```

### GroovyLexer (viewing Antlr Tokens)

To view the antlr tokens that the source code has been broken into you need to do the following in groovy-core subdirectory

```
java -cp
target/install/embeddable/groovy-all-1.6-beta-2.jar
org.codehaus.groovy.antlr.LexerFrame
```



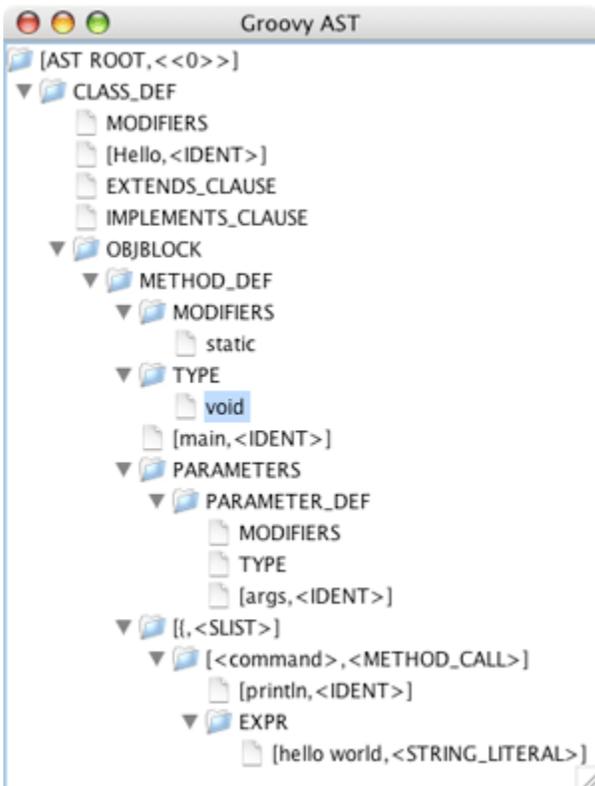
### GroovyRecognizer (viewing Antlr AST)

To view the antlr AST that the recognized tokens have built

```
java -cp
```

```
target/install/embeddable/groovy-all-1.6-beta-2.jar
```

```
org.codehaus.groovy.antlr.Main  
-showtree Hello.groovy
```

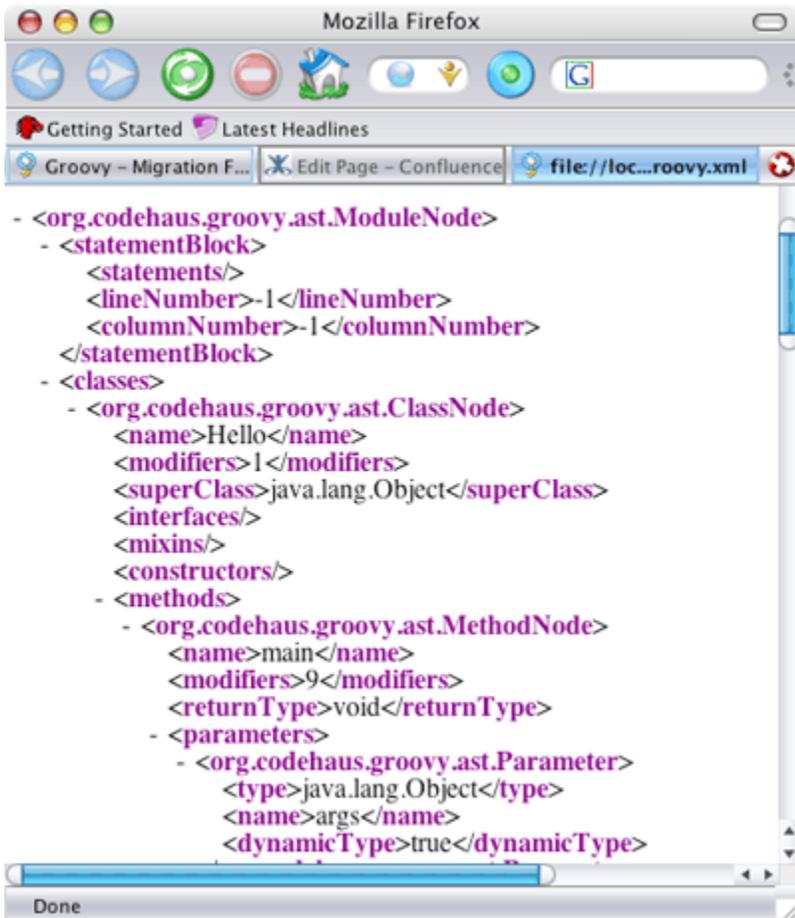


### AntlrParserPlugin (viewing Groovy AST)

To view the Groovy AST that is one step closer to the generated bytecode you can generate [hello.groovy.xml](#) using these unix commands.

This can be generated by changing the **groovy.ast** system property. By doing this we can diff the generated Groovy AST artifacts for debugging and migration purposes.

```
$ export JAVA_OPTS="-Dgroovy.ast=xml"
$ groovyc Hello.groovy
Written AST to Hello.groovy.xml
```

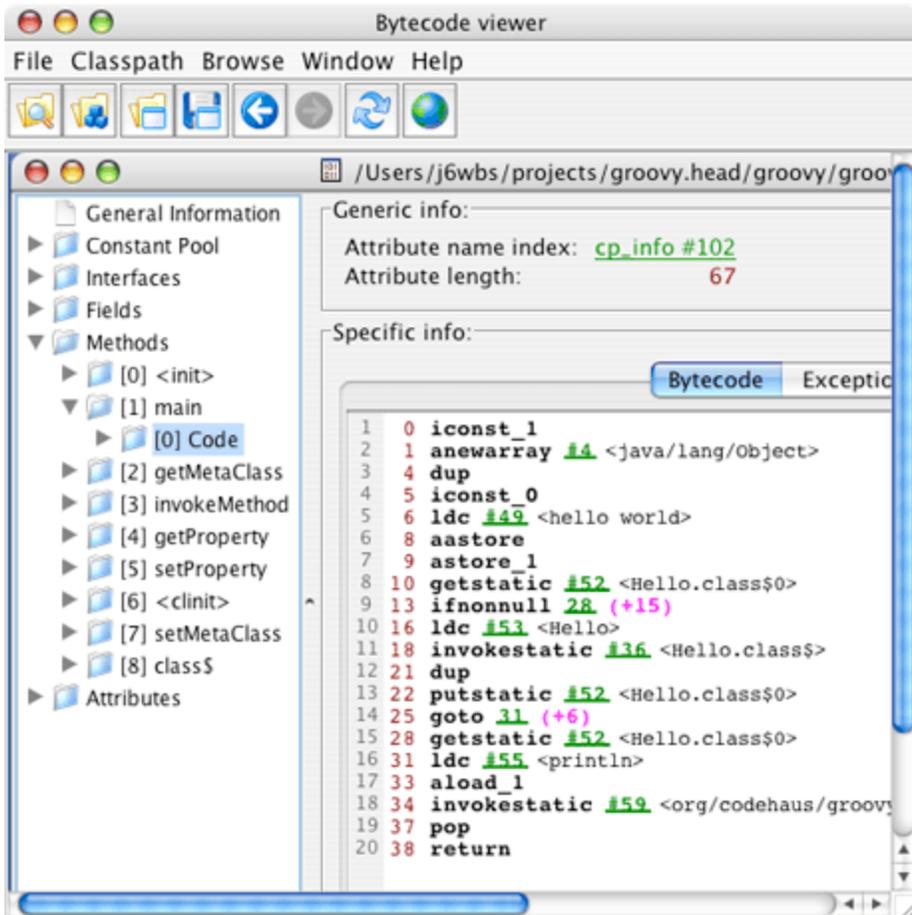


The screenshot shows a Mozilla Firefox browser window with the address bar displaying 'file://loc...roovy.xml'. The main content area shows the XML representation of the Groovy AST for the file 'Hello.groovy'. The XML structure is as follows:

```
- <org.codehaus.groovy.ast.ModuleNode>
- <statementBlock>
  <statements/>
  <lineNumber>1</lineNumber>
  <columnNumber>1</columnNumber>
</statementBlock>
- <classes>
- <org.codehaus.groovy.ast.ClassNode>
  <name>Hello</name>
  <modifiers>1</modifiers>
  <superClass>java.lang.Object</superClass>
  <interfaces/>
  <mixins/>
  <constructors/>
  - <methods>
    - <org.codehaus.groovy.ast.MethodNode>
      <name>main</name>
      <modifiers>9</modifiers>
      <returnType>void</returnType>
      - <parameters>
        - <org.codehaus.groovy.ast.Parameter>
          <type>java.lang.Object</type>
          <name>args</name>
          <dynamicType>true</dynamicType>
```

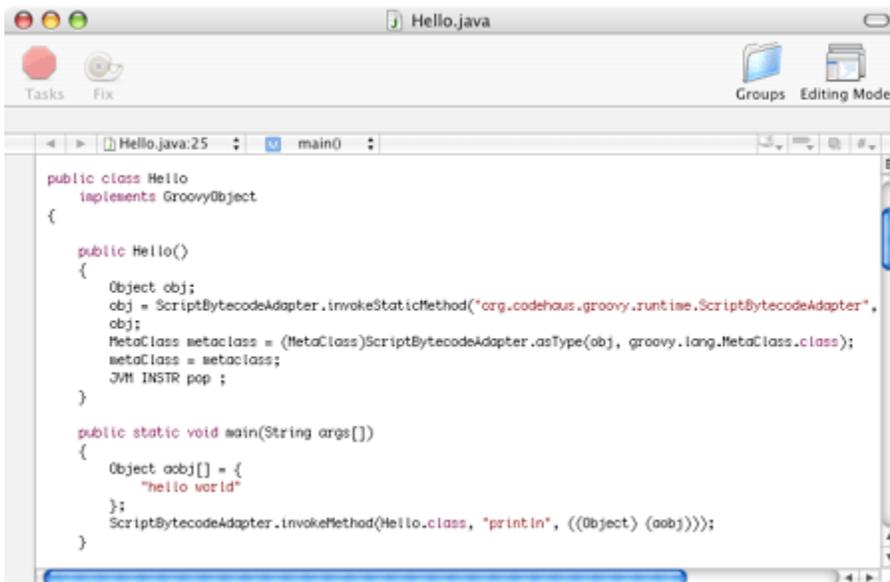
## Viewing Bytecode

One interesting bytecode viewer is [jclasslib](#) which renders [hello.class](#) in this manner...



## Decompiling bytecode back to Java

If, however, you are bamboozled by bytecode... The easiest to grok mechanism for reading the compiled code is to use a tool like [JAD](#) to decompile the [hello.class](#) into a readable [hello.java](#)



**Git**

**Groovy sources**

The sources of the Groovy project are hosted on Github: <https://github.com/groovy/groovy-core>

Additionally, the sources are mirrored on Codehaus' own Git infrastructure as well: <http://git.codehaus.org/gitweb.cgi?p=groovy-git.git>

You can learn about the repository details on the Xircles Codehaus admin interface: <http://xircles.codehaus.org/projects/groovy/repo/git/repo>

If you're interested in contributing, you can send us GitHub pull requests, or submit patches through JIRA. Please see our [contribution page](#) for more.

You can also get the sources for each releases in the form of a zip archive. Please head to our [download section](#) to download those source packages.

## Checkout the code

First of all, you'll need to have Git installed on your machine, whether through the support of your IDE, or as a command-line tool.

If you want to checkout the source code of Groovy, there are three different URLs you can use. From the command-line, you can use the command:

```
// anonymous access  
git clone  
git://github.com/groovy/groovy-core.git  
  
// read/write access  
git clone  
https://username@github.com/groovy/groovy-core.git  
  
git clone  
git@github.com:groovy/groovy-core.git
```

You can checkout different branches, in particular:

- `master` is the latest Groovy branch, for the upcoming major version
- `GROOVY_1_8_X` is the branch of the current Groovy 1.8.x versions (current stable version)
- `GROOVY_1_7_X` is the branch for the previous official version of Groovy 1.7.x

For fetching a branch the first time, simply use:

```
git fetch origin
```

To checkout a particular branch:

```
git checkout master  
git checkout GROOVY_1_8_X  
git checkout GROOVY_1_7_X
```

**i Developers:** Make sure your SSH information is up-to-date on Github or on Codehaus Xircles and that your SSH key is available to your command-line client or IDE integration. Please also make sure that you configured your email and name properly, by issuing the following commands:

```
git config --global user.name "Firstname Lastname"  
git config --global user.email "your_email@youremail.com"
```

### Committing your changes (developers)

Use the commit command to commit your changes locally:

```
git commit -m "Your commit message"
```

### Picking commits from another branch

Say you have committed your changes on `master` and want to merge a particular comming on `GROOVY_1_8_X`, you can procede as follows:

```
git checkout master
git commit -m "Fixed GROOVY-1234"
// this would return a12bc3... as commit
number

git checkout GROOVY_1_8_X
git cherry-pick a12bc3
```

### Staying up-to-date

To see what's the status of your source tree, you can call:

```
git status
```

And if you want to see all the latest commits that you have locally, you can do:

```
git log
```

To retrieve the changes that have been pushed to the server, you can do:

```
git pull
```

Of more explicitly:

```
git pull origin master
```

### Pushing your changes upstream

The various commits you've made are done locally, now is the time to share them with the world by pushing your changes to your Github clone, or to a publicly available Git repository:

```
git push  
git push origin master
```

If you're a Groovy despot, you can also push your changes to Codehaus for manual synchronization purpose. But for that, first, you'll have to have configured an additional remote with:

```
git remote add codehaus  
ssh://git@git.codehaus.org/groovy-git.git
```

Then you can push the changes back to GitHub as well:

```
git push codehaus master
```

### **Pushing a local branch to the remote repository**

To push a local branch to the Codehaus Git repository or on the GitHub mirror, you can do the following:

```
git push origin myLocalBranch  
git push codehaus myLocalBranch  
  
git push myRemoteLocation myLocalBranch
```

### **Merging pull requests from GitHub**

Contributors might bring their contributions in the form of "pull requests" on our GitHub mirror. Groovy despots can merge the pull requests on GitHub through the web interface by following this proposed workflow:

```
// create a new branch to test the pull
request
git checkout -b test_foopatch_onto_1_8
// now you are on branch
test_foopatch_onto_1_8
// let's add the git branch of the
contributor as a remote
git remote add someperson
git://github.com/groovy/somepersonsrepo.git
// fetch the particular commits
git fetch someperson
the_branch_with_the_changes
// merge it in our test branch
git merge FETCH_HEAD
// test the changes
gradle test or gradle test
// if all tests pass, then we can safely use
the web based merge UI of Github
```

### Additional links

If you want to learn more about Git, there are many available resources online, such as:

- the [GitHub cheatsheet](#)
- the [Pro Git book](#)

### Groovy Backstage

#### How Groovy works behind the scenes

With your Groovy scripts, you have multiple options on how to use them:

- run groovyc on them to create Java \*.class files
- evaluate the script at runtime (either from a file or from a Java String)
- make your script available as Java Class-objects through the GroovyClassLoader (no \*.class files generated!)

*No \*.java source files are ever generated.*

All the above may seem like magic and it somehow is: the magic of imaginative, mindful software engineering.

## **Everything starts with the Groovy grammar.**

*There is the notion of the 'new' and the 'old' parser. Only the new one (as of Feb.05) is described here.*

The Groovy grammar is written in terms of an ANTLR (= *ANother Tool for Language Recognition*) grammar. The tool can handle grammars of type LL(k), where the Java grammar is of type LL(2) and Groovy is of type LL(3).

The difference is in the number of tokens that the parser needs to look ahead for recognizing e.g. "==" (2 tokens for Java) or "===" (3 tokens for Groovy). To be more correct, the problem is in recognizing the first "=" character. The parser needs to "look ahead" to derive its meaning.

ANTLR formulates the grammar in terms of "rules" that fully implement EBNF (Extended Backus-Naur Form) enriched with Java code blocks, special functions and some other things.

With the recognition of a rule, actions can be triggered that come as usual Java code. ANTLR dumps out Javacode representing a parser capable to recognize the given grammar. And this parser executes the embedded code blocks from the grammar - the "action".

## **Parser Generation and AST**

ANTLR takes the Groovy grammar file "Groovy.g" to create the Groovy parser. When the parser is fed with the source code of a Groovy script, it produces the AST (= *Abstract Syntax Tree*) that represents that code as a run-time structure.

## **Byte Code Generation**

From the AST, it is possible to create Java Byte Code: either for making it persistent as \*.class files or for making it directly available as Class objects through the GroovyClassLoader.

This ClassGeneration is done with the help of objectweb's ASM tool. (The ASM name does not mean anything: it is just a reference to the "asm" keyword in C, which allows some functions to be implemented in assembly language.)

ASM provides a Java API to construct or modify Byte Code on a given AST.

The API for bytecode generation heavily relies on the Visitor Pattern. The main entry point for the class generation is `org.org.codehaus.groovy.classgen.AsmClassGenerator.java`.

It is a large class. There are visitXYExpression methods called when converting the AST to bytecode. For example visitArrayExpression is called when creating arrays in bytecode.

## **More Links on the topic**

### [Groovy Method Invokation](#)

Tools:

- <http://www.antlr.org/>
- <http://asm.objectweb.org/>

General:

- <http://compilers.iecc.com/comparch/article/99-02-109>
- [http://en.wikipedia.org/wiki/Category:Parsing\\_algorithms](http://en.wikipedia.org/wiki/Category:Parsing_algorithms)

- <http://lambda.uta.edu/cse5317/spring02/notes/notes.html>

## Groovy Method Invokation

### *Where it all starts*

Say you have a Groovy code like

```
println 'hi'
```

As explained in [Groovy Backstage](#), there is bytecode generated to achieve the desired behaviour of printing to stdout.

The easiest way of looking at the generated bytecode is to groovyc your Groovy source to a class file and process it with a Java Decompiler (e.g. JAD). See also: [From source code to bytecode](#)

The resulting code looks as follows (only the relevant snippet):

```
...
Object aobj[] = { "hi" };
ScriptBytecodeAdapter.invokeMethod(this,
    "println", ((Object) (aobj)));
...
```

### *Invocation scheme*

There is a delegation scheme like

- `ScriptBytecodeAdapter.invokeMethod(...)` (static method)
  - `InvokerHelper.invokeMethod(...)` (static method)
    - `Invoker.invokeMethod(...)` (instance method called on `InvokerHelper`'s single instance)

### *MetaClass and MetaClassRegistry*

`Invoker` calls `invokeMethod(...)` on the **MetaClass** of our class (with exceptions, see below). It finds this **MetaClass** by looking it up in the **MetaClassRegistry**. The `Invoker` holds a single instance of this registry.

 When working with the `MetaClassRegistry`, `InvokerHelper.getInstance().getMetaRegistry()` is the only instance, you should ever use.

Exceptions (when `MetaClass.invokeMethod(...)` is not used):

- for Closures, `Closure.invoke(...)` is used
- for GroovyObjects `obj` of type `GroovyInterceptable`, `obj.invokeMethod(methodName, asArray(arguments))` is called

- for any other `GroovyObject obj` when method invocation through its `MetaClass` fails, `obj.invokeMethod(methodName, asArray(arguments))` is called

`MetaClass.invokeMethod(...)` finally cares for the invocation, either by reflection or by dynamic bytecode generation. Dynamic bytecode generation is supposed to be faster. For a class `MyClass` it generates `gjdk.groovy.lang.MyClass_GroovyReflector` with an `invoke` method.

*Does `MyClass_GroovyReflector` contain methods according to `MyClass.groovy` that can be called directly ?*

The cool thing about `MetaClass` is that you can dynamically add or remove methods to it. One can even replace the whole `MetaClass` in the `MetaClassRegistry`. See `ProxyMetaClass` for an example.

back to [Groovy Backstage](#)

## Groovy Internals

This page is work in progress. It will document the internals of groovy, the ideas and the techniques used so other developers may it have more easy to contribute to groovy in the future.

### Parser

### Package Layout

### Bytecode Hints

### ClassLoader Structure

#### *One Class One Class Loader*

When are two classes the same? If the name is equal and if the class loader is equal. This means you can have multiple versions of the same class if you load the class through different class loaders. Of course this versions don't really have to be the same. This also means if you have code like

### DuplicateClass.groovy

```
Class cls = foo.class
assert cls.name==Foo.class.name
assert cls==Foo.class
```

may fail because `cls` is not `Foo`. This also means calling a method like

## DuplicateClass.groovy

```
def f(Foo f){
    println "f(Foo) called"
}

def f(Object o){
    println "f(Object) called"
}
```

with an Foo does not mean that "f(Foo) called" will be printed! This is no secret, you will find it in the language specification.

### *Class Loading Conventions*

There are small conventions about how to do class loading in Java.

1. always return the same class for the same name
2. use your cache before asking a parent loader
3. no parent loader doesn't mean the system loader
4. ask your parent loader before trying to load a class by yourself

Most of the loaders in groovy are violating one or more of these rules. I will explain why and how in the next sections

### *RootLoader*

First let us start with the RootLoader. This one used used to start the console programs and is something like an advanced java launcher. When using Groovy embedded this loader is usually not used. The RootLoader sees itself as a root in a class loader tree. Against the convention to ask the parent before loading a class this loader tries to load it by itself first. Only when this fails the parent is asked. The parent is usually the system class loader then. This step was needed because of many problems with multiple versions of libs, causing multiple classes of the same name where they are not expected, or classes loaded from the wrong libs. Because of that reason this loader may also be used in ant or maven to avoid clashes with the classes used by ant.

### *GroovyClassLoader*

### *ReflectorLoader*

## Optional Typing

### *Assignment*

If you assign a value to a typed variable in Groovy, then this assignment will include an implicit cast. A Declaration counts as assignment in that sense as well.

## Assignment.groovy

```
String x = 8
```

is in fact

## AssignmentExpanded.groovy

```
String x = (String) 8
```

Note: there is nothing to be done but boxing, if the cast is to Object, since everything in Groovy is an Object. The cast is realized in Groovy by writing `ScriptByteCodeAdapter.castToType(8, String.class)`, which in turn will call [DefaultTypeTransformation.castToType\(Object, Class\)](#).

### *Casting Rules in General*

The following rules apply in the order given.

Casting rules for identity:

- value null stays null
- a cast to Object gives the value itself
- if the value class is equal the casting class, nothing is changed
- if the value class is a subclass of the casting class, nothing is changed
- if we cast to an array, see array casting rules
  - These cases will cause a new value being created from the old one if the cast type is a Collection:
- a cast to HashSet will create a new HashSet using the `HashSet(Collection)` constructor
- if the value is an array and the we cast we try to invoke first the parameterless constructor of the cast type, to then add each element of the array using `add(Object)`. Should there be no such constructor or its invocation fail, we fail with a `GroovyCastException`
- a cast to String will invoke `toString()` on the value
- a cast to char or Character will invoke the char casting rules
- a cast to Boolean or boolean will invoke the rules for Groovy Truth, see paragraph later
- a cast to Class we will use the `toString()` representation of value to execute `Class.forName` with that. The `ClassLoader` used for this will always be the loader for `DefaultTypeTransformation.class`.
- if we cast to a Number or primitive number type we will try to convert value to a number and then use one of the methods to create the kind of value we actually need. Should value not be a number already, but a Character, we use the `charValue()` for further steps. In case of a GString or String, we use the value of the one element String, if existing. Should it not exist, we fail with a `GroovyCastExpression`. If value is neither String, GString, Character or a Number, we fail with a `GroovyCastExpression`. A resulting Double with positive or negative infinity is only allowed, if the original value is already a double value (this may for example fail for the case of a BigDecimal, that does not fit into a Double). If not we fail with a `GroovyCastException`. `BigDecimal` and `BigInteger` are created by the `doubleValue()` in case we cast from a float or double. In the other cases we use the `toString()` representation.
- if we cast a String or GString value to an Enum type, we call `Enum#valueOf` to get an enum value based on the given string representation of that enum value

- if the value is an Collection, Map or Object[], we try to invoke the cast type constructor. In case of a Object[] or Collect, the elements are used for the arguments. A Map will be given as single argument.
- if we reach this point, the cast fails with a GroovyCastException

#### **Char Casting Rules**

- if the value is a Char, nothing is to be done
- if the value is a Number, we use the int value to create a char
- otherwise we execute toString() to get a String and if that String is a one element String, we will use that element to create the char. Is the String longer we fail with a GroovyCastException

#### **Casting Rules for Boolean / Groovy Truth implementation**

The implementation of this is determined by calling the asBoolean method of the value. It will realize those cases:

- if value is null: false
- empty Collections, arrays, CharSequence (contains empty String): false
- boolean false: false
- numbers equaling to 0: false
- the char0 : false
- Enumeration that has no more elements: false
- Matcher in which the find method returns false: false
- in the other cases we return true

#### **Casting to an Array**

We try to create a collection from the value (see collection casting rules), create an array of the length of that collection and then set each element of the array with the according element of the collection after we cast the collection item to the component type of our array. This is realizing a recursive call to the original casting logic.

#### **Casting to a Collection**

- null values are represented by Collections.EMPTY\_LIST
- collection values are themselves
- a Map is represented as its entry set
- an array by transforming the array into a collection
- a MethodClosure using

```
MethodClosure method = (MethodClosure) value;
    IteratorClosureAdapter adapter = new
IteratorClosureAdapter(method.getDelegate());
    method.call(adapter);
    return adapter.asList();
```

, but might be broken and untested.

- a String or GString by a list of characters
- a File by its text lines
- an enum by the values() method
- all other cases by Collections.singletonList(value)

## **Release Process**

### **Instructions (for 2.0+ releases)**

### **i** JDK 7 mandatory

Make sure you are **using a Java 7 JDK**, otherwise Groovy would be compiled without invoke dynamic support

- Update the pdf of the online documentation and commit it
- perform a clean checkout into a new, empty directory

```
mkdir groovy-release
cd groovy-release
git clone git@github.com:groovy/groovy-core.git
```

- create a temporary release branch and change to it

```
cd groovy-core/
(optional) git checkout branch_to_release_from
git checkout -b release
```

- Change the release version information

```
./gradle.properties:
  groovyVersion = 2.0.0-rc-1
  groovyBundleVersion = 2.0.0-rc-1
```

- commit the change

```
git commit -a -m "RC-1 version change"
```

- Tag with that new release

```
git tag -a GROOVY_2_0_5 -m "tagging release of Groovy 2.0.5"
```

- Test the release

```
./gradlew clean test
```

If everything passes, proceed to the next step. Otherwise, release is not possible and problems should be fixed first.

- Build zips and jars using:

## `./gradlew clean dist`

- Upload all the zips (but not jars) to the WebDAV distribution site (<https://dav.codehaus.org/dist/groovy/distributions>), for example, or through rsync if you're authorized:

```
rsync -vlogDrzP ./target/distributions/*.zip
$USER@groovy.codehaus.org:/projects/groovy/dist/
```

(just check they are in the right place afterwards; you may still need a webdav client to move them into the right subdirectories if something goes wrong)

- Upload the javadocs through rsync to the WebDAV web site (<https://dav.codehaus.org/groovy/>), for example:

```
rsync -vlogDrzP ./target/html/*
$USER@groovy.codehaus.org:/projects/groovy/web/
```

- Put m2 jars into right place for uptake into repo1 (see *Publishing artifacts* on [Building Groovy from Source](#) for more details):

Prior to uploading, you must make sure you have rights to upload artifacts to the CodeHaus repository. If so, you must have two system properties set before uploading:

```
groovy.deploy.username
groovy.deploy.password
```

Then you may upload artifacts using the following command (note there are lots of artifacts to upload, do not be surprised if it takes more than an hour):

```
./gradlew clean uploadArchives
```

Or with the user / password passed as properties:

```
./gradlew -Dgroovy.deploy.username=USER -Dgroovy.deploy.password=PASSWORD
clean uploadArchives
```

Should there be a problem when uploading the jars, check that there is a file `~/m2/settings.xml` containing:

```
<?xml version="1.0"?>
<settings>
  <servers>
    <server>
      <id>codehaus.org</id>
      <username>USER</username>
      <password>PASSWORD</password>
    </server>
  </servers>
</settings>
```

Where USER and PASSWORD are replaced if the right values.

- If you are affected by authentication issues, due to the [odd certificates used by Codehaus](#), and if the installation of those certificates are problematic on your machine (if the certificates are rejected or if you're not running as root, etc), you can use a local keystore:

```
sudo $JAVA_HOME/bin/keytool -import -alias StartSSL-CA -file
~/Downloads/startssl-CA.pem -keystore ~/.keystore
sudo $JAVA_HOME/bin/keytool -import -alias StartSSL-Intermediate -file
~/Downloads/startssl-Intermediate.pem -keystore ~/.keystore

export JAVA_OPTS="-Xmx2048M -Djavax.net.ssl.keyStore=$HOME/.keystore
-Djavax.net.ssl.keyStorePassword=changeit
-Djavax.net.ssl.trustStore=$HOME/.keystore
-Djavax.net.ssl.trustStorePassword=changeit"

./gradlew -Dgroovy.deploy.username=USER -Dgroovy.deploy.password=PASSWORD
uploadArchives
```

- remove the release branch and push the release tag

```
git checkout <branch you released from>
git branch -D release
git push --tags
```

- advance the version in your branch in gradle.properties (don't forget the -SNAPSHOT)
- Update the download links: <http://docs.codehaus.org/display/GROOVY/Download>
- Update the versions.json file in <http://dist.groovy.codehaus.org/> through webdav to add a reference to the new version, so that GVM can automatically pickup new releases

```
{
  "version": "2.1.1",
  "link":
  "http://dist.groovy.codehaus.org/distributions/groovy-binary-2.1.1.zip",
  "type": "final",
  "default": true
}
```

- Announce the release (mailing-list, Groovy blog, twitter, Google+...)
- Update the release version information to the next snapshot

if the certificate needs an update see: <https://docs.codehaus.org/display/HAUSMATES/Installing+a+new+CA+into+the+JDK>

### Instructions (for pre-2.0 releases)

- Update the pdf of the online documentation and commit it
- create a temporary release branch and change to it

```
git branch release
git checkout release
```

- Change the release version information

```

                                     "files"
-----
./build.properties:
  groovyVersion = 1.8.9
  groovyBundleVersion = 1.8.9
./gradle.properties:
  groovyVersion = 1.8.9
  groovyBundleVersion = 1.8.9
.pom.xml:
  <version>1.8.9</version>
```

- commit the change
- Tag with that new release

```
git tag -a GROOVY_1_8_9 -m "tagging release of Groovy 1.8.9"
```

- Build zips and jars using:

```
ant clean dist
```

You may need to set `ANT_OPTS=-Xmx512m` (or similar) if you run out of memory when running GroovyDoc.

- Upload all the zips (but not jars) to the WebDAV distribution site (<https://dav.codehaus.org/dist/groovy/distributions>), for example, or through rsync if you're authorized:

```
rsync -vlogDrzP ./target/dist/*.zip
$USER@groovy.codehaus.org:/projects/groovy/dist/
```

(just check they are in the right place afterwards; you may still need a webdav client to move them into the right subdirectories if something goes wrong)

- Put m2 jars into right place for uptake into repo1 (see *Publishing artifacts* on [Building Groovy from Source](#) for more details):

```
ant -DskipTests=true deploy
```

If releasing Groovy 1.6, also do this:

```
ant -DskipTests=true -DforceRetro=true -Djdk14home=... deploy // for
jdk14 artifacts also
```

(Note 1: there is currently some duplicated work done by 'ant dist' above and 'ant deploy' here which we need to eventually consolidate)

Should there be a problem when uploading the jars, check that there is a file `~/m2/settings.xml` containing:

```
<?xml version="1.0"?>
<settings>
  <servers>
    <server>
      <id>codehaus.org</id>
      <username>USER</username>
      <password>PASSWORD</password>
    </server>
  </servers>
</settings>
```

Where USER and PASSWORD are replaced if the right values.

- Upload the javadocs through rsync to the WebDAV web site (<https://dav.codehaus.org/groovy/>), for example:

```
rsync -vlogDrzP ./target/html/*
$USER@groovy.codehaus.org:/projects/groovy/web/
```

- remove the release branch and push the release tag

```
git checkout <branch you released from>
git branch -D release
git push --tags
```

- advance the version in your branch in pom.xml, build.properties and gradle.properties (don't forget the -SNAPSHOT)
- Update the download links: <http://docs.codehaus.org/display/GROOVY/Download>
- Announce the release (mailing-list, Groovy blog, twitter, Google+...)
- Update the release version information to the next snapshot

if the certificate needs an update see: <https://docs.codehaus.org/display/HAUSMATES/Installing+a+new+CA+into+the+JDK>

## Setup Groovy Development Environment

Steps to setup development environment for Groovy core:

1. Install JDK 1.4.2 or above (e.g. Sun JDK 5.0 version 1.5.0\_06):
  - JDK is available on <http://java.sun.com>.
  - Remember to set the JAVA\_HOME environment variable after installation.
  - (Optional) Add JAVA\_HOME/bin to PATH for executing Java commands.
2. If you want to use Eclipse then install Eclipse 3.1 or above (e.g. Eclipse 3.2):
  - Eclipse is available on <http://www.eclipse.org>.
  - Install a Git plugin
3. If you want to build from the command line the install Ant 1.7 (possibly works with 1.6.5 but this is not properly tested):
  - Ant 1.7 is available on <http://ant.apache.org/>.
4. Extract groovy source, either:
  - If using Eclipse:
    - Decide where your workspace is (will be reference later), switch to the Git perspective. If you are not familiar with Eclipse, in menu Help -> Welcome may help you a lot.
    - Browse [Git](#) to know Git information to checkout sources.
    - Create a new repository, extract groovy source (the folder groovy-core. You can extract it as project name groovy)
  - If not using Eclipse then from command line: `svn co http://svn.codehaus.org/groovy/trunk/groovy/groovy-core`
5. Build groovy source
  - For command line builds, simply go to the groovy-core directory you extracted, execute "ant" in command line, make sure the build success, browse groovy/target/test-reports if you got JUnit test fails.
6. Done!

## Cookbook Examples

Larger examples of using Groovy in the Wild with a focus on applications or tasks rather than just showing off the features, APIs or modules:

- [Accessing SQLServer using groovy](#)
- [Alternate Spring-Groovy-Integration](#)
- [Auto setup and download dependencies jars](#)

- [Batch Image Manipulation](#)
- [Calculating a SHA1 hash for large files](#)
- [command line groovy doc or methods lookup](#)
- [Compute distance from Google Earth Path \(in .kml file\)](#)
- [Concurrency with Groovy](#)
- [Convert SQL Result To XML](#)
- [Embedded Derby DB examples](#)
- [Embedding a Groovy Console in a Java Server Application](#)
- [Executing External Processes From Groovy](#)
- [Expect for Groovy](#)
- [Formatting simple tabular text data](#)
- [Greedy Coin Changer in Groovy](#)
- [Groovy Alternatives to Inner Classes](#)
- [Groovy as script](#)
- [Guy Steele's word split example using Groovy and GPar](#)
- [How many Triangles \(list, regex, grep\) example](#)
- [Installing or Update Groovy Automatically](#)
- [InstallTextMateGroovyBundle](#)
- [Integrating Groovy in an application - a success story](#)
- [Iterator Tricks](#)
- [Martin Fowler's closure examples in Groovy](#)
- [Other Examples](#)
- [Parsing Groovy Doc Online](#)
- [Plotting graphs with JFreeChart](#)
- [PoorMansMixins](#)
- [Reading from a Blob](#)
- [Recipes For File](#)
- [Regex search for a class in the JVM runtime classpath](#)
- [Search one or more jar files for a specified file or directory](#)
- [Simple file download from URL](#)
- [Solving Sudoku](#)
- [StreamingMarkupBuilder and IMAP client example](#)
- [SwingBuilder with custom widgets and observer pattern](#)
- [Tomcat tools](#)
- [Unsign Jar Files \(Recursively\)](#)
- [Using Groovy with Boost](#)
- [Using Hibernate with Groovy](#)
- [Using JGoodies Animation with Groovy](#)
- [Using JScience with Groovy](#)
- [Using MarkupBuilder for Agile XML creation](#)
- [Using Regex Patterns to find word phone numbers](#)
- [Using the Delegating Meta Class](#)
- [Using the Eclipse Modeling Framework \(EMF\)](#)
- [Using the Eclipse UML2 Framework](#)
- [Using the Proxy Meta Class](#)
- [Windows Look And Feel for groovyConsole](#)
- [WrappingGroovyScript](#)
- [Writing to a Blob](#)
- [Yaml and Groovy](#)

Additional real-world example for the German readers: <http://berndschiiffer.blogspot.com/2007/03/groovy-im-fluss-ein-beispiel-aus-der.html>

## Accessing SQLServer using groovy

\*DISCLAIMER: \*Use at your own risk. Author is not responsible for any damages resulting from direct or indirect use of the instructions here.

### Accessing Microsoft SQLServer using Groovy

This is an example of how to access Microsoft SQL Server (2000) database using groovy to create reports on a unix box using a Microsoft JDBC driver. The instructions to install the JDBC driver itself are given in **Appendix A**. The script name is **queryMSSQL.groovy**. Assumptions are:

- The script takes arguments one or more queryfiles and executes them against a Microsoft SQLServer 2000 database defined using options on the command line.
- host on which SQLServer resides is reachable from the unix host and that there are no firewall issues.
- All queries have one bind variable, which is satisfied by the argument to option **-v**
- USAGE: **groovy queryMSSQL.groovy -h -s sqlserverhost [-P port] -u userid -p password -v value -t textfile queryfile [queryfile]**
- Option / arguments info:
  1. **-P port** - denotes the port where SQLServer is listening
  2. **-u userid\*** - \* denotes userid (on SQLServer)
  3. **-p password** - denotes password for the userid on SQLServer
  4. **-v value** - value to satisfy bind variable (in a where clause eg. WHERE col = ...). If no ? is seen in queryfile, then no bind variables are involved. In this case the value passed should be **none**.
  5. **-t textfile\* \*** - The name of text file where output would go
  6. **queryfile** - A file containing query

## queryMSSQL.groovy

```
import java.sql.Connection
import java.sql.DriverManager
import javax.sql.DataSource
import groovy.sql.Sql
def cli = new CliBuilder( usage: 'groovy
queryMSSQL.groovy -h -s sqlserverhost [-P
port] -u userid -p password -v value -t
textfile queryfile [queryfile]...' )
cli.h(longOpt:'help', 'usage information')
cli.s(argName:'servername',
longOpt:'server', args:1, required:true,
type:GString, 'sqlserverhost')
```

```
cli.P(argName:'port', longOpt:'port',
args:1, required:false, type:GString,
'port')
cli.u(argName:'userid', longOpt:'userid',
args:1, required:true, type:GString,
'userid')
cli.p(argName:'password',
longOpt:'password', args:1, required:true,
type:GString, 'password')
cli.v(argName:'value', longOpt:'value',
args:1, required:true, type:GString,
'value')
cli.t(argName:'textfile', longOpt:'text',
args:1, required:true, type:GString, 'text
file')
def opt = cli.parse(args)
if (!opt) return
if (opt.h) cli.usage()
def port = 1433
if (opt.P) port = opt.P // If the port was
defined
def servername = opt.s
def userid = opt.u
def password = opt.p
def valuetobind = opt.v
def textfile = opt.t
def outFile
def outFileWriter
try {
    outFile = new File(textfile)
    outFile.write(""); // truncate if
```

```

output file already exists
} catch (Exception e) {
    println "ERROR: Unable to open $textfile
for writing";
    return;
}
driver =
Class.forName("com.microsoft.jdbc.sqlserver.
SQLServerDriver").newInstance();
Connection conn =
DriverManager.getConnection("jdbc:microsoft:
sqlserver://$servername:$port", userid,
password);

try {
    if (args.length == 0) {
        usage_error = "Error: Invalid number
of arguments"
        usage_error =
"\n$usage_error\nUSAGE: groovy
queryMSSQL.groovy queryfile\n"
        throw new
IllegalArgumentException(usage_error)
    }
    Sql sql = new Sql(conn)
// After options processing the remaining
arguments are query files
// Go through the query files one at a time
for execution
    for (queryfilename in opt.arguments()) {
        queryfile = new File(queryfilename)

```

```

        query = "" // initialize the query
string
        param_count = 0 // Number of
placeholders needed for parameters to query
        pattern = /\?/ // pattern to look
for to find number of parameters
        // read the query from the query
file (line by line) and build it
        queryfile.eachLine { it ->
            query += " " + it
        }
        // number of bind variables to satisfy
is obtained by number of ? seen in the query
        query.eachMatch(pattern) {
param_count++ }
        println '-.' * 40
        println "query is ${query}"

        println "Output is:"
        println '=' * 80
        def count = 0 // row count
        paramlist = []
        if (valuetobind != "none")
            1.upto(param_count) { paramlist
<< valuetobind }
            sql.eachRow(query, paramlist) { row
->
                count++; // increment number of
rows seen so far
                //println "$count. ${row.name}"
// print out the column name

```

```
        recstr = "" // initialize the
string that represents row
        meta = row.getMetaData() // get
metadata about the row

        for (col in 0..<meta.columnCount)
{
        // record is stored in a
string called recstr
        if (recstr == "") {
            recstr = row[col]
        }
        else {
            recstr += "," + row[col]
        }
    }

    outFile.append(recstr + "\n")
}
}
conn.close()
} catch(Exception e) {
    print e.toString()
}
```

```
}  
finally {  
}
```

## Appendix A - Installing the Microsoft JDBC driver on unix

These notes are based on instruction provided in <http://support.microsoft.com/kb/313100>.

- Download SQL Server 2000 Driver for JDBC Service Pack 3. This is done by getting the file mssqlserver.tar from Microsoft site:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=07287B11-0502-461A-B138-2AA54BFDC03A&displaylang=en>

- Upload the tar file mssqlserver.tar to \$HOME/download (choose a suitable directory).
- Extract the files from mssqlserver.tar using **tar xvf mssqlserver.tar**
- Make a directory where the JDBC driver will be installed (say \$HOME/mssqljdbcsp3) using **mkdir \$HOME/mssqljdbcsp3**
- Change to \$HOME/download and run **./install.ksh**
- When prompted for the installation directory choose **\$HOME/mssqljdbcsp3**. This results in the message: SQL Server 2000 driver for JDBC is installed in the following location: \$HOME/mssqljdbcsp3
- Set the CLASSPATH variable in the startup file (**.login** or **.profile**) to include the following jar files:

1. \$HOME/lib/msbase.jar
2. \$HOME/lib/mssqlserver.jar
3. \$HOME/lib/msutil.jar

In Bourne/Korn shell CLASSPATH can be appended to using:

```
export CLASSPATH="$CLASSPATH:$HOME/lib/msbase.jar:$HOME/lib/mssqlserver.jar:$HOME/lib/msutil.jar"
```

## Alternate Spring-Groovy-Integration

### Abstract

Alternate way to integrate Groovy in a Java-Application using Spring, without the usage of `<lang:groovy/>`-Tags.

### Advantages

- Works with Spring-AOP
- You can use other Groovy-Classes e.g. internal datatypes, in your Beans without the need to inject them.

### Example

## Java Code that's create the App-Context

```
...
private GenericApplicationContext
createAppContext(String path) {
    GenericApplicationContext ctx = new
GenericApplicationContext();

        GroovyClassLoader defaultClassLoader
= new
GroovyClassLoader(this.getClass().getClassLo
ader());

        XmlBeanDefinitionReader reader = new
XmlBeanDefinitionReader(ctx);
    reader.setBeanClassLoader(defaultClassLoade
r);
    reader.loadBeanDefinitions(path);
    ctx.refresh();
    return ctx;
}
...
```

## Sample beans.xml

```
<bean id="tm"
class="ticketing.bo.impl.TicketManagerImpl"/
>
```

## ticketing/bo/impl/TicketAssemblerImpl.groovy

```
class TicketAssemblerImpl implements
IBaseAssembler {
    ...
}
```

Hopefully that is helpful...

### Auto setup and download dependencies jars

this script [SetupGroovyLib.groovy](#) provide the following:

```
groovy SetupGroovyLib.groovy -h
```

```
Usage SetupGroovyLib jarname [jarname...]
```

```
The jarname is in format of Maven2
artifacts groupId:artifactId:version name.
```

```
You may use site like
http://www.mvnrepository.com to help you
find the jarname.
```

```
This script default arguments to:
```

```
    javax.activation:activation:1.1
```

```
    javax.mail:mail:1.4
```

```
    mysql:mysql-connector-java:5.1.5
```

```
These jars will allow you to send emails
and access to MySQL database!
```

NOTE: If you are using groovyConsole, you will need to restart it as it won't pick up any new jar added to ~/.groovy/lib automatically.

See also [Installing or Update Groovy Automatically](#)

**More examples**

## Using commons-lang

Run `groovy SetupGroovyLib.groovy commons-lang:commons-lang:2.3`

Now try out:

```
import org.apache.commons.lang.*  
StringUtils.splitByWholeSeparator("aXXbXXc",  
"XX")
```

## Batch Image Manipulation

An example image manipulation script, It's not that fresh anymore but it was laying around when at the sametime some people want to see more examples.

So here is my little contribution.

Note to the groovy gurus: find more groovier ways.

```
/**  
 * A batch image manipulation utility  
 *  
 * A wrote this script just to get groovy,  
 batch manipulate images in about  
 * 240 lines of code (without this  
 comment)!!!.  
 *  
 * commands:  
 * values ending with '%' means size  
 relative to image size.  
 * values ending with 'px' means values in  
 absolute pixels.  
 * values without postfix use default  
 notation.  
 *  
 * expressions:  
 * scale(width,height) * height is
```

```

optional(use width) e.g: scale(50%) ==
scale(50%,50%)
* fit(width,height)          * relative scale
the image until if fits (default as scale)
*                          * bounds of the
given box, usefull for generating of
thumbnails.
* rotate(degrees,x,y)        * the rotation
position x and y are optional (default is
50%)
*
* TODO: move(x,y)           * move the image
within its own bounds (can be done with
margin)
*                          * y is
optional(same height)
* TODO: color(type)        * color
transformation
* TODO: shear(degrees,x,y) * x and y is
optional
* margin(x,y,x2,y2)        * add margins to
image (resize image canvas), this operation
can't
*                          * be used on a
headless environment.
* parameters:
* -d                        * working
directory (default current directory)
* -e                        * execute
expressions from command line.
* -f                        * execute

```

expressions from file.

```
* -p * file mathing
pattern default is \.png|\.jpg
* -q * output file
pattern can use {0} .. {9}
* * backreferences
from the input pattern. default: output/{0}
* -h * help, nothing
special (maybe this doc using heredoc)
*
* Example generate thumbnails(take *.png
from images fit them in a 100X100 box,
* add 10px margin, put them in the
thumbnail dir.)
*
* $ groovy image.groovy -d images -e
"fit(100px,100px) margin(5)" -p "(.*)\.png"
-q "thumbnail/{1}.png"
*
* @author Philip Van Bogaert alias tbone
*/
```

```
import java.io.*;
import javax.imageio.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.geom.*;
import java.util.*;
```

```
class GroovyImage {
```

```

File srcDir = new File(".")
def operations = []
def pattern = ~".*(\\.png|\\.jpg)"
def outputPattern = "output/{0}"

void addOperation(command) {

    def matcher = command =~
"([a-z]+)\\.\\.((.*)\\.\\.)*";
    matcher.find();

    def method = matcher.group(1);
    def args =
matcher.group(2).split(",").toList();

    switch(method) {
        case "scale": //
vertical, horizontal
operations.add([this.&parseAndScale, argsLength(args, 2)]);
                break;

        case "rotate": // degrees, x, y
operations.add([this.&parseAndRotate, argsLength(args, 3)]);
                break;

        case "margin": //
left, top, right, bottom

```

```
operations.add([this.&parseAndMargin, argsLength(args, 4)]);
```

```
    break;
```

```
    case "fit": // width,height
```

```
operations.add([this.&parseAndFit, argsLength(args, 2)]);
```

```
    break;
```

```
    }
```

```
  }
```

```
    BufferedImage
```

```
parseAndRotate(image, degrees, x, y) {
```

```
    def parsedRadians = 0;
```

```
    try {
```

```
        parsedRadians =
```

```
Math.toRadians(Double.parseDouble(degrees));
```

```
    }
```

```
    catch(NumberFormatException except)
```

```
{
```

```
    }
```

```
    def parsedX =
```

```
parseValue(x, image.width, true, "50%");
```

```
    def parsedY =
```

```
parseValue(y, image.height, true, parsedX);
```

```
    return
```

```
rotate(image, parsedRadians, parsedX, parsedY);
```

```

    }

    BufferedImage rotate(image, radians, x, y)
{
    def transform = new
AffineTransform();
    transform.rotate(radians, x, y);
    def op = new
AffineTransformOp(transform, AffineTransformO
p.TYPE_BILINEAR);
    return op.filter(image, null);
}

```

```

    BufferedImage
parseAndScale(image, horizontal, vertical) {
    def parsedHorizontal =
parseValue(horizontal, image.width, false, "100
%");
    def parsedVertical =
parseValue(vertical, image.height, false, parse
dHorizontal);
    return
scale(image, parsedHorizontal, parsedVertical)
;
}

```

```

    BufferedImage
scale(image, horizontal, vertical) {
    def transform = new
AffineTransform();

```

```
transform.scale(horizontal,vertical);
    def op = new
AffineTransformOp(transform,AffineTransformO
p.TYPE_BILINEAR);
    return op.filter(image,null);
}
```

```
    BufferedImage
parseAndMargin(image,left,top,right,bottom)
{
    def parsedLeft =
parseValue(left,image.width,true,"0px");
    def parsedTop =
parseValue(top,image.height,true,parsedLeft)
;
    def parsedRight =
parseValue(right,image.width,true,parsedLeft
);
    def parsedBottom =
parseValue(bottom,image.height,true,parsedTo
p);
    return
margin(image,parsedLeft,parsedTop,parsedRigh
t,parsedBottom);
}
```

```
    BufferedImage
margin(image,left,top,right,bottom) {
    def width = left + image.width +
right;
    def height = top + image.height +
```

```

bottom;
    def newImage = new
BufferedImage(width.intValue(),
height.intValue(),BufferedImage.TYPE_INT_ARG
B);
    // createGraphics() needs a display,
find workaround.
    def graph =
newImage.createGraphics();
    graph.drawImage(image,new
AffineTransform(1.0d,0.0d,0.0d,1.0d,left,top
),null);
    return newImage;
}

```

```

    BufferedImage
parseAndFit(image,width,height) {
    def parsedWidth =
parseValue(width,image.width,true,"100%");
    def parsedHeight =
parseValue(height,image.height,true,parsedWi
dth);

```

```

    def imageRatio = image.width /
image.height;
    def fitRatio = parsedWidth /
parsedHeight;

```

```

    if(fitRatio < imageRatio) {
        parsedHeight = image.height *
(parsedWidth/image.width);

```

```
        } else {
            parsedWidth = image.width *
(parsedHeight/image.height);
        }

        return
parseAndScale(image,parsedWidth+"px",parsedH
eight+"px");
    }
}
```

```
BufferedImage manipulate(image) {
    for(operation in operations) {
        image =
operation[0].call([image] + operation[1]);
    }
    return image;
}
```

```
void batch() {
    def images = getImages();
    for(imageMap in images) {
        imageMap.image =
manipulate(imageMap.image);
        storeImage(imageMap);
    }
}
```

```
Object getImages() {
    def imageMaps = [];
```

```

        for(i in srcDir.listFiles()) {
            if(!i.isDirectory()) {
                def subpath = i.path;

if(subpath.startsWith(srcDir.path)) {
                    subpath =
subpath.substring(srcDir.path.length());
                }
                def matcher = subpath =~
pattern;

                    if(matcher.find()) {

imageMaps.add(["file":i, "matcher":matcher]);
                        }
                    }
                }
            imageMaps.each({it["image"] =
ImageIO.read(it["file"]); });
            return imageMaps;
        }

```

```

void storeImage(imageMap) {
    def groupIndex = 0;
    def name = outputPattern;
    def matcher = imageMap.matcher;
    while(groupIndex <=
matcher.groupCount()) {
        name =
name.replaceAll("\\${groupIndex}\\}", matche
r.group(groupIndex++));

```

```

    }
    def type =
name.substring(name.lastIndexOf(".") + 1, name.
length());
    def file = new File(srcDir, name);
    file.mkdirs();

ImageIO.write(imageMap.image, type, file);
}

static void main(args) {
    def argList = args.toList();
    def script = '';
    def groovyImage = new GroovyImage();

    // command line parsing bit, NOTE:
-h does System.exit(2)
    def argAndClosure =
['-d': {groovyImage.srcDir = new File(it)},

'-q': {groovyImage.outputPattern = it},

'-p': {groovyImage.pattern = it},

'-h': {groovyImage.help()}}];
    // parse non-conditional arguments

parseMultipleCommandArgs(argList, argAndClosu
re);

// expression, file, nothing

```

```

        if(!parseCommandArg(argList, '-e',
{script = it})) {

parseCommandArg(argList, '-f', {script = new
File(it).text});
    }

    // execution bit
    def commands = script =~
"([a-z]{1,}\\[^\]\*\[^\])";
    while(commands.find()) {

groovyImage.addOperation(commands.group(1));
    }
    groovyImage.batch();

}

static boolean
parseCommandArg(args, arg, closure) {
    def index = args.indexOf(arg);

    if(index != -1 && index + 1 <
args.size()) {
        closure.call(args[index + 1]);
        return true;
    } else {
        return false;
    }
}
}

```

```

    static void
parseMultipleCommandArgs (args, argAndClosureM
ap) {
    for(argAndClosure in
argAndClosureMap) {

parseCommandArg (args, argAndClosure.key, argAn
dClosure.value);
    }
}

void help() {
    println('usage: groovy image.groovy
-i <inputDir> -o <outputDir> -e
"<expressions>" ');
    System.exit(2);
}

/**
 * absolute true -> returns pixels.
 *           false -> returns relative
decimal (e.g 1.0).
 */
Number
parseValue(value, size, absolute, defaultValue=
"0") {
    def pattern =
"(-?[0-9]+\.\.?[0-9]*) (.*)";
    def matcher = value =~ pattern;
    if(!matcher.find()) {
        matcher = defaultValue =~

```

```
pattern;
    matcher.find();
}

def decimalValue =
Double.parseDouble(matcher.group(1));
def type = matcher.group(2);

if(absolute) { // pixels
    switch(type) {
        case "%":
            return (int) size *
(decimalValue / 100);
        case "px":
        default:
            return (int) decimalValue;
    }
}
else { // scale
    switch(type) {
        case "px":
            return decimalValue /
size;
        case "%":
            return decimalValue /
100;
        default:
            return decimalValue;
    }
}
}
```

```
Object argsLength(args,length) {
    if(args.size() < length) {
        while(args.size() < length) {
            args.add("");
        }
    } else {
        args = args.subList(0,length);
    }
}
```

```
        return args;
    }
}
```

## Calculating a SHA1 hash for large files

### Description

There are a number of posts on the net about SHA1 hashing in groovy but they tend to assume small data sets. This script calculates a SHA1 hash for an arbitrary size argument file using a one MB memory buffer.

Note that the script uses the `File.eachByte(int, closure)` method which is only available in Groovy v1.7.4 and later. You can naturally replace this construct with some other pattern, but as it fits so beautifully for the purpose, the script makes use of it.

### Code

Paste the below into a file "sha1.groovy" and run with a file param, i.e.:

### Calling the script

```
> groovy sha1.groovy myLargeFile.iso
```

and now for the code:

### Script Code

```
import java.security.MessageDigest

int KB = 1024
int MB = 1024*KB

File f = new File(args[0])
if (!f.exists() || !f.isFile()) {
    println "Invalid file $f provided"
    println "Usage: groovy sha1.groovy
<file_to_hash>"
}

def messageDigest =
MessageDigest.getInstance("SHA1")

long start = System.currentTimeMillis()

f.eachByte(MB) { byte[] buf, int bytesRead
->
    messageDigest.update(buf, 0, bytesRead);
}

def sha1Hex = new BigInteger(1,
messageDigest.digest()).toString(16).padLeft
( 40, '0' )
long delta =
System.currentTimeMillis()-start

println "$sha1Hex took $delta ms to
calculate"
```

Author: Matias Bjarland, Iteego Inc

## command line groovy doc or methods lookup

The attached script can lookup both Java and Groovy's meta methods and quickly display the method signatures.

### Usage and Examples

Help

```
$ groovy gdoc  
usage gdoc className -- display all public  
methods signatures.  
usage gdoc package.className -- same as  
above with full package prefix.  
usage gdoc className methodName -- display  
only methods that match this name.
```

All public methods

```
groovy gdoc File  
equals  
    public boolean  
java.lang.Object.equals(java.lang.Object)  
    public boolean  
java.io.File.equals(java.lang.Object)  
getClass  
    public final native java.lang.Class  
java.lang.Object.getClass()  
hashCode  
    public native int  
java.lang.Object.hashCode()  
    public int java.io.File.hashCode()  
notify
```

```
public final native void
java.lang.Object.notify()
notifyAll
public final native void
java.lang.Object.notifyAll()
toString
public java.lang.String
java.lang.Object.toString()
public java.lang.String
java.io.File.toString()
wait
public final void java.lang.Object.wait()
throws java.lang.InterruptedException
public final native void
java.lang.Object.wait(long) throws
java.lang.InterruptedException
public final void
java.lang.Object.wait(long,int) throws
java.lang.InterruptedException
canRead
public boolean java.io.File.canRead()
canWrite
public boolean java.io.File.canWrite()
compareTo
public int
java.io.File.compareTo(java.io.File)
public volatile int
java.io.File.compareTo(java.lang.Object)
createNewFile
public boolean
java.io.File.createNewFile() throws
```

```
java.io.IOException
createTempFile
    public static java.io.File
java.io.File.createTempFile(java.lang.String
,java.lang.String) throws
java.io.IOException
    public static java.io.File
java.io.File.createTempFile(java.lang.String
,java.lang.String,java.io.File) throws
java.io.IOException
delete
    public boolean java.io.File.delete()
deleteOnExit
    public void java.io.File.deleteOnExit()
exists
    public boolean java.io.File.exists()
getAbsolutePath
    public java.io.File
java.io.File.getAbsolutePath()
getAbsolutePath
    public java.lang.String
java.io.File.getAbsolutePath()
getCanonicalFile
    public java.io.File
java.io.File.getCanonicalFile() throws
java.io.IOException
getCanonicalPath
    public java.lang.String
java.io.File.getCanonicalPath() throws
java.io.IOException
getName
```

```
public java.lang.String  
java.io.File.getName()  
getParent
```

```
public java.lang.String  
java.io.File.getParent()  
getParentFile
```

```
public java.io.File
java.io.File.getParentFile()
... many more...
```

Filter method names(ones from JDK built-in)

```
$ groovy gdoc File list
list
public java.lang.String[]
java.io.File.list()
public java.lang.String[]
java.io.File.list(java.io FilenameFilter)
listFiles
public java.io.File[]
java.io.File.listFiles()
public java.io.File[]
java.io.File.listFiles(java.io FileFilter)
public java.io.File[]
java.io.File.listFiles(java.io FilenameFilter)
listRoots
public static java.io.File[]
java.io.File.listFiles()
```

Filter method names(ones from Groovy's Dynamic methods)

```
$ groovy gdoc File each
eachFile
public static void
org.codehaus.groovy.runtime.DefaultGroovyMethods.eachFile(java.io.File, groovy.lang.Closure)
```

re) throws

`java.io.FileNotFoundException, java.lang.IllegalArgumentExcep`  
`tion`

`each`

`public static java.lang.Object`  
`org.codehaus.groovy.runtime.DefaultGroovyMet`  
`hods.each(java.lang.Object, groovy.lang.Closu`  
`re)`

`eachFileMatch`

`public static void`  
`org.codehaus.groovy.runtime.DefaultGroovyMet`  
`hods.eachFileMatch(java.io.File, java.lang.Ob`  
`ject, groovy.lang.Closure) throws`  
`java.io.FileNotFoundException, java.lang.Ille`  
`galArgumentExcep`  
`tion`

`eachLine`

`public static void`  
`org.codehaus.groovy.runtime.DefaultGroovyMet`  
`hods.eachLine(java.io.File, groovy.lang.Closu`  
`re) throws java.io.IOException`

`eachObject`

`public static void`  
`org.codehaus.groovy.runtime.DefaultGroovyMet`  
`hods.eachObject(java.io.File, groovy.lang.Clo`  
`sure) throws`  
`java.io.IOException, java.lang.ClassNotFoun`  
`dE`  
`xception`

`eachDirRecurse`

`public static void`  
`org.codehaus.groovy.runtime.DefaultGroovyMet`  
`hods.eachDirRecurse(java.io.File, groovy.lang`

```
.Closure) throws
java.io.FileNotFoundException,java.lang.IllegalArgumentException
eachFileRecurse
    public static void
org.codehaus.groovy.runtime.DefaultGroovyMethods.eachFileRecurse(java.io.File,groovy.lang.Closure) throws
java.io.FileNotFoundException,java.lang.IllegalArgumentException
eachDir
    public static void
org.codehaus.groovy.runtime.DefaultGroovyMethods.eachDir(java.io.File,groovy.lang.Closure) throws
java.io.FileNotFoundException,java.lang.IllegalArgumentException
eachByte
    public static void
org.codehaus.groovy.runtime.DefaultGroovyMethods.eachByte(java.io.File,groovy.lang.Closure) throws java.io.IOException
eachDirMatch
    public static void
org.codehaus.groovy.runtime.DefaultGroovyMethods.eachDirMatch(java.io.File,java.lang.Object,groovy.lang.Closure) throws
java.io.FileNotFoundException,java.lang.IllegalArgumentException
eachWithIndex
    public static java.lang.Object
```

```
org.codehaus.groovy.runtime.DefaultGroovyMethods.  
eachWithIndex(java.lang.Object, groovy.lang.Closure)
```

Using package prefix(any things that's outside of Groovy's default imports.)

```
groovy gdoc groovy.sql.Sql each  
eachRow
```

```
    public void
```

```
groovy.sql.Sql.eachRow(groovy.lang.GString, groovy.lang.Closure) throws
```

```
java.sql.SQLException
```

```
    public void
```

```
groovy.sql.Sql.eachRow(java.lang.String, groovy.lang.Closure) throws
```

```
java.sql.SQLException
```

```
    public void
```

```
groovy.sql.Sql.eachRow(java.lang.String, groovy.lang.Closure, groovy.lang.Closure) throws
```

```
java.sql.SQLException
```

```
    public void
```

```
groovy.sql.Sql.eachRow(java.lang.String, java.util.List, groovy.lang.Closure) throws
```

```
java.sql.SQLException
```

```
each
```

```
    public static java.lang.Object
```

```
org.codehaus.groovy.runtime.DefaultGroovyMethods.each(java.lang.Object, groovy.lang.Closure)
```

```
eachWithIndex
```

```
    public static java.lang.Object
```

```
org.codehaus.groovy.runtime.DefaultGroovyMethods.eachWithIndex(java.lang.Object, groovy.lang.Closure)
```

## Compute distance from Google Earth Path (in .kml file)

Every Google Earth user is going mad about path and distance: you can get Path or Distance but not both. This little snippet help you to get path AND distance.

1. Create your path as usual with Google Earth
2. Save it as a .kml file (not .kmz)
3. Run the this script

The current version is **KML 2.2**

```
/**
 * Compute distance in Google Earth KML path
 * file from a path
 * @author Marc DEXET ( marcdexet [at] gmail
 * [dot] org )
 */
class Point {
    def lat
    def lon
    public Point(){}

    public Point(String gps) {
        def xyz = gps.tokenize(',');
        lat = Double.parseDouble( xyz[1] )
        lon = Double.parseDouble( xyz[0] )
    }
    public String toString() {
        return "LAT: ${lat} LON: ${lon}"
    }

    public static double distance(Point p0,
Point p1) {
        return Haversine.compute(p0, p1)
    }
}
```

```

}

/**
 * List of Points
 */
class PointList {
  def points
  def distance
  def partiels = []

  public PointList( List points ) {
    this.points = points
    compute()
  }

  void compute() {
    def nbPointList = points.size()
    distance = 0;
    partiels = []
    for( idx in 1..(nbPointList-1) ) {
      def p0 = points[(idx-1)]
      def p1 = points[idx]
      def dist = Point.distance(p0,p1)
      partiels << dist
      distance = distance+dist
    }
  }
}

/**
 * Haversine algorithmus

```

```

* (thanks to
http://www.movable-type.co.uk/scripts/latlong.html)
*/
class Haversine {
    static double R = 6371

    static double compute(Point p1, Point
p2) {
        def dLat =
Math.toRadians(p2.lat-p1.lat);
        def dLon =
Math.toRadians(p2.lon-p1.lon);
        def a = Math.sin(dLat/2) *
Math.sin(dLat/2) +
                Math.cos(
Math.toRadians(p1.lat) ) *
                Math.cos(
Math.toRadians(p2.lat) ) * Math.sin(dLon/2)
* Math.sin(dLon/2);
        def c = 2 * Math.atan2(Math.sqrt(a),
Math.sqrt(1-a));
        def d = R * c;
        return d
    }
}

class KmlParser {
    static ns = new
groovy.xml.Namespace("http://earth.google.co

```

```

m/kml/2.2", 'ns')
    List parse( Object input ) {
        def kml = new
groovy.util.XmlParser().parse( input );
        def coords = kml[ ns.Document ][
ns.Placemark ][ ns.LineString ][
ns.coordinates ].text()
        def myList = coords.tokenize();
        def points = []
        myList.each{ gestring -> points
<< new Point(gestring) }
        return points;
    }
}

```

```
// Application
```

```

def kmlParser = new KmlParser()
def points = kmlParser.parse( args[0] )
def PointList pointList = new PointList(
points )
def partiels = pointList.partiels;
def distance = pointList.distance;

java.text.DecimalFormat f = new
java.text.DecimalFormat( "0.000" );
java.text.DecimalFormat n = new
java.text.DecimalFormat( "00" );
println "Distance totale:
${f.format(distance)} km"
partiels.eachWithIndex { d, i ->

```

```
println "${n.format(i)}) ${f.format(d)}  
km"  
}
```

Archives

KML 2.1

```
/**  
 * Compute distance in Google Earth KML path  
file from a path  
 * @author Marc DEXET ( marcdexet [at] gmail  
[dot] org )  
 */  
class Point {  
    def lat  
    def lon  
    public Point(){}  
  
    public Point(String gps) {  
        def xyz = gps.tokenize(',');  
        lat = Double.parseDouble( xyz[1] )  
        lon = Double.parseDouble( xyz[0] )  
    }  
    public String toString() {  
        return "LAT: ${lat} LON: ${lon}"  
    }  
  
    public static double distance(Point p0,  
Point p1) {  
        return Haversine.compute(p0, p1)
```

```

    }
}

/**
 * List of Points
 */
class PointList {
    def points
    def distance
    def partiels = []

    public PointList( List points ) {
        this.points = points
        compute()
    }

    void compute() {
        def nbPointList = points.size()
        distance = 0;
        partiels = []
        for( idx in 1..(nbPointList-1) ) {
            def p0 = points[(idx-1)]
            def p1 = points[idx]
            def dist = Point.distance(p0,p1)
            partiels << dist
            distance = distance+dist
        }
    }
}

/**

```

```

* Haversine algorithmus
* (thanks to
http://www.movable-type.co.uk/scripts/latlong.html)
*/
class Haversine {
    static double R = 6371

    static double compute(Point p1, Point
p2) {
        def dLat =
Math.toRadians(p2.lat-p1.lat);
        def dLon =
Math.toRadians(p2.lon-p1.lon);
        def a = Math.sin(dLat/2) *
Math.sin(dLat/2) +
                Math.cos(
Math.toRadians(p1.lat) ) *
                Math.cos(
Math.toRadians(p2.lat) ) * Math.sin(dLon/2)
* Math.sin(dLon/2);
        def c = 2 * Math.atan2(Math.sqrt(a),
Math.sqrt(1-a));
        def d = R * c;
        return d
    }
}

class KmlParser {
    static ns = new

```

```
groovy.xml.Namespace("http://earth.google.com/kml/2.1", 'ns')
```

```
    List parse( Object input ) {
        def kml = new
groovy.util.XmlParser().parse( input );
        def coords = kml[ ns.Document ][
ns.Placemark ][ ns.LineString ][
ns.coordinates ].value[0]
        def myList = coords.tokenize();
        def points = []
        myList.each{ gestring -> points
<< new Point(gestring) }
        return points;
    }
}
```

```
// Application
```

```
def kmlParser = new KmlParser()
def points = kmlParser.parse( args[0] )
def PointList pointList = new PointList(
points )
def partiels = pointList.partiels;
def distance = pointList.distance;
```

```
java.text.DecimalFormat f = new
java.text.DecimalFormat( "0.000" );
java.text.DecimalFormat n = new
java.text.DecimalFormat( "00" );
println "Distance totale:
```

```
${f.format(distance)} km"
```

```
partiels.eachWithIndex { d, i ->
```

```
println "${n.format(i)}) ${f.format(d)}  
km"  
}
```

## Concurrency with Groovy

### *Using threads and AtomicInteger*

From Groovy you can use all of the normal concurrency facilities in Java and combine them with threads and closures as necessary. E.g. a (slightly modified) atomic counter from a Groovy example [here](#):

```
import
java.util.concurrent.atomic.AtomicInteger

def counter = new AtomicInteger()

synchronized out(message) {
    println(message)
}

def th = Thread.start {
    for( i in 1..8 ) {
        sleep 30
        out "thread loop $i"
        counter.incrementAndGet()
    }
}

for( j in 1..4 ) {
    sleep 50
    out "main loop $j"
    counter.incrementAndGet()
}

th.join()

assert counter.get() == 12
```

The output will be something like this:

```
thread loop 1
main loop 1
thread loop 2
thread loop 3
main loop 2
thread loop 4
thread loop 5
main loop 3
thread loop 6
main loop 4
thread loop 7
thread loop 8
```

#### *Fibonacci with Executors*

A Groovy script to naively calculate the Fibonacci series inspired by the example [here](#).

Note: a version using memoizing will be much more efficient but this illustrates using Java's built-in concurrency primitives from Groovy.

```

import java.util.concurrent.*

CUTOFF = 12      // not worth parallelizing
for small n
THREADS = 100

println "Calculating Fibonacci sequence in
parallel..."
serialFib = { n -> (n < 2) ? n :
serialFib(n-1) + serialFib(n-2) }
pool = Executors.newFixedThreadPool(THREADS)
defer = { c -> pool.submit(c as Callable) }
fib = { n ->
  if (n < CUTOFF) return serialFib(n)
  def left = defer{ fib(n-1) }
  def right = defer{ fib(n-2) }
  left.get() + right.get()
}

(8..16).each{ n -> println "n=$n =>
${fib(n)}" }
pool.shutdown()

```

Which produces this:

**Calculating Fibonacci sequence in parallel...**

**n=8 => 21**

**n=9 => 34**

**n=10 => 55**

**n=11 => 89**

**n=12 => 144**

**n=13 => 233**

**n=14 => 377**

**n=15 => 610**

**n=16 => 987**

If you want to convince yourself that some threads are actually being created, replace the last each line with:

```
showThreads = { println  
Thread.allStackTraces.keySet().join('\n') }  
(8..16).each{ n -> if (n == 14)  
showThreads(); println "n=$n => ${fib(n)}" }
```

which will add something like the following to your output:

```
Thread[Finalizer,8,system]
Thread[Reference Handler,10,system]
Thread[TimerQueue,5,system]
Thread[pool-1-thread-2,5,main]
Thread[Thread-3,6,main]
Thread[Thread-2,6,main]
Thread[pool-1-thread-3,5,main]
Thread[AWT-Shutdown,5,main]
Thread[Signal Dispatcher,9,system]
Thread[Attach Listener,5,system]
Thread[AWT-EventQueue-0,6,main]
Thread[DestroyJavaVM,5,main]
Thread[pool-1-thread-1,5,main]
Thread[Java2D Disposer,10,system]
Thread[pool-1-thread-5,5,main]
Thread[pool-1-thread-6,5,main]
Thread[pool-1-thread-4,5,main]
Thread[AWT-Windows,6,main]
```

#### *Fibonacci with Functional Java*

If you wish to make use of the [functionaljava](#) (tested with 2.1.3) library (see the link to the original example), you could use a Groovy program such as this:

```
import fj.*
import fj.control.parallel.Strategy
import static fj.Function.curry as fcurry
import static fj.P1.curry as pcurry
import static fj.P1.fmap
import static
fj.control.parallel.Actor.actor
```

```
import static fj.control.parallel.Promise.*
import static fj.data.List.range
import static
java.util.concurrent.Executors.*
```

```
CUTOFF  = 12    // not worth parallelizing
for small n
START    = 8
END      = 16
THREADS  = 4
```

```
pool = newFixedThreadPool(THREADS)
su    = Strategy.executorStrategy(pool)
spi   = Strategy.executorStrategy(pool)
add   = fcurry({ a, b -> a + b } as F2)
nums  = range(START, END + 1)
```

```
println "Calculating Fibonacci sequence in
parallel..."
```

```
serialFib = { n -> n < 2 ? n : serialFib(n -
1) + serialFib(n - 2) }
```

```
print = { results ->
  def n = START
  results.each{ println "n=${n++} => $it" }
  pool.shutdown()
} as Effect
```

```
calc = { n ->
  n < CUTOFF ?
```

```
    promise(su, P.p(serialFib(n))) :  
      calc.f(n - 1).bind(join(su,  
pcurry(calc).f(n - 2)), add)  
  } as F
```

```
out = actor(su, print)
```

```
join(su,  
fmap(sequence(su)).f(spi.parMapList(calc).f(  
nums))).to(out)
```

#### *Using java.util.concurrent.Exchanger*

An example using `Exchanger`. We have two threads - one keeping evens and exchanging odd values with the other thread. Meanwhile the other thread is working in reverse fashion; keeping the odds and exchanging the evens. The algorithm here is dumb in that it relies on the same number of swaps for each side - which is fine for this example but would need to be altered for more general input values.

```
def x = new java.util.concurrent.Exchanger()  
def first=1..20, second=21..40, evens, odds  
def t1 = Thread.start{ odds = first.collect{  
it % 2 != 0 ? it : x.exchange(it) } }  
def t2 = Thread.start{ evens =  
second.collect{ it % 2 == 0 ? it :  
x.exchange(it) } }  
[t1, t2]*.join()  
println "evens: $evens"  
println "odds: $odds"
```

Which has the following output:

```
evens: [2, 22, 4, 24, 6, 26, 8, 28, 10, 30,  
12, 32, 14, 34, 16, 36, 18, 38, 20, 40]  
odds: [1, 21, 3, 23, 5, 25, 7, 27, 9, 29,  
11, 31, 13, 33, 15, 35, 17, 37, 19, 39]
```

#### *Catching Exceptions with an Exception Handler*

When catching Exceptions with an exception handler in Groovy Scripts, there is potential for interaction with Groovy's runtime which catches exceptions and filters stacktraces. The best way to avoid this interaction is to create your own thread and call `setDefaultUncaughtExceptionHandler` directly on that thread instance as per below:

```
def th = Thread.start {
    println 'start'
    println Thread.currentThread()
    sleep 1000
    throw new NullPointerException()
}
th.setDefaultUncaughtExceptionHandler({t,ex
->
    println 'ignoring: ' + ex.class.name
} as Thread.UncaughtExceptionHandler)
th.join()
```

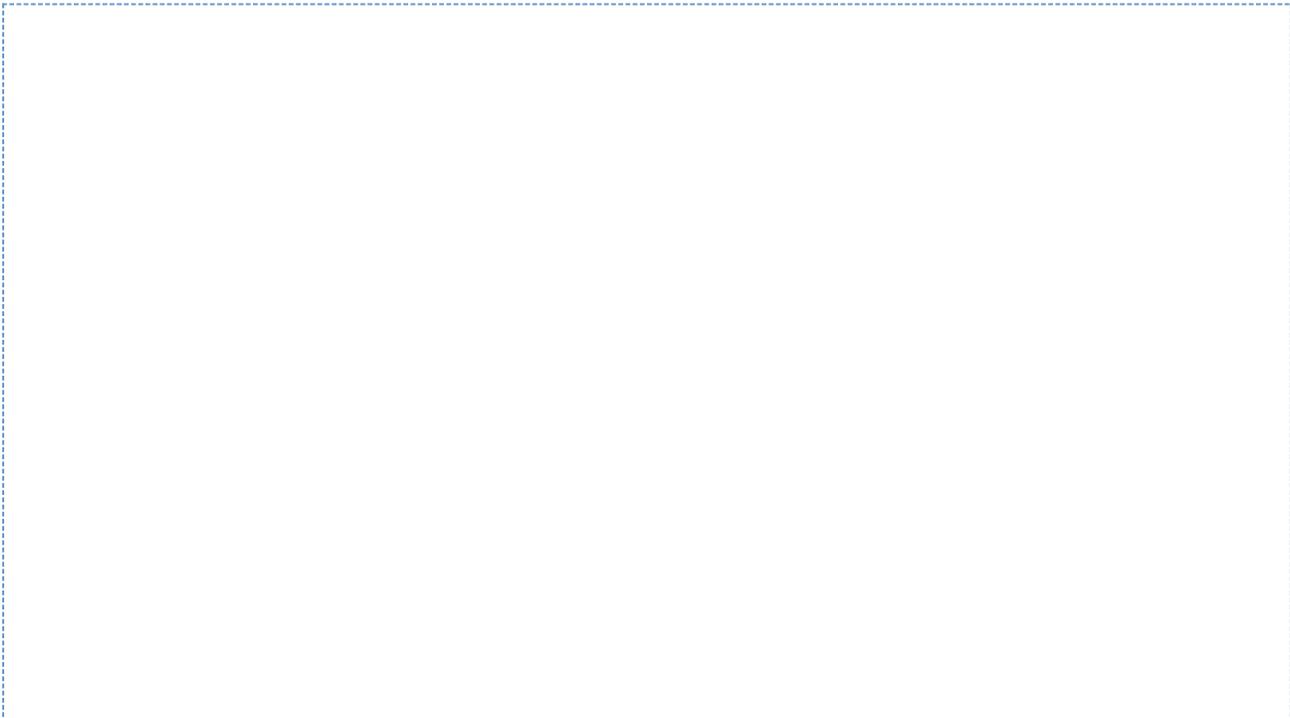
#### *More Information*

See also:

1. [Functional Programming with Groovy](#)
2. [Multithreading with SwingBuilder](#)

### **Convert SQL Result To XML**

How to convert SQL Result to XML ?



```

import groovy.sql.Sql
import groovy.xml.MarkupBuilder
def schema = "PROD"
def sql =
Sql.newInstance("jdbc:oracle:thin:@hostname:
1526:${schema}", "scott", "tiger",
"oracle.jdbc.driver.OracleDriver")

/* Request */
def req = ""
SELECT id, name, givenname, unit FROM
${schema}.people
WHERE
in_unit=1
AND visible=0
""

def out = new File('out.xml')
def writer = new FileWriter( out )
def xml = new MarkupBuilder( writer )

xml.agents {
    sql.eachRow( req as String ) {
        /* For each row output detail */
        row ->
            xml.agent(id:row.id) {
                name( row.name )
                givenname( row.givenname )
                unit( row.unit )
            }
    }
}

```

Output is

```
<agents>                                <!--
xml.agents {                               -->
  <agent id='870872'>                       <!--
agent(id:row.id) {                           -->
  <name>ABTI</name>                          <!--
name( row.nom )                             -->
  <givenname>Jean</givenname>               <!--
givenname( row.prenom ) -->
  <unit>Sales</unit>                        <!--
unit( row.unite )                             -->
  </agent>                                   <!--
}                                             -->
...
</agents>
```

## Embedded Derby DB examples

Be sure that derby.jar is either:

- in the groovy lib folder
- explicitly set in the classpath (ie. set CLASSPATH=%CLASSPATH%;/path/to/derby.jar)
- loaded at runtime (ie. this.class.classLoader.rootLoader.addURL(new URL("file:derby.jar")) )

If you are using JDK 1.5 or greater, a LinkageError will occur. Removing the mx4j-3.0.2.jar library from %GROOVY\_HOME%\lib will make it work again, but a better solution would be preferable.

Look at <http://jira.codehaus.org/browse/GROOVY-2303> for further information, or ask on the user@groovy.codehaus.org.

This will create derby.log and derbyDB folder in the folder where the script is.

If something goes wrong, simply remove derbyDB folder, and everything will be clean for next run.

```
//uncomment the next line to load the derby
```

```
jar at runtime :
//set
CLASSPATH=%CLASSPATH%;d:\\path\\to\\derby.jar

import groovy.sql.*
import java.sql.*

protocol = 'jdbc:derby:'
def props = new Properties()
props.put('user', 'user1')
props.put('password', 'user1')

def sql =
Sql.newInstance("${protocol}derbyDB;create=true", props)

/* Creating table, adding few lines,
updating one */
sql.execute('create table people(id int,
name varchar(40), second_name varchar(40),
phone varchar(30), email varchar(50))')

println("Created table 'people'")

sql.execute("insert into people values
(1, 'John', 'Doe',
'123456', 'johndoe@company.com')")
sql.execute("insert into people values
(2, 'Bill', 'Brown',
'324235', 'billbrown@company.com')")
```

```
sql.execute("insert into people values
(3, 'Jack', 'Daniels',
'443323', 'jackdaniels@company.com')")

println('Inserted people')

sql.execute("update people set
phone='443322', second_name='Daniel''s'where
id=3")

println('Updated person')

/* Simple query */
def rows = sql.rows('SELECT * FROM people
ORDER BY id')
rows.each {println it}

/* Dropping table 'people' */
sql.execute('drop table people')
println ("Table 'people' dropped")

try{
DriverManager.getConnection('jdbc:derby;;shu
tdown=true')
}
catch (SQLException se){
    gotSQLExc = true
}

println('Finish!')
```

## Embedding a Groovy Console in a Java Server Application

The following example of using Groovy to execute arbitrary code in a running web application is excerpted from a [longer article](#) originally posted to [Bruce Fancher's blog](#). The code and examples can be downloaded from [here](#).

In order to add the GroovyServer to the application, I added the following lines to the Spring applicationContext.xml file:

### appContext.xml

```
<bean id="shirtService"
class="org.pimpmyshirt.service.ShirtServiceI
mpl" />

<bean id="contextWrapper"
class="org.pimpmyshirt.spring.ApplicationCon
textWrapper" />

<bean id="groovyService" abstract="true"
init-method="initialize"
destroy-method="destroy">
  <property name="bindings">
    <map>
      <entry key="context"
value-ref="contextWrapper" />
      <entry key="shirtService"
value-ref="shirtService" />
    </map>
  </property>
</bean>

<bean id="groovyShellService"
class="com.iterative.groovy.service.GroovySh
ellService" parent="groovyService">
```

```
<property name="socket" value="6789" />  
  <property name="launchAtStart"  
value="true" />  
</bean>
```

```
<bean id="groovyConsoleService"  
class="com.iterative.groovy.service.GroovyCo  
nsoleService" parent="groovyService" />
```

You'll note that there are actually two Groovy-related services. The first one, `groovyShellService`, is the networked wrapper around the `InteractiveShell`. The second one, `groovyConsoleService`, is a wrapper around the `GroovyConsole`, which is a Swing-based application that provides essentially the same facility as the `InteractiveShell`, but in an application with a nice GUI. Since only the `GroovyShellService` allows remote access to an application it is focus of this article. But if you're running the server application on the same machine you're developing on, you can hit the URL <http://localhost:8080/pimpmyshirt/launchGroovyConsole.html>, which will trigger a simple Spring web controller to launch an instance of the `GroovyConsole`. Just note that for some reason exiting the `GroovyConsole` will cause Tomcat to exit, but since I mostly use the `GroovyShellService`, and this is only intended for development and testing purposes, I haven't bothered to try and find out why this is.

Both services inherit from the `groovyService` abstract bean, which includes bindings for the `shirtService`, which is a service included in the `PimpMyShirt` application that we'll explore with Groovy, and an instance of `ApplicationContextWrapper`, which is a class that implement's Springs `ApplicationContext` and `ApplicationContextAware` interfaces. The `ApplicationContextWrapper` is given a reference to the Spring application context through the `ApplicationContextAware` interface, and delegates all of `ApplicationContext`'s methods to this instance. I did this because I didn't want the `GroovyServices` to be dependent on Spring, and while there might very well be a simpler way to pass an instance of the application context to a bean without implementing `ApplicationContextAware`, I don't know what it is.

After building the application with the included ant script, a war file is produced that it should be possible to deploy to any J2EE application server (although I've only tested it with Tomcat). Once it's deployed and launched, the first thing to do is to connect to the web application at <http://hostname:8080/pimpmyshirt/index.html> and enter some ratings for the shirts, in order to have some data in the application before we test it:



Now we can connect to the `GroovyServer` and run some code to display the application's state. As configured, the application will launch the server on port 6789 when it starts, so assuming the application is running on the same machine you're sitting in front of, you can connect to it by just opening a shell and typing `telnet localhost 6789`. What you'll see is exactly what you'd get if you were to run `groovysh` on it's own:

**Let's get Groovy!**

**=====**

**Version: 1.0 JVM: 1.5.0\_07-87**

**Type 'exit' to terminate the shell**

**Type 'help' for command help**

**Type 'go' to execute the statements**

**groovy>**

We can now issue commands to the interpreter and see the results:

```

def shirts = shirtService.getAllShirts();

shirts.each() {
    def shirtRating =
shirtService.getRating(it);
    out.println "Color: ${it.color}, Long
Sleeve: ${it.longSleeve}, Graphical:
${it.print.graphical},
Text: ${it.print.text}, Low Votes:
${shirtRating.numberOfLowVotes}, Medium
Votes: ${shirtRating.numberOfMediumVotes},
High Votes:
${shirtRating.numberOfHighVotes}"
}

go

```

```

Color: WHITE, Long Sleeve: false, Graphical:
false, Text: JavaPolis, Low Votes: 1, Medium
Votes: 0, High Votes: 2
Color: BLUE, Long Sleeve: true, Graphical:
false, Text: Spring Rocks!, Low Votes: 0,
Medium Votes: 3, High Votes: 0

```

This code uses a Groovy closure to iterate over the ratings for available shirts and prints out the stats for each one. I won't explain Groovy syntax in detail as I go through these examples, since full documentation on programming in Groovy can be found on the Groovy web site. However, before going further I need to point out one difference between executing Groovy in this environment compared to its typical usage. You'll note that in the above example I've used "out.println" instead of just "println," as is usually the case in Groovy code. This is because in normal Groovy "println" writes to System.out, and in a server application, System.out is usually redirected to a log file, which is not where we usually want the output of our scripts to go. To work around this, the GroovyShellService passes in the socket's OutputStream bound to the token "out." So to print to the interactive console, we need to use "out.println" instead of just "println." Although there are other ways to work around this problem, which might be

more transparent from the point of view of a user of the shell, I've chosen to do it this way since it's the easiest to implement, and the most explicit with regards to what's actually happening under the covers.

Note that since we've configured the GroovyServer in the Spring application context to have a binding to "shirtService," it's already available to us. If we hadn't done so, we could've also gotten a reference to the service from the application context by prepending the following to the code snippet above:

```
def shirtService =  
context.getBean("shirtService");
```

We can also call methods on the Spring application context to see, for example, what beans are available:

```
context.getBeanDefinitionNames().each() {  
out.println it };  
go  
  
shirtService  
contextWrapper  
groovyService  
groovyShellService  
groovyConsoleService  
  
===> null
```

And in addition to the application services that are defined in the Spring context, we can also interrogate the web and application server layers to see how they're configured. For example, we could get the ServletContext from Spring and display its attribute names and values:

```
def servletContext =  
context.getServletContext();  
  
Enumeration e =  
servletContext.getAttributeNames();  
while (e.hasMoreElements()) {
```

```
def attributeName = e.nextElement();
out.println "${attributeName}:
${servletContext.getAttribute(attributeName)
} \n";
}
}
go
```

```
org.apache.catalina.jsp_classpath:
/usr/local/apache-tomcat-5.5.23/webapps/pimp
myshirt/WEB-INF/classes/
/usr/local/apache-tomcat-5.5.23/webapps/pimp
myshirt/WEB-INF/lib/commons-collections-3.1.
jar:(etc . . .)
```

```
javax.servlet.context.tempdir:
/usr/local/apache-tomcat-5.5.23/work/Catalin
a/localhost/pimpmyshirt
```

```
org.springframework.web.servlet.FrameworkSer
vlet.CONTEXT.pimpmyshirt:
org.springframework.web.context.support.XmlW
ebApplicationContext: display name
[WebApplicationContext for
namespace 'pimpmyshirt-servlet']; startup
date [Mon Apr 30 01:52:03 EDT 2007]; child
of
[org.springframework.web.context.support.Xml
WebApplicationContext: display name [Root
WebApplicationContext];
startup date [Mon Apr 30 01:52:02 EDT 2007];
root of context hierarchy; config locations
```

```
[classpath:org/pimpmyshirt/service/applicati  
onContext.xml]]; config locations  
[/WEB-INF/pimpmyshirt-servlet.xml]
```

```
interface
```

```
org.springframework.web.context.WebApplicati  
onContext.ROOT:
```

```
org.springframework.web.context.support.XmlW  
ebApplicationContext: display name [Root  
WebApplicationContext];
```

```
startup date [Mon Apr 30 01:52:02 EDT 2007];
```

```
root of context hierarchy; config locations
```

```
[classpath:org/pimpmyshirt/service/applicati  
onContext.xml]
```

```
org.apache.catalina.resources:
```

```
org.apache.naming.resources.ProxyDirContext@  
fc1695
```

```
org.apache.catalina.WELCOME_FILES:
```

```
{"index.html", "index.htm", "index.jsp"}
```

```
==> null
```

We can see from this output that Spring's `WebApplicationContext`, in which the servlets that Spring uses to provide hooks into its web framework are defined is bound to the `ServletContext`'s `"org.springframework.web.servlet.FrameworkServlet.CONTEXT.pimpmyshirt"` attribute. If we wanted a list of which beans were configured in the `WebApplicationContext`, we could print them out by doing the following:

```
def servletContext =  
context.getServletContext();  
def servletAppContext =  
servletContext.getAttribute("org.springframework.  
web.servlet.FrameworkServlet.CONTEXT.pi  
mpmyshirt");  
servletAppContext.getBeanDefinitionNames().e  
ach() { out.println it };  
go  
  
viewResolver  
messageSource  
multipartResolver  
/index.html  
/image.html  
flowController  
composeShirt  
composeShirtAction  
/launchGroovyConsole.html
```

We could also explore further, and get an instance of the `RateShirts` view from Spring's `ViewResolver`:

```
def servletContext =
context.getServletContext()
def servletAppContext =
servletContext.getAttribute("org.springframework
work.web.servlet.FrameworkServlet.CONTEXT.pi
mpmyshirt");
def viewResolver =
servletAppContext.getBean("viewResolver");
def view =
viewResolver.buildView("RateShirts");
view
go

===>
org.springframework.web.servlet.view.JstlVie
w: name 'RateShirts'; URL
[/WEB-INF/jsp/RateShirts.jsp]
```

Or we could get an instance of the RateShirts web controller and view the contents of it's model data after it's initialized:

```
def servletContext =
context.getServletContext();
def servletAppContext =
servletContext.getAttribute("org.springframework
work.web.servlet.FrameworkServlet.CONTEXT.pimpmyshirt");
def rateShirtsController =
servletAppContext.getBean("/index.html");
rateShirtsController.getModel()
go

===>
{shirtRatings=[org.pimpmyshirt.domain.ShirtRating@6ebc80,
org.pimpmyshirt.domain.ShirtRating@198a2f],
ratings=[Lorg.pimpmyshirt.domain.Rating;@88b2fa]}
```

Obviously this is a sample application with a single service that doesn't do very much, so there isn't that much more we can do with it that would be all that interesting. However, for a real application with dozens or more services that were reasonably complex, it shouldn't be hard to imagine the usefulness of being able to interact with them to test their functionality and experiment with using them.

## Executing External Processes From Groovy

**Goal:** execute a program via a command line from groovy code

**Option 1: executing a string**

```
A string can be executed in the standard
java way:
def command = ""executable arg1 arg2
arg3"" // Create the String
def proc = command.execute()
// Call *execute* on the string
proc.waitFor()
// Wait for the command to finish

// Obtain status and output
println "return code: ${proc.exitValue()}"
println "stderr: ${proc.err.text}"
println "stdout: ${proc.in.text}" // *out*
from the external program is *in* for groovy
```

Gotchas: Take care if you wish to pass a quoted argument that contains white space – it will be split into multiple arguments, e.g.:

```
""executable "first with space"
second"".execute()
```

will invoke executable with the following arguments:

- arg1 = "first
- arg2 = with
- arg3 = space"
- arg4 = second

In such a case, you may prefer to use one of the array or list of String variations, e.g.:

```
["executable", "first with space",
"second"].execute()
```

## Option 2: using ant builder's exec task

Ant has an exec task and it be accessed from the AntBuilder object

```
def ant = new AntBuilder() // create an
antbuilder
ant.exec(outputproperty:"cmdOut",
         errorproperty:"cmdErr",
         resultproperty:"cmdExit",
         failonerror:"true",
         executable:
' /opt/myExecutable' ) {

arg(line:""*"first with space"* second"")
}
println "return code:
${ant.project.properties.cmdExit}"
println "stderr:
${ant.project.properties.cmdErr}"
println "stdout:      ${
ant.project.properties.cmdOut}"
```

The good thing is that you now have all the ant features at your disposal and Ant will not break up quoted args containing whitespace.

See also: [Process Management](#)

## Expect for Groovy

To create a simple groovy version of the ["expect"](#) concept, all you need are the three Groovy classes below.

The main of IOsession shows how to use it:

### OutputSession.groovy

```
class OutputSession implements
```

```

java.io.Closeable {
    OutputStream outputStream
    String charsetName
    Thread thread
    Long slow
    final Object sendLock = new Object()
    boolean aborted
    boolean closed = false

    OutputSession(OutputStream outputStream)
{
    setOutputStream outputStream
}

    OutputSession(OutputStream outputStream,
String charsetName) {

setCharsetName(charsetName)
    setOutputStream outputStream
}

    void send(String outStr) throws
Exception {
        send outStr, 0
    }

    void send(String outStr, long timeout)
throws Exception {
        send charsetName != null ?
outStr.getBytes(charsetName) :
outStr.getBytes(), timeout

```

```

    }

    void send(byte[] outBytes, long timeout)
    {
        send outBytes, 0, outBytes.length,
timeout
    }

    void send(final byte[] outBytes, final
int outPtr, final int outLen, final long
timeout) {
        synchronized (sendLock) {
            aborted = false
            if (closed) {
                throw new
Exception("OutputSession is closed, cannot
send out string:
'$charsetName==null?new
String(outBytes,outPtr,outLen):new
String(outBytes,outPtr,outLen,charsetNa
me) ' ")
            }
            if (outputStream == null) {
                throw new
Exception("OutputStream is null, cannot send
out string: '$charsetName==null?new
String(outBytes,outPtr,outLen):new
String(outBytes,outPtr,outLen,charsetNa
me) ' ")
            }
            if (outLen > 0) {

```

```

        thread = Thread.startDaemon(
            {
                if (slow ==
null) {
outputStream.write(outBytes, outPtr, outLen)
outputStream.flush()
                outBytes =
null
                outLen = 0
            } else {
                int ptr =
outPtr
                int len =
outLen
                while (len >
0) {
outputStream.write(outBytes, ptr, 1)
outputStream.flush()
                ptr++
                len--
                if (len
!= 0 && slow != null) {
sleep slow
                }
            }
        }
    }
}

```

```

        }
    )
}
try {
    if (timeout == 0) {
        thread.join()
    } else {
        thread.join(timeout)
    }
} catch (InterruptedException
ex) {
}
if (closed) {
    throw new
IllegalStateException("OutputSession was
closed while sending string:
'$charsetName==null?new
String(outBytes,outPtr,outLen):new
String(outBytes,outPtr,outLen,charsetNa
me) ' ")
}
if (aborted) {
    return;
}
if (outBytes != null) {
    throw new Exception("Timeout
while sending string:
'$charsetName==null?new
String(outBytes,outPtr,outLen):new
String(outBytes,outPtr,outLen,charsetNa
me) ' ")

```

```

        }
    }
}

void abort() {
    aborted = true
    thread?.interrupt()
}

void close() {
    closed = true
    abort()
}
}

```

## InputSession.groovy

```

class InputSession implements
java.io.Closeable {
    static final int buffersize = 2048
    byte[] inputData = new byte[buffersize]
    Thread thread
    StringBuffer stringBuffer = new
StringBuffer()
    int caret = 0
    String charactersetName
    InputStream inputStream
    Set<OutputSession> pipes = new
HashSet<OutputSession>()
    boolean aborted
}

```

```

boolean closed = false
boolean timedOut
long expirationTime
String group

String receivedUntil
Object receivedObject
Object expectations

final Object lock = new Object()

InputSession(InputStream inputStream,
String charsetsetName) {
    setCharsetsetName charsetsetName
    setInputStream inputStream
}

InputSession(InputStream inputStream) {
    setInputStream inputStream
}

void setExpectations(Object
expectations) {
    if (expectations instanceof
Collection || expectations instanceof Map) {
        this.expectations = expectations
    } else {
        this.expectations = new
ArrayList()
        ((ArrayList)
this.expectations).add(expectations)

```

```

        }
    }

    String expect(Object expectations, long
timeout) {
        expirationTime =
System.currentTimeMillis() + timeout
        synchronized (lock) {
            if (closed) {
                throw new
IllegalStateException("Cannot expect on a
closed InputSession")
            }
            setExpectations(expectations)
            timedOut = false
            aborted = false
            receivedObject = null
            receivedUntil = null
            if (stringBuffer.length() >
caret) {
                if (check(null)) {
                    return group
                }
            }
            while (this.expectations !=
null) {
                if(isClosed()){
                    throw new
IllegalStateException("InputSession was
closed while expecting: $expectations")
                }
            }
        }
    }
}

```

```

        if (isAborted()) {
            return null
        }
        long timeToWait =
expirationTime - System.currentTimeMillis()
        if (timeToWait > 0L) {
            println "waiting
$timeToWait ms"

            lock.wait(timeToWait)
        } else {
            println "timed out"
            this.expectations=null
            timedOut = true
            return null
        }
    }
    return group
}
}

```

```

    void setInputStream(final InputStream
inputStream) {
        if (inputStream != this.inputStream)
{
            if (inputStream == null) {
                throw new
IllegalArgumentException("Cannot set
InputStream to null")
            }
            thread?.interrupt()
            this.inputStream = inputStream

```

```

        thread = Thread.startDaemon(
            {
                int total = 0
                while (total >= 0) {
                    total =
inputStream.read(inputData, 0, buffersize)
                    if (total > 0) {

add(inputData, 0, total)
                    }
                }
            }
        )
    }
}

```

```

    void add(byte[] data, int start, int
total) {
        if (total > 0) {
            if (charsetName != null) {
                add(new String(data, start,
total, charsetName))
            }
            add(new String(data, start,
total))
        }
    }
}

```

```

    void add(String string) {
        if (string != null &&
string.length() > 0) {

```

```

        System.out.println("inputStream:
" + string)
        for (OutputSession pipe: pipes)
        {
            pipe.send string
        }
        check string
    }
}

```

```

protected boolean check(String string) {
    synchronized (lock) {
        if (string != null) {
            stringBuffer.append(string)
        }
        if (expectations != null) {
            Integer pos = null
            Object received = null
            expectations.each {Object
expectation ->
                if (pos != 0) {
                    def o = expectation
instanceof Map.Entry ? ((Map.Entry)
expectation).key : expectation
                    int p = -1
                    String g = null
                    if (o instanceof
Pattern) {
                        Matcher m =
((Pattern) o).matcher(stringBuffer)
                            if

```

```

(m.find(caret)) {
                                p =
m.start()
                                g =
m.group()
                                }
                                } else {
                                g = o.toString()
                                p =
stringBuffer.indexOf(g, caret)
                                //println
"looking for '$g' in '$stringBuffer' from:
$caret index:$p"
                                }
                                if (p >= 0 && (pos
== null || p < pos)) {
                                received =
expectation
                                group = g
                                pos = p
                                }
                                }
                                if (pos != null) {
                                //println "Received:
$received"
                                receivedUntil =
stringBuffer.substring(caret, pos)
                                caret += pos +
group.length()
                                receivedObject =

```

received

```
                expectations = null
                lock.notify()
                return true
            }
        }
    }
    return false
}

void abort() {
    aborted = true
    thread?.interrupt()
}

void close() {
    synchronized (lock) {
        closed = true
        abort();
    }
}
```

```
    }  
  }  
}
```

## IOSession.groovy

```
class IOSession implements java.io.Closeable  
{  
    static final int buffersize = 2048  
    Map<Object, InputSession> inputSessions  
= new HashMap<Object, InputSession>()  
    Map<Object, OutputSession>  
outputSessions = new HashMap<Object,  
OutputSession>()  
  
    Map pipes = new HashMap()  
  
    InputSession inputSession  
    OutputSession outputSession  
    Thread outputThread  
    Thread inputThread  
    String charsetsetName  
    long timeout = 10000  
  
    boolean closed = false  
  
    final sendLock = new Object()  
  
    IOSession(Object object) {  
        add object
```

```

    }

    IOsession(Object object, String
charactersetName) {
        add object, charactersetName
    }

    void setSlow(Long slow) {
        if (outputSession == null) {
            throw new
IllegalStateException("Cannot set slow")
        }
        outputSession.setSlow slow
    }

    void add(Object object, String
charactersetName) {
        if (object == null) {
            throw new
IllegalArgumentException("Cannot add null to
Session")
        }
        if (!addOutputStream(object,
charactersetName) && !addInputStream(object,
charactersetName) && !addProcess(object,
charactersetName)) {
            throw new
IllegalArgumentException("Cannot add
'$object' to Session")
        }
    }
}

```

```

    void add(Object object) {
        if (object == null) {
            throw new
IllegalArgumentException("Cannot add null to
Session")
        }
        if (!addOutputStream(object) &&
!addInputStream(object) &&
!addProcess(object)) {
            throw new
IllegalArgumentException("Cannot add
'$object' to Session")
        }
    }

```

```

    boolean addOutputStream(Object object) {
        if (object == null) {
            return false
        }
        if (object instanceof File) {
            return addOutputStreamEntry(new
MapEntry(object, new FileOutputStream((File)
object)))
        }
        if (object instanceof String) {
            return addOutputStreamEntry(new
MapEntry(object, new
FileOutputStream((String) object)))
        }
        if (object instanceof OutputStream)

```

```

{
    return addOutputStreamEntry(new
MapEntry(object, (OutputStream) object))
    }
    if (object instanceof Map.Entry) {
        return
addOutputStreamEntry((Map.Entry) object)
    }
    return false
}

boolean addOutputStream(Object object,
String charsetsetName) {
    if (object == null) {
        return false
    }
    if (object instanceof File) {
        return addOutputStreamEntry(new
MapEntry(object, new FileOutputStream((File)
object)), charsetsetName)
    }
    if (object instanceof String) {
        return addOutputStreamEntry(new
MapEntry(object, new
FileOutputStream((String) object)),
charsetsetName)
    }
    if (object instanceof OutputStream)
{
        return addOutputStreamEntry(new
MapEntry(object, (OutputStream) object),

```

```

charactersetName)
    }
    if (object instanceof Map.Entry) {
        return
addOutputStreamEntry((Map.Entry) object,
charactersetName)
    }
    return false
}

```

```

boolean addOutputStreamEntry(Map.Entry
entry) {
    if (entry.value instanceof
OutputStream) {
        OutputSession old =
outputSessions.get(entry.key)
        if (old != null) {

old.setOutputStream((OutputStream)
entry.value)
        } else {
            old = new
OutputStream((OutputStream) entry.value)
            outputSessions.put
entry.key, old
        }
        if (outputSession == null) {
            outputSession = old
        }
        return true
    }
}

```

```

        return false
    }

    boolean addOutputStreamEntry(Map.Entry
entry, String charsetName) {
        if (entry.value instanceof
OutputStream) {
            OutputSession old =
outputSessions.get(entry.key)
            if (old != null) {
                old.setCharsetName
charsetName

old.setOutputStream((OutputStream)
entry.value)
            } else {
                old = new
OutputSession((OutputStream) entry.value,
charsetName)
                outputSessions.put
entry.key, old
            }
            if (outputSession == null) {
                outputSession = old
            }
            return true
        }
        return false
    }

    boolean addProcess(Object object) {

```

```
        boolean added = false
        if
(object.metaClass.respondsTo(object,
"getOutputStream")) {
            added |=
addOutputStream(object.getOutputStream())
        }
        if
(object.metaClass.respondsTo(object,
"getInputStream")) {
            added |=
addInputStream(object.getInputStream())
        }
        if
(object.metaClass.respondsTo(object,
"getErrorStream")) {
            added |=
addInputStream(object.getErrorStream())
        }
        return added
    }
}
```

```
    boolean addProcess(Object object, String
charactersetName) {
        boolean added = false
        if
(object.metaClass.respondsTo(object,
"getOutputStream")) {
            added |=
addOutputStream(object.getOutputStream(),
charactersetName)
        }
    }
}
```

```

        }
        if
(object.metaClass.respondsTo(object,
"getInputStream")) {
            added |=
addInputStream(object.getInputStream(),
charactersetName)
        }
        if
(object.metaClass.respondsTo(object,
"getErrorStream")) {
            added |=
addInputStream(object.getErrorStream(),
charactersetName)
        }
        return added
    }
}

```

```

boolean addInputStream(Object object) {
    if (object == null) {
        return false
    }
    if (object instanceof File) {
        return addInputStreamEntry(new
MapEntry(object, new FileInputStream((File)
object)))
    }
    if (object instanceof String) {
        return addInputStreamEntry(new
MapEntry(object, new
FileInputStream((String) object)))
    }
}

```

```

    }
    if (object instanceof InputStream) {
        return addInputStreamEntry(new
MapEntry(object, (InputStream) object))
    }
    if (object instanceof Map.Entry) {
        return
addInputStreamEntry((Map.Entry) object)
    }
    return false
}

```

```

    boolean addInputStream(Object object,
String charsetsetName) {
    if (object == null) {
        return false
    }
    if (object instanceof File) {
        return addInputStreamEntry(new
MapEntry(object, new FileInputStream((File)
object)), charsetsetName)
    }
    if (object instanceof String) {
        return addInputStreamEntry(new
MapEntry(object, new
FileInputStream((String) object)),
charsetsetName)
    }
    if (object instanceof InputStream) {
        return addInputStreamEntry(new
MapEntry(object, (InputStream) object),

```

```

charactersetName)
    }
    if (object instanceof Map.Entry) {
        return
addInputStreamEntry((Map.Entry) object,
charactersetName)
    }
    return false
}

boolean addInputStreamEntry(Map.Entry
entry) {
    if (entry.value instanceof
InputStream) {
        InputSession old =
inputSessions.get(entry.key)
        if (old != null) {

old.setInputStream((InputStream)
entry.value)
        } else {
            old = new
InputSession((InputStream) entry.value)
            inputSessions.put(entry.key,
old)
        }
        if (inputSession == null) {
            inputSession = old
        }
    }
    return true
}

```

```

        }
        return false
    }

    boolean addInputStreamEntry(Map.Entry
entry, String charsetName) {
        if (entry.value instanceof
InputStream) {
            InputSession old =
inputSessions.get(entry.key)
            if (old != null) {
                old.setCharsetName
charsetName

old.setInputStream((InputStream)
entry.value)
            } else {
                old = new
InputSession((InputStream) entry.value,
charsetName)
                inputSessions.put(entry.key,
old)
            }
            if (inputSession == null) {
                inputSession = old
            }
            return true
        }
        return false
    }
}

```

```

    void send(Closure outStr) throws
Exception {
    send(outStr.call(this).toString())
}

    void send(String str) throws Exception {
    if (outputSession == null) {
        throw new
IllegalStateException("Cannot send, because
outputSession is not set")
    } else {
        outputSession.send(str, timeout)
    }
}

    void send(byte[] outBytes) {
    if (outputSession == null) {
        throw new
IllegalStateException("Cannot send, because
outputSession is not set")
    } else {
        outputSession.send(outBytes,
timeout)
    }
}

    void send(byte[] outBytes, int outPtr,
int outLen) {
    if (outputSession == null) {
        p

```

```
        throw new
IllegalStateException("Cannot send, because
outputSession is not set")
    } else {
        outputSession.send(outBytes,
outPtr, outLen, timeout)
    }
}
```

```
void fail(String failureMessage) {
    throw new
IllegalStateException(failureMessage)
}
```

```
void fail(Closure failureMessage) {
fail(failureMessage.call(this).toString())
}
```

```
String expect(Object expectations, long
timeout, Closure onTimeout) {
    if (inputSession == null) {
        throw new
IllegalStateException("Cannot expect,
because inputSession is not set")
    }
    inputSession.expect(expectations,
timeout)
    if (inputSession.isClosed()) {
        throw new
IllegalStateException("InputSession was
```

```

closed while expecting: $expectations")
    }
    if (inputSession.isAborted()) {
        return null
    }
    if (inputSession.isTimedOut()) {
        return onTimeout?.call(this)
    }
    Object o =
inputSession.getReceivedObject()
    if (o == null) {
        throw new
IllegalStateException("InputSession ended
without receiving: $expectations")
    }
    if (o instanceof Map.Entry &&
((Map.Entry) o).value instanceof Closure) {
        return ((Closure) ((Map.Entry)
o).value).call(this)
    }
    return inputSession.getGroup()
}

```

```

String expect(Object expectations,
Closure onTimeout) {
    return expect(expectations, timeout,
onTimeout)
}

```

```

String expect(Object expectations, long

```

```

timeout) {
    return expect(expectations, timeout,
{IOSession it -> it.fail "Timeout while
expecting: $expectations"})
}

String expect(Object expectations) {
    return expect(expectations, timeout)
}

void abort() {
    outputSessions.each {OutputSession
it -> it.abort()}
    inputSessions.each {InputSession it
-> it.abort()}
}

void close() throws java.io.IOException
{
    closed = true
    outputSessions.each {OutputSession
it -> it.close()}
    inputSessions.each {InputSession it
-> it.close()}
}

public static void main(String[] args) {
    println "main..."
    ServerSocket serverSocket = new
ServerSocket(8995)
    Thread.start {

```

```

        IOsession client = new
IOsession(new Socket('127.0.0.1', 8995))
        println 'client sending GET...'
        client.send 'GET'
        client.expect 'HTML'
        println
Double.parseDouble(client.expect(~/[~+]?[0-9
]*\.[0-9]+\s/).toString().trim())
        client.send 'PUT'
        println "Client OK"
    }
    Thread.start {
        IOsession server = new
IOsession(serverSocket.accept())
        println 'server expecting...'
        println "expect on server
returned: " + server.expect([(~/G?T/):
{IOsession session -> println 'server has
received a GET request'; session.send
'HTML2.34\n'}, 'PUT': {println 'server has
received a PUT request'}}])
        println "expect on server
returned: " + server.expect([new
MapEntry('GET', {println 'server has
received a GET'}), 'PUT'])
        println "Server OK"
    }
}

```

```
    }  
  }  
}
```

## Formatting simple tabular text data

### Formatting simple tabular text data

This class has been posted first time on the Groovy-User Mailing List by Raffaele Castagno in this format:

```
class TableTemplateFactory  
{  
  def columns = [];      // contains columns  
names and theyr length  
  def header1 = '';    // contains columns  
names  
  def header2 = '';    // contains  
underscores  
  def body      = '';  // the rows of the  
table  
  def footer    = '';  // actually unused:  
can contain footer notes, totals, etc.  
  
  def addColumn(name, size)  
  {  
    columns << [name:name, size:size];  
  }  
  
  def getTemplate()  
  {  
    header1 = "\n";  
    columns.each{ header1 += ' <%print
```

```
"'+it.name+'".center('+it.size+')%> ' };
    header2 = "\n";
    columns.each{ header2 += ' <%print
"_"*'+it.size+' %> ' };
    body = '\n<% rows.each {%>';
    // If a value is longer than given
column name, it will be trunked
    columns.each{body += '
${it.'+it.name+'.toString().padRight('+it.si
ze+').substring(0,'+it.size+')}' '};
    body += '\n<% } %>';
```

```
    return header1 + header2 + body +
    footer;
  } }
```

and later "groovyfied" by Gavin Grover:

```
class TableTemplateFactory{
  def columns = []
  def addColumn(name, size) { columns <<
[name:name, size:size]; this }
  def getTemplate() { ""
${columns.collect{ " <%print
\"$it.name\".center($it.size)%> " }.join()}
${columns.collect{ " <%print \"_\"*$it.size
%> " }.join()}
<% rows.each {%>${columns.collect{ "
\${it.$it.name}.toString().padRight($it.siz
e).substring(0,$it.size)} " }.join()}
<% } %>""
  }
}
```

First version is here only as an example of the "groovify process". Of course, the Gavin's version is better.

This class emulate the output of most RDBMS consoles (ie. Oracle SQL\*, MySql).

Here's an usage example (again, grooved up by Gavin):

```

import groovy.text.Template; import
groovy.text.SimpleTemplateEngine
def ttf = new
TableTemplateFactory().addColumn("name",
15).addColumn("age", 4)
def names = [] << [name:"Raffaele",
age:"23"] << [name:"Griorgio", age:"30"]
def binding = ['rows': names]
println new
SimpleTemplateEngine().createTemplate(ttf.te
mplate).make(binding).toString()

```

This is the output:

<b>name</b>	<b>age</b>
<b>Raffaele</b>	<b>23</b>
<b>Griorgio</b>	<b>30</b>

Actually is really limited: column width must be declared, and strings are truncated to that given size.

#### **Wish-list:**

- Automatic column width based on maximum string length
- Multiline records
- Multiline fields
- More formatting options (alignment, case, etc)
- Management of footer fields (totals, formulae, etc)
- Automatic line-wrap based on screen size

#### **Version with ability to wrap text within a column (rather than truncating it).**

The version below is part of a larger project that you are free to grab and use for you own. The main project wiki page is

[here](#). We prefer static typing for our larger projects, so we have modified the original code to suit our style. Hope it is useful for you. Victor Vlasenko authored most of this enhancement under contract for [Build Lackey Labs](#), and I helped clean up the code a bit for presentation on this site [].

**Summary of Enhancement**

This enhancement enables wrapping-to-multi-lines for column text that exceeds the width of a given column. We break out word boundaries (white space) where possible, The implementation we built on truncated the overly long column value instead of wrapping it.

**Example:**



for this data

```
      [col1: '1',  
      col2: 'A very long text  
message for you with a  
superLongWordThatCantBeBrokenAtAWordBoundary  
Withoutwrapping']
```

Instead of this:

```
      col1          col2  
      ----  
-----  
      1          A very long  
text mes
```

We get this:

```
      col1          col2  
      ----  
-----  
      1          A very long  
text  
                message for  
                a  
superLongWordThatC  
antBeBrokenOnWordBou  
ndaryWithoutwrapping
```

```
package com.lackey.provis.cmdline.util

/*
 *   Reproduced from
 *
http://groovy.codehaus.org/Formatting+simple+tabular+text+data
 * (with some code reformatting according to
 my readability preferences)
 */

class TableTemplateFactory {
    def columns = [];    // contains columns
names and their length
    def columnLen = [:]; // contains lengthes
of the columns
    def header1 = '';   // contains columns
names
    def header2 = '';   // contains
underscores
    def body = '';      // the rows of the
table
    def footer = '';    // actually unused:
can contain footer notes, totals, etc.

/**
 * Breaks up long line into multiline at
the word boundary
```

```

*
* TODO move this method to some generic
text utils class
*
* @param input long input line
* @param lineWidth maximum output lines
width
*
* @return multiline as an array of
strings
*/
protected static List<String>
wrapLine(input, lineWidth) {
    List<String> lines = []
    def line = ""
    def addWord;

    addWord = {word ->
        // Add new word if we have space in
current line
        if ((line.size() + word.size()) <=
lineWidth) {
            line <<= word
            if (line.size() < lineWidth)
                line <<= " "
            // Our word is longer than line
width, break it up
        } else if (word.size() > lineWidth) {
            def len = lineWidth - line.length()
            line <<= word.substring(0, len)
            word = word.substring(len)

```

```

        lines += line.toString()

        while (word.size() > lineWidth) {
            lines += word.substring(0,
lineWidth);
            word = word.substring(lineWidth);
        }
        line = word
        if (line.size() > 0 && line.size() <
lineWidth)
            line <<= " "
            // No more space in line - wrap to
another line
        } else {
            lines += line.toString()
            line = ""

            addWord(word)
        }
    }

    input.split(" ").each() {
        addWord(it)
    }

    lines += line.toString()

    return lines
}

/**

```

```
* Wraps values in rows according to the
column width. Value wrapping performed at
* the word boundary.
```

```
*
```

```
* @param rows input rows array
```

```
*
```

```
* @return rows array with multiline
values
```

```
*/
```

```
public List<Map<String,String>>
wrapRows(unwrappedRows) {
    List<Map<String,List<String>>>
multilineRows = []
    List<Integer> rowHeights = []

    // Preprare unwrappedRows with multiline
values
    unwrappedRows.each() {
        unwrappedRow ->
        def multiLineRow = [:]
        int height = 1
        unwrappedRow.each() {
            column -> // column in unwrapped
row
                List<String> multilineValue =
wrapLine(column.value,
columnLen[column.key])
                if (multilineValue.size() > height)
                    height = multilineValue.size()
                multiLineRow[column.key] =
multilineValue // multiLineValue is list
```

```

of strings
    }
    multilineRows << multiLineRow
    rowHeights << height
}

return
foldMultiLineRowsIntoPlainRows(multilineRows
, rowHeights)
}

// For each array of strings (wrapped
lines) in multiLineRows we fold those in to
// a new array of rows (plain rows). For
any given column that consists of an array
// of wrapped lines we either insert the
appropriate wrapped line, or a blank if no
// more lines are left for that column.
//
private List<Map<String,String>>
foldMultiLineRowsIntoPlainRows(

List<Map<String,List<String>>>
multilineRows,

List<Integer> rowHeights) {
    List<Map<String,String>> plainRows = []
    multilineRows.eachWithIndex() {
        Map<String,List<String>> mlRow, int
idx ->

```

```

    int height = rowHeights[idx]

    for (i in 0..<height) {
        Map<String,String> row = [:]

        mlRow.each() {
            Map.Entry<String, List<String>>
col ->
                List<String>
listOfStringsForColumn = mlRow[col.key]
                row[col.key] =
listOfStringsForColumn[i] ?: ""
                }
            plainRows << row
        }
    }

    return plainRows
}

public TableTemplateFactory
addColumn(String name, int size) {
    columns << [name: name, size: size];
    columnLen[name] = size
    return this
}

def getTemplate() {
    header1 = "\n";
    columns.each {
        header1 += ' <%print "' + it.name +

```

```

''.center(' + it.size + ')%> '
    };
    header2 = "\n";
    columns.each {
        header2 += ' <%print "_"* ' + it.size +
' %> '
    };
    body = '\n<% rows.each {%>';

    // If a value is longer than given
column name, it will be trunked
    columns.each {
        body +=
            '{it.' +
                it.name +
'.toString().padRight(' +
                it.size + ').substring(0,' +
                it.size +
                ')} '
    };
    body += '\n<% } %>';
    return header1 + header2 + body +
footer;
    }
}

```

```

package com.lackey.provis;

```

```

/*

```

```

*

```

```

* Copyright 2009. Build Lackey Labs. All

```

**Rights Reserved.**

\*

\*

\* Licensed under the Apache License,  
Version 2.0 (the "License");

\* you may not use this file except in  
compliance with the

\* License. You may obtain a copy of  
the License at

\*

\*

<http://www.apache.org/licenses/LICENSE-2.0>

\*

\* Unless required by applicable law  
or agreed to in writing, software

\* distributed under the License is  
distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF  
ANY KIND, either express or implied.

\*

\* See the License for the specific  
language governing permissions

\* and limitations under the License.

\*

\* Author: cbedford

\* Date: Jan 18, 2009

\* Time: 3:21:52 PM

\*/

`import org.testng.annotations.*`

```
import org.testng.TestNG
import org.testng.TestListenerAdapter

import org.slf4j.*
import groovy.text.Template;
import groovy.text.SimpleTemplateEngine
import
com.lackey.provis.cmdline.util.TableTemplate
Factory
```

```
public class TestTableTemplates {
    private static Logger logger =
LoggerFactory.getLogger(TestTableTemplates.c
lass.name)
```

```
    @Test (enabled = true)
    public void test_X() {
        final TableTemplateFactory factory = new
TableTemplateFactory()
        def ttf = factory.addColumn("name",
15).addColumn("age", 4)
        def names = [] << [name:"Raffaele Smoke
BOON laforge calm nwe box ooex", age:"23"]
<< [name:"Griorgio", age:"30"]
        def binding = ['rows': names]
        println new
SimpleTemplateEngine().createTemplate(ttf.te
mplate).make(binding).toString()
    }
```

```
@Test (enabled = true)
public void testWrapLineSingleWord() {
    TableTemplateFactory ttf = new
TableTemplateFactory()
    assert (ttf.wrapLine("aa bb cc", 2) ==
["aa", "bb", "cc"])
}
```

```
@Test (enabled = true)
public void testWrapLineSeveralWords() {
    TableTemplateFactory ttf = new
TableTemplateFactory()
    assert (ttf.wrapLine("This is some text
that must be broken up to multiline" +
        " at a word boundary", 9) == ["This
is ", "some text", "that must", "be broken",
        "up
to ", "multiline", "at a word", "boundary
"])
}
```

```
@Test (enabled = true)
public void testWrapLineWithLongWord() {
    TableTemplateFactory ttf = new
TableTemplateFactory()
    assert (ttf.wrapLine("A very long text
message for you with a" +
        "
superLongWordThatCantBeBrokenOnWordBoundaryW
ithoutwrapping", 20) ==
        ["A very long text ", "message for
```

```

you with", "a superLongWordThatC",
        "antBeBrokenOnWordBou",
"ndaryWithoutwrapping" ] )
    }

    @Test (enabled = true)
    public void
testMultilineRowWrappingWithActualPrinting()
{
    def ttf = new
TableTemplateFactory().addColumn("name",
15).addColumn("age", 4)
    def names = [] << [name:"Raffaele
Gudinio BanderasHulioLopezMakachino",
age:"819 4096"] << [name:"Griorgio",
age:"30"]
    def wrappedNames = ttf.wrapRows(names)
    assert wrappedNames ==
        [ ["name": "Raffaele ",
"age": "819 "],
            ["name": "Gudinio
Bandera", "age": "4096"],
            ["name": "sHulioLopezMaka", "age": ""],
            ["name": "chino ",
"age": ""],
            ["name": "Griorgio ",
"age": "30 "]]

    def binding = ['rows': wrappedNames ]
    final String templateOutput =

```

```
getTemplateOutput(binding, ttf)
    System.out.println templateOutput
}

private String
getTemplateOutput(Map<Object, List> binding,
TableTemplateFactory ttf) {
    return new
SimpleTemplateEngine().createTemplate(ttf.te
```

```
mplate).make(binding).toString()
    }
}
```

## Greedy Coin Changer in Groovy

Here is a version of the [Greedy](#) Coin Changer problem inspired by these [Scala](#) and [Python](#) solutions.

```
enum UsCoin {
    quarter(25), dime(10), nickel(5),
penny(1)
    UsCoin(v) { value = v }
    final value
}

enum OzzieCoin {
    fifty(50), twenty(20), ten(10), five(5)
    OzzieCoin(v) { value = v }
    final value
}

def plural(word, count) {
    if (count == 1) return word
    word[-1] == 'y' ? word[0..-2] + "ies" :
word + "s"
}

def change(currency, amount) {
```

```
currency.values().inject([]){ list, coin
->
    int count = amount / coin.value
    amount = amount % coin.value
    list += "$count
    ${plural(coin.toString(), count)}"
    }
}

println change(UsCoin, 71)    // => [2
```

```
quarters, 2 dimes, 0 nickels, 1 penny]
println change(OzzieCoin, 95) // => [1
fifty, 2 twenties, 0 tens, 1 five]
```

This solution uses Java 5 enums (supported in Groovy 1.5+) but other variations are possible.

## Groovy Alternatives to Inner Classes

Groovy 1.0 and 1.1 do not support normal Java inner classes (you can have such classes only in Scripts or you can code them in Java of course).

In many cases though, you don't need them. Here is a Groovy 1.1 example which shows you how to construct a class on the fly which implements several interfaces. This would be a prime example where you would typically use an inner class in Java. In Groovy we don't have to:

```
import groovy.swing.SwingBuilder
import static java.awt.BorderLayout.*
import java.awt.event.*

// set up variables
count = 0
def textlabel
def text = "Actions: "
def update = { c ->
    text += c
    textlabel.text = text
}

// create the listener
def closureMap = [
    mousePressed:    { update 'M' },
    keyPressed:      { update 'K' },
    focusLost:       { update 'F' },
    windowIconified: { update 'W' }
]
```

```
def interfaces = [WindowListener,  
KeyListener, MouseListener, FocusListener]  
def listener =  
ProxyGenerator.instantiateAggregate(closureM  
ap, interfaces)  
  
// now the GUI  
def swing = new SwingBuilder()  
def frame = swing.frame(title:'Frame') {  
    BorderLayout()  
    textlabel = label(text:text,  
constraints: NORTH)  
    button = button(text:'Click Me',  
constraints: SOUTH)  
}  
frame.addWindowListener listener  
['Key', 'Mouse', 'Focus'].each {  
    button."add${it}Listener" listener  
    textlabel."add${it}Listener" listener
```

```
}  
frame.pack()  
frame.show()
```

## Groovy as script

The debate is not over, scripting vs programming. In fact the definitions between these two still go blurry. In the context of Java, a program comprises of definition of a class and its structure. For example, the simplest program may have just the class definition.

```
class ClassName {  
}
```

Obviously, this particular code doesn't do anything other than just defining the class. A bare minimum useful program may contain a couple of more lines of code in Java. Lets try to write a program that prints "hello world"

```
class WorldGreeter {  
    public static void Main(String args[]) {  
        System.out.println ("hello world") ;  
    }  
}
```

The same greeter program can be re-written using a static block or even with the constructor. So, programs follow some strict guidelines in terms of structure.

On the other hand, scripts have a mixture of free-hand statements and semi-to-fully structured method elements. We will try to write the greeter program in groovy.

```
println "hello world"
```

Thats it!! No class definitions, no curly brackets, no strict syntax checks such as semicolons. Just a simple readable one line statement. All other overheads have been "encapsulated". After all encapsulation is one of the qualities of any Object Oriented Program (oops, Script), isn't it?

Ok, lets not delve too much into the contradiction between script and program, neither it is true 'encapsulation' or not. This article is to show the power of scripting using Groovy by walking you with a business example. Enough has been said about the short-hand structure of groovy, how it shrinks coding elements into small fractions to achieve

the same result as its counterpart, Java. So, we will take a business problem and try to solve using a groovy script. Think of groovy as a replacement to your perl scripting or shell scripting, in this context. Groovy is not the competitor to Java, it is just complementing it.

Feed files are general scenario in business applications. Applications receive/fetch feed files from another application or location, to process and import into the application stream. There are many situations where we need to create a diff or delta between two successive feeds to make the feed processing perform better. For our example, lets assume a feed with an XML structure given below.

```
<WorkRequests>  
  <WorkRequest id='1'>  
    <Client>  
      <name>BigClient</name>  
      <businessUnit>DotCom</businessUnit>  
    </Client>  
    <RequestType>Requested</RequestType>  
    <RequestDate>2009-12-31</RequestDate>  
    <ProductInfo>Create the plugin component  
for the upper deck</ProductInfo>  
  </WorkRequest>  
</WorkRequests>
```

Briefly on the structure, the external entities (customer applications) send work requests to the enterprise application, through feed files. The application processes all the work request records by navigating the tree like structure of different work request elements. Basically each request contains client information, request type, date, supporting product information and order processing information. The real functionality of the application is something out of scope for this example.

Our script needs exactly two input data to create the delta file, viz., current feed file, last run feed file. The script has to validate and respond to the caller based on the values of the input elements. The idea is to build the script with three sections, first one to verify the input data, second to define a reusable block of code that does the actual check and find out the difference and finally, a small block that writes the output into a new file. On the way we will handle some exception cases also.

Lets start building the script by validating the input data.

```
if (this.args.size() !=2 ) {  
    println "this script expects exactly two  
arguments"  
    // You can even print the usage here.  
    return  
}
```

If the number of arguments is anything not equal to 2, then we will not proceed. We can print the usage and exit too.

So, the arguments are string values of input file, file to compare and the output folder name.

```

def inputFile = new File(this.args[0])

def fileToCompare = new File(this.args[1])

// Some more validations.
if (!inputFile.exists()) {
    // Nothing to process, return
    println "The input file doesn't exist.
Process incomplete"
    return
}

if (!fileToCompare) {
    // invalid file for comparison
    println "File to compare doesn't exist.
Process incomplete"
    return
}

// If we are here, then the input data is
valid, we can start the main processing.

```

Lets skip some of the sanity checks such as whether the input file is an XML, is it following the XSD structure etc. Lets assume the well structured, required XML files, ready for processing. We have some special parser API available in groovy for processing XML files as dotted notations. Lets load the input file using the API.

```

def todaysFile = new XmlSlurper().parse
(inputFile)

```

With this single statement we have loaded the XML file into memory and all the elements of the XML file are available as dotted notations and as XPATH query elements. All we are interested is to check whether each workrequest record in today's file is a new/modified record in comparison with the "corresponding record" of the last run feed file. Here another assumption is that the id attribute of the work request is something like a primary key and never changes.

As we loaded todaysFile into memory we will load the fileToCompare also.

```
def lastRunFile = new  
XmlSlurper().parse(fileToCompare)
```

Now we need to write a simple closure routine that fetches the WorkRequest record from the last run file by searching the workrequest id.

```
def getLastRunWorkRequest = { wrID ->  
  def lastRunWorkRequest =  
  lastRunFile.WorkRequest.find(it. @id.text()  
  == wrID)  
  return lastRunWorkRequest  
}
```

The above routine searches the entire last run file, looking for a specific work request using the xpath query. If it finds a matching record, then the entire work request is returned. otherwise a null will be returned. Now we need to write another small routine that verifies whether two elements (of todays file and last run file) are changed.

```
def dataChanged = { todays, lastrun ->  
  return todays.toString() !=  
  lastrun.toString()  
}
```

Simple enough, isn't it?

With the two helper routines ready to fire, we need to loop through the feed file to see which record is changed and which is not. The changed or the new records will have to be included in the delta file and the unchanged records will be left out.

```

def deltaRequests = []

todaysFile.WorkRequest.each {
    def todays = it
    def lastrun = getLastRunWorkRequest
(todays.@id.text())
    if ( (!lastrun) || (dataChanged (todays,
lastrun) ) ) {
        deltaRequests.add (todays)
    }
}

```

Now we have collected all the records to be processed in the deltaRequests list. Only thing pending is to attach this list under workrequests element of the XML and write the delta file. To write the XML output there is an API by name groovy.xml.StreamingMarkupBuilder. We will use this to write the file.

```

def mb = new
groovy.xml.StreamingMarkupBuilder()

def doc = mb.bind {
    WorkRequests {
        mkp.yield deltaRequests
    }
}

def outputWriter = new FileWriter(new
File(inputFile).name + '_Delta')
outputWriter << doc

```

With very simple steps of processing, we have written a clean script that processes two similar structured XML file and creates the delta file. When this script is plugged in into the application that processes redundant feeds on a

daily process, we can significantly reduce the processing time by just creating the delta file and making the delta file as the input to the application.

## Guy Steele's word split example using Groovy and GPar

The Fortress programming language has some very powerful features. Guy Steele's slides [here](#) (slide 52 onwards) illustrate some of its features including how to split a sentence into words in parallel.

Here we look at some ways to use Groovy and GPar to solve that problem.

First we can write an imperative sequential version:

```
def swords = { s ->
  def result = []
  def word = ''
  s.each{ ch ->
    if (ch == ' ') {
      if (word) result += word
      word = ''
    } else word += ch
  }
  if (word) result += word
  result
}
```

And test it as follows:

```
assert swords("Here is a sesquipedalian
string of words") ==
  ['Here', 'is', 'a',
  'sesquipedalian', 'string', 'of', 'words']
```

Now, we can ask ourselves a question. Is this solution in its current form easy to turn into a parallel solution? The answer is NO. Each step in our loop relies on the result from the previous step. So, if I split the sentence into say two pieces, and I am processing the start of the second piece, I don't know whether a character or space finished the previous piece. I.e. the order in which I process information is important in the current algorithm and to make it work easily in parallel, we need to refactor the algorithm into some equivalent one where the order is no longer important. Technically we require that the algorithm is associative. The figure below shows what we mean.

## Associativity (order doesn't matter)

$$(\square \diamond \square) \diamond \square = \square \diamond (\square \diamond \square) = \square \diamond \square \diamond \square$$

$\diamond$  = any operator, e.g. +, \*, concat

If we have three things to calculate, it doesn't matter if we do the first two first or the last two first.

Another useful property that would be good for our algorithm to have is left and right identity ("zero") elements.

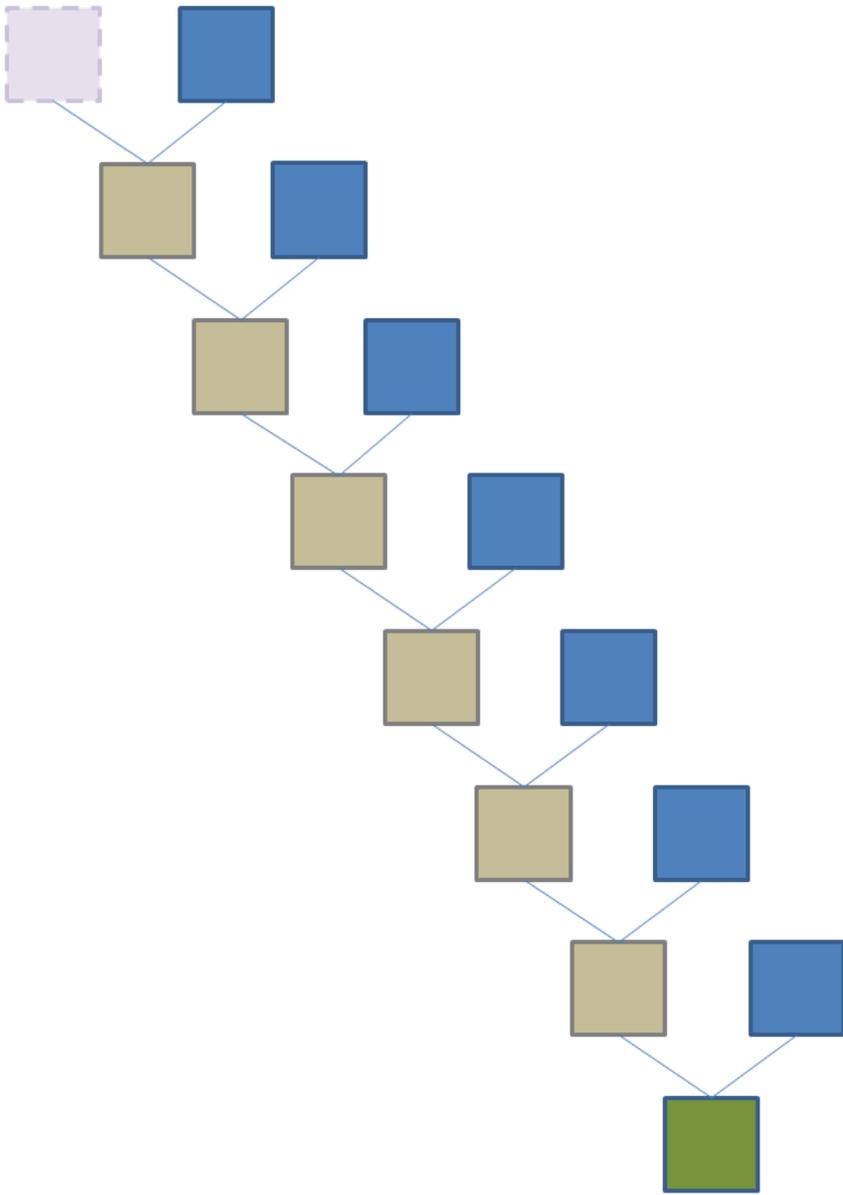
## Some Zero or identity element

$$\square \diamond \square = \square \diamond \square = \square$$

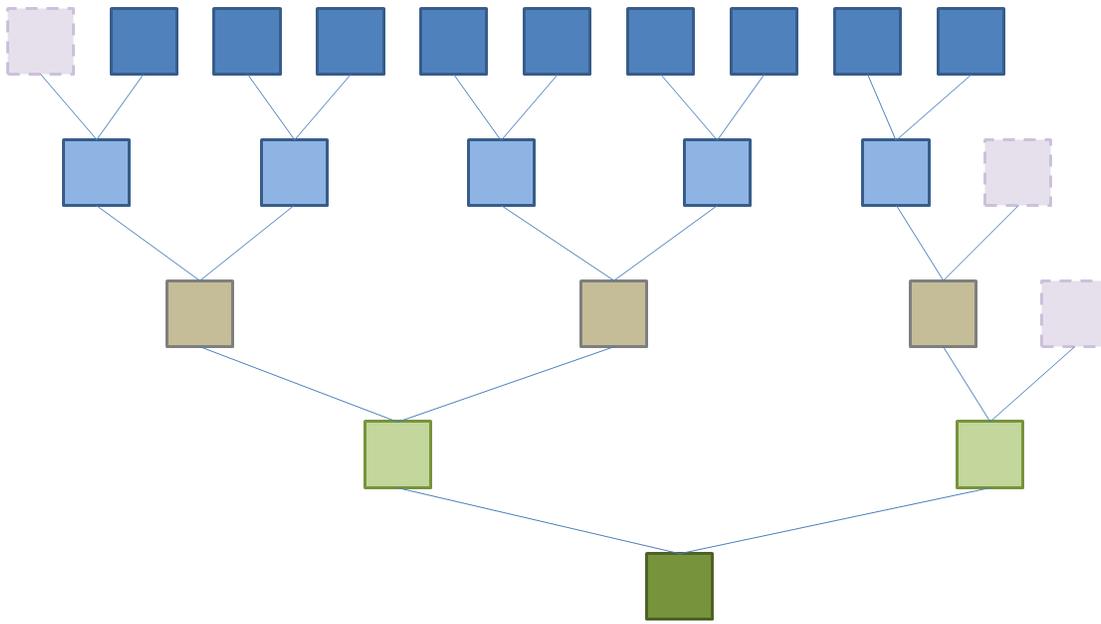
$\diamond$  = any operator, e.g. +, \*, concat

$\square$  = some zero/identity element, e.g. 0 for +, 1 for \*

This property allows us to break our data up into pieces and (if needed) substitute a ZERO element at one or the other end, i.e. our algorithm will play nicely at the boundary points. Basically we can turn this:



into this:



Technically, combining associativity with left and right identity gives us a monoid. The details aren't important here but basically algorithms with such properties can be easily made parallel. This is exactly what we want when using techniques such as map/reduce or divide and conquer as part of our solution.

## Monoid = Associative + left & right identity elements

So getting back to word splitting we need to change the algorithm to remember the boundary conditions. To do that we will create a little domain for remembering "chunks" and "segments" of partial results. Here is one way to code that solution (and credit goes to Guy Steele's article for devising this domain):

```

class Util { static maybeWord(s) { s ? [s] :
[] } }
import static Util.*

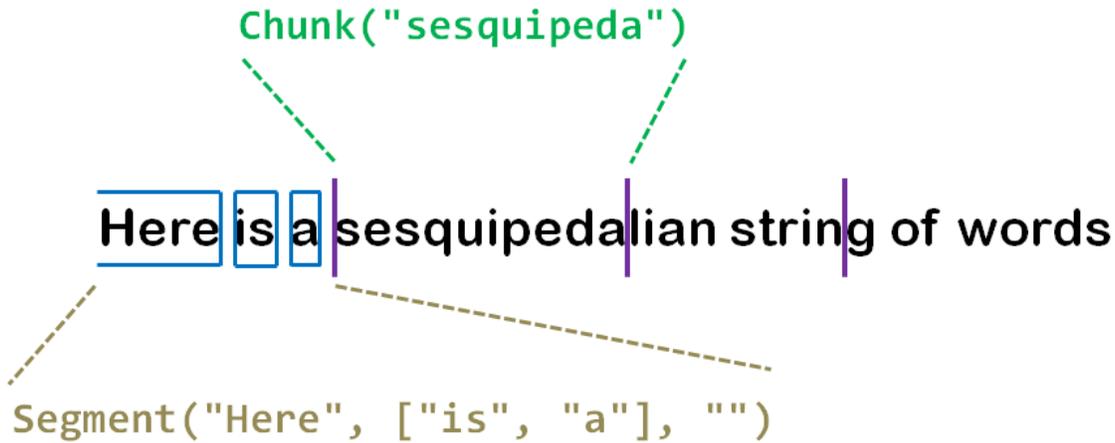
@Immutable class Chunk {
    String s
    public static final ZERO = new Chunk('')
    def plus(Chunk other) { new Chunk(s +
other.s) }
    def plus(Segment other) { new Segment(s +
other.l, other.m, other.r) }
    def flatten() { maybeWord(s) }
}

@Immutable class Segment {
    String l; List m; String r
    public static final ZERO = new Segment('',
[], '')
    def plus(Chunk other) { new Segment(l, m,
r + other.s) }
    def plus(Segment other) { new Segment(l, m
+ maybeWord(r + other.l) + other.m, other.r)
}
    def flatten() { maybeWord(l) + m +
maybeWord(r) }
}

```

We'll have more to say about these classes later but basically we have plus ("+") methods for adding Segments and Chunks, a ZERO identity element and a flatten method which takes our Chunk or Segment and converts it into a list of words.

Guy Steele's slides show using this domain in more detail. Here is a summary snapshot to illustrate the main idea.



Now we are ready to write a slightly more functional flavored sequential solution:

```
def processChar(ch) { ch == ' ' ? new
Segment(' ', [], ' ') : new Chunk(ch) }
def swords(s) { s.inject(Chunk.ZERO) {
result, ch -> result + processChar(ch) } }
```

And test it as follows:

```
assert swords("Here is a sesquipedalian
string of words").flatten() ==
    ['Here', 'is', 'a',
'sesquipedalian', 'string', 'of', 'words']
```

More importantly, because of the properties of the classes in our domain, we are ready to write some parallel solutions. Before doing that, let's take a slight diversion to illustrate how we might further check some of our algorithm's properties. We'll use the Java port of quickcheck:

```
@GrabResolver('http://download.java.net/maven/2')
@Grab('net.java:quickcheck:0.5')

import static
net.java.quickcheck.generator.CombinedGeneratorsIterables.someNonEmptyLists
import static
net.java.quickcheck.generator.PrimitiveGenerators.nonEmptyStrings

def total = 0

someNonEmptyLists(nonEmptyStrings()).each {
words ->
    def someSegment = words.size() > 2 ?
        new Segment(words[0], words[1..-2],
words[-1]) :
        new Segment('', words, '')
    def expected = someSegment.flatten()
    [Chunk.ZERO, Segment.ZERO].each { zero ->
        assert expected == (someSegment +
zero).flatten()
        assert expected == (zero +
someSegment).flatten()
    }
    total++
}

println "$total tests"
```

This produces the output:

**200 tests**

and gives me confidence that my algorithm is a monoid. The above showed that `Segments` have left and right identity elements for "+" (in fact both `Check.ZERO` and `Segment.ZERO` work). If we wanted to, we could have additional checks for the associativity piece or for properties of `Chunk`.

What have we achieved? Well, now we are in a position to write some parallel versions of our algorithm. Many have a common strategy and that is to divide the input into sections. Typically the division stops once the sections reach a certain granularity of size. As a general rule, if we divide past a certain level of granularity, then the overheads associated with setting up the parallelism outweigh the parallelism gains.

Here's one version which uses an old school concurrent hash map. This version divides the input sentence into 4 pieces and solves each piece in a separate thread, storing the results into a concurrent hash map.

```
THREADS = 4  
def pwords(s) {  
  int n = (s.size() + THREADS - 1) / THREADS  
  def map = new ConcurrentHashMap()  
  (0..<THREADS).collect { i ->  
    Thread.start {  
      def (min, max) = [[s.size(), i *  
n].min(), [s.size(), (i + 1) * n].min()]  
      map[i] = swords(s[min..<max])  
    }  
  }* .join()  
  (0..<THREADS).collect { i -> map[i]  
}.sum().flatten()  
}
```

And test it using this code:

```
assert pwords("Here is a sesquipedalian
string of words") ==
    ['Here', 'is', 'a',
'sesquipedalian', 'string', 'of', 'words']
```

Alternatively, I can use GPar and not have to deal with as much explicit synchronization.

Firstly, we'll look at a map reduce version. It is comprised of a partition part which splits the data up - in our case into 4 pieces. Then the map part runs our serial solution for each piece - notice that the mapping is independent and can be run completely in parallel. For the reduce part we will use our "+" monoid. Here is the code:

```
@Grab('org.codehaus.gpars:gpars:0.11')
import static
groovyx.gpars.GParsPool.withPool

THRESHHOLD = 10

def partition(piece) {
    piece.size() <= THRESHHOLD ? piece :
        [piece[0..<THRESHHOLD]] +
partition(piece.substring(THRESHHOLD))
}

def pwords = { input ->
    withPool(THREADS) {

partition(input).parallel.map(swords).reduce
{ a, b -> a + b }.flatten()
    }
}
```

The previous example used an explicit reduce to make that part of our solution obvious but we can also simplify a bit more using the built-in sum reduction:

```
partition(input).parallel.map(swords).sum().  
flatten()
```

As an alternative, we can use the fork/join capabilities which are available in Java 7 (or the JSR166y library - this library is typically bundled with GParS):

```
def pwords(input) {  
  withPool(THREADS) {  
    runForkJoin(0, input.size(), input) {  
first, last, s ->  
      def size = last - first  
      if (size <= THRESHHOLD) {  
        swords(s[first..<last])  
      } else { // divide and conquer  
        int pivot = first + size / 2  
        forkOffChild(first, pivot, s)  
        forkOffChild(pivot, last, s)  
        childrenResults.sum()  
      }  
    }.flatten()  
  }  
}
```

Alternatively, we can use GParS' dataflow capabilities. With dataflow we write expressions which declaratively express the relationships in our data. In our case the pieces must be summed (using our "+" monoid) once they have been calculated:

```

import groovyx.gpars.dataflow.DataFlows
import static
groovyx.gpars.dataflow.DataFlow.task

def partition(s, n, i) { s[[s.size(), i *
n].min()..<[s.size(), (i + 1) * n].min()] }

def pwords(s) {
    int n = (s.size() + THREADS - 1) / THREADS
    new DataFlows().with {
        task { a = swords(partition(s, n, 0)) }
        task { b = swords(partition(s, n, 1)) }
        task { c = swords(partition(s, n, 2)) }
        task { d = swords(partition(s, n, 3)) }
        task { sum1 = a + b }
        task { sum2 = c + d }
        task { sum = sum1 + sum2 }
//          task { sum = a + b + c + d } //
    alternative
        sum
    }.flatten()
}

```

Finally, we can create a parallel array version. This version (as currently written) doesn't limit the amount of parallelism to some granularity level, instead it follows the same approach as our functional sequential version but just replaces inject with collectParallel which will automatically perform its steps in parallel. Here is the code:

```

def pwords(input) {
  withPool(THREADS) {
    input.collectParallel{ processChar(it)
  }.sumParallel().flatten()
  }
}

```

Just as one final example, here is a variation of our original ConcurrentHashMap version but making use of GParas agents to guard a non-concurrent HashMap.

```

def pwords(s) {
  int n = (s.size() + THREADS - 1) / THREADS
  def agent = new Agent<Map>([:],
{it?.clone()})
  (0..<THREADS).collect { i ->
    Thread.start {
      def (min, max) = [[s.size(), i *
n].min(), [s.size(), (i + 1) * n].min()]
      def result = swords(s[min..<max])
      agent << { it[i] = result; updateValue
it }
    }
  }*.join()
  assert agent.val instanceof LinkedHashMap
  (0..<THREADS).collect { i -> agent.val[i]
}.sum().flatten()
}

```

Here, the agent acts as a synchronization layer between our code and the standard LinkedHashMap (the default map for Groovy). This offers no particular advantage over ConcurrentHashMap but illustrates how agents work in general. We supply the agent with code to run and it runs that code after performing the necessary synchronization.

## How many Triangles (list, regex, grep) example

Motivation for this page was to add more list and regex example code. I had a hard time creating the regex which was acceptable to the groovy interpreter or which worked as I expected. I found examples of using static regex but very little variable based regex.

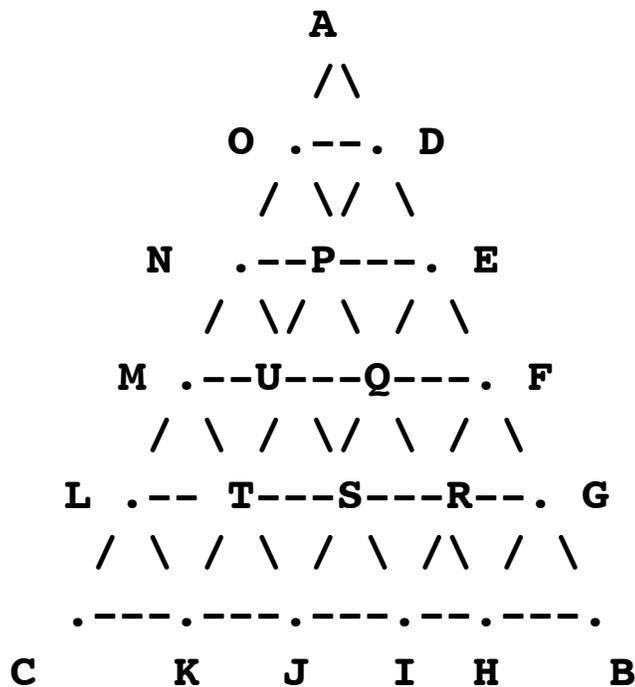
Secondly, I had seen this problem posed in multiple places and a solution in perl for the "lines2" which did not seem to work for "lines1". (my mistake, did not use enough points in the loops)

Problem: Count all triangles in the diagram.

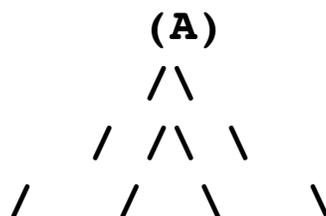
Solution: select 3 points shared by 3 different lines

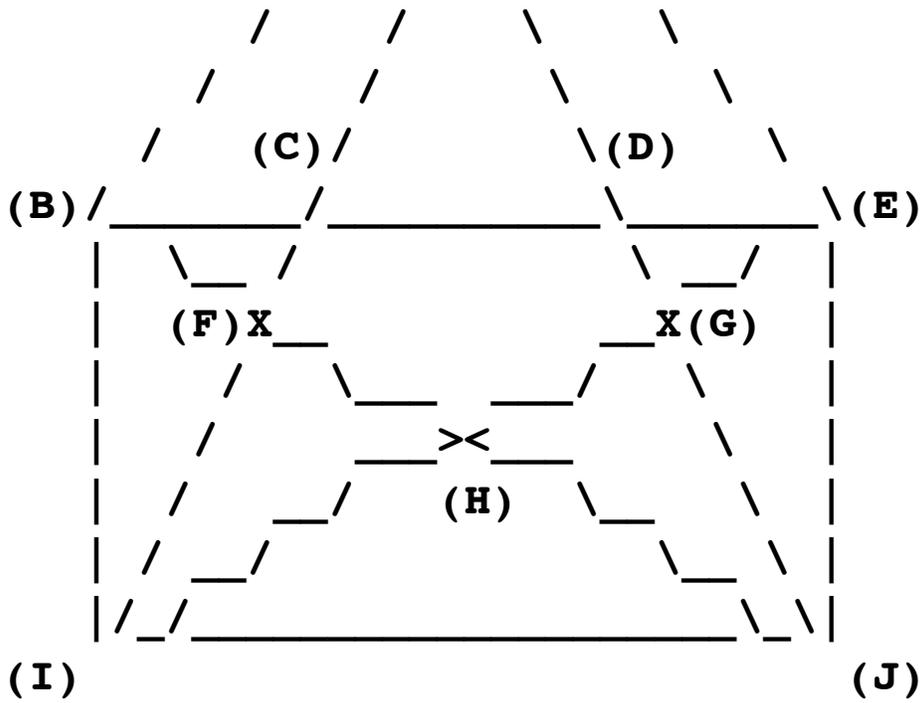
```
/*
```

```
lines1
```



```
lines2
```





\*/

```

lines2 = ['ae',
            'adgj',
            'acfi',
            'ab',
            'bcde',
            'bfhj',
            'bi',
            'ij',
            'ihge',
            'je'
            ]

lines1 = ['adefgb',
            'aonmlc',
            'bhijkc',

```

```
'do',  
'dputk',  
'epn',  
'eqsj',  
'fri',  
'fqum',  
'gh',  
'grstl',  
'hrqpo',  
'isun',  
'jtm',  
'kl'  
]
```

```
// Echo Lines
```

```
def computeTriangles =  
{ lines ->
```

```
  println()
```

```
  // Initialize
```

```
  count = 0
```

```
  size = lines.size
```

```
  println "find triangles, $count, $size"
```

```
  for (pt1 in 'a'..'u')
```

```
  {
```

```
    for (pt2 in 'b'..'u')
```

```

    {
        for (pt3 in 'c'..'u')
        {
            line =
lines.grep(~/(.*$pt1.*$pt2.*|.*$pt2.*$pt3.*|
.*$pt1.*$pt3.*)/)
            if (line.size == 3)
            {
                println (++count + ":
$pt1,$pt2,$pt3 : " + line )
            }
        }
    }
}

```

```

computeTriangles (lines1)
computeTriangles (lines2)

```

```
// Termination
```

```
println ("Terminated Normally")
```

Output

```
find triangles, 0, 15
1: a,b,c : ["adefgb", "aonmlc", "bhijkc"]
2: a,d,o : ["adefgb", "aonmlc", "do"]
3: a,e,n : ["adefgb", "aonmlc", "epn"]
4: a,f,m : ["adefgb", "aonmlc", "fqum"]
5: a,g,l : ["adefgb", "aonmlc", "grstl"]
6: d,b,k : ["adefgb", "bhijkc", "dputk"]
7: d,e,p : ["adefgb", "dputk", "epn"]
8: d,f,u : ["adefgb", "dputk", "fqum"]
9: d,g,t : ["adefgb", "dputk", "grstl"]
10: d,p,o : ["do", "dputk", "hrqpo"]
11: e,b,j : ["adefgb", "bhijkc", "eqsj"]
12: e,f,q : ["adefgb", "eqsj", "fqum"]
13: e,g,s : ["adefgb", "eqsj", "grstl"]
14: e,q,p : ["epn", "eqsj", "hrqpo"]
15: e,s,n : ["epn", "eqsj", "isun"]
16: f,b,i : ["adefgb", "bhijkc", "fri"]
17: f,g,r : ["adefgb", "fri", "grstl"]
18: f,i,u : ["fri", "fqum", "isun"]
19: f,r,q : ["fri", "fqum", "hrqpo"]
20: g,b,h : ["adefgb", "bhijkc", "gh"]
21: g,h,r : ["gh", "grstl", "hrqpo"]
22: h,o,c : ["aonmlc", "bhijkc", "hrqpo"]
23: h,p,k : ["bhijkc", "dputk", "hrqpo"]
24: h,q,j : ["bhijkc", "eqsj", "hrqpo"]
```

25: h,r,i : ["bhijkc", "fri", "hrqpo"]  
26: i,n,c : ["aonmlc", "bhijkc", "isun"]  
27: i,s,j : ["bhijkc", "eqsj", "isun"]  
28: i,u,k : ["bhijkc", "dputk", "isun"]  
29: j,m,c : ["aonmlc", "bhijkc", "jtm"]  
30: j,t,k : ["bhijkc", "dputk", "jtm"]  
31: k,l,c : ["aonmlc", "bhijkc", "kl"]  
32: p,o,n : ["aonmlc", "epn", "hrqpo"]  
33: p,u,n : ["dputk", "epn", "isun"]  
34: q,j,m : ["eqsj", "fqum", "jtm"]  
35: q,o,m : ["aonmlc", "fqum", "hrqpo"]  
36: q,p,u : ["dputk", "fqum", "hrqpo"]  
37: q,s,u : ["eqsj", "fqum", "isun"]  
38: r,i,s : ["fri", "grstl", "isun"]  
39: r,o,l : ["aonmlc", "grstl", "hrqpo"]  
40: r,p,t : ["dputk", "grstl", "hrqpo"]  
41: r,q,s : ["eqsj", "grstl", "hrqpo"]  
42: s,j,t : ["eqsj", "grstl", "jtm"]  
43: s,n,l : ["aonmlc", "grstl", "isun"]  
44: s,u,t : ["dputk", "grstl", "isun"]  
45: t,k,l : ["dputk", "grstl", "kl"]  
46: t,m,l : ["aonmlc", "grstl", "jtm"]  
47: u,n,m : ["aonmlc", "fqum", "isun"]  
48: u,t,m : ["dputk", "fqum", "jtm"]

find triangles, 0, 10

1: a,b,c : ["acfi", "ab", "bcde"]  
2: a,b,d : ["adgj", "ab", "bcde"]  
3: a,b,e : ["ae", "ab", "bcde"]  
4: a,b,f : ["acfi", "ab", "bfhj"]  
5: a,b,i : ["acfi", "ab", "bi"]

6: a,b,j : ["adgj", "ab", "bfhj"]  
7: a,c,d : ["adgj", "acfi", "bcde"]  
8: a,c,e : ["ae", "acfi", "bcde"]  
9: a,d,e : ["ae", "adgj", "bcde"]  
10: a,f,j : ["adgj", "acfi", "bfhj"]  
11: a,g,e : ["ae", "adgj", "ihge"]  
12: a,i,e : ["ae", "acfi", "ihge"]  
13: a,i,g : ["adgj", "acfi", "ihge"]  
14: a,i,j : ["adgj", "acfi", "ij"]  
15: a,j,e : ["ae", "adgj", "je"]  
16: b,c,f : ["acfi", "bcde", "bfhj"]  
17: b,c,i : ["acfi", "bcde", "bi"]  
18: b,d,j : ["adgj", "bcde", "bfhj"]  
19: b,f,i : ["acfi", "bfhj", "bi"]  
20: b,h,e : ["bcde", "bfhj", "ihge"]  
21: b,i,e : ["bcde", "bi", "ihge"]  
22: b,i,h : ["bfhj", "bi", "ihge"]  
23: b,i,j : ["bfhj", "bi", "ij"]  
24: b,j,e : ["bcde", "bfhj", "je"]  
25: c,i,e : ["acfi", "bcde", "ihge"]  
26: d,g,e : ["adgj", "bcde", "ihge"]  
27: d,j,e : ["adgj", "bcde", "je"]  
28: f,i,h : ["acfi", "bfhj", "ihge"]  
29: f,i,j : ["acfi", "bfhj", "ij"]  
30: g,j,e : ["adgj", "ihge", "je"]  
31: h,g,j : ["adgj", "bfhj", "ihge"]  
32: h,j,e : ["bfhj", "ihge", "je"]  
33: i,g,j : ["adgj", "ij", "ihge"]

```
34: i,h,j : ["bfhj", "ij", "ihge"]
35: i,j,e : ["ij", "ihge", "je"]
Terminated Normally
```

## Installing or Update Groovy Automatically

Do you need to install or update Groovy? Try java [InstallGroovy](#) `groovy-binary-1.5.0.zip`, this is a simple java program to download, unzip and link a Groovy binary release package into your /opt local system. The source code is also [attached](#).

NOTE: the class file is compiled with Java 1.5, so you will need at least that to run it. I think Groovy currently support Java 1.4 though, oh well, you just can't get everything. Can you?

Also see [Auto setup and download dependencies jars](#)

## InstallTextMateGroovyBundle

```
#!/usr/bin/env groovy

/**
 * Download and setup TextMate Bundle for
 * groovy. Assume you have svn and groovy
 * already installed.
 * @author Zemian Deng
 * @since Thu Nov 15 15:02:11 EST 2007
 */

path =
System.properties[ 'user.home' ]+"/Library/App
lication Support/TextMate/Bundles"
dir = new File(path)
if(!dir.exists()){
    println "Creating path ${path}"
    dir.mkdirs()
}

println "Checking out groovy bundle..."
proc = "svn co
http://macromates.com/svn/Bundles/trunk/Bund
les/Groovy.tmbundle".execute(
    [ "LC_CTYPE=en_US.UTF-8" ],
    dir
)
proc.waitFor()
println proc.text

println "Verifying groovy bundle..."
dir.list().each{f-> println f}
```

# Integrating Groovy in an application - a success story

## Introduction

As I read in the Groovy user mailinglist that some people complained missing information about how to integrate Groovy into Java applications, I decided to give one example of how we integrated Groovy as extension language for our graphical developer tool.

First a bit of background regarding the application to be scripted:

It is a little SDE GUI for starting different build targets for different components in a workarea. As we noticed that, depending on some individual tasks or different roles a developer takes in a team, extension and customisation would be a nice feature, we integrated Groovy as scripting language used to plug in individual features at the user site.

## The solution in overview

### *Extension points in the GUI:*

We created two "extension points" in the GUI: An empty "user" menu ready to be filled with items and an empty panel at the bottom of the GUI being able to be filled with e.g. custom buttons.

There is also a specific script output window, where the script can place messages or other textual output. The opening of this window is part of the API (see point 2).

### *Integration point:*

To keep the Groovy integration at one location, we created the class ScriptConnector, which is the adaptor between Groovy and the application.

It calls the Groovy engine, maintains the binding, provides some API methods to be called inside the Groovy script, leading to better separation which keeps the script clean from the application's intera.

BTW: One requirement was, that errors in the Groovy integration should not break the rest of the application, but should only affect the customised parts, so exceptions are caught and shown as 'warnings' in a dialog window.

### *One plugin script:*

The plugin feature is provided by one dedicated plugin script which is customisable/extensible by the user. He can use all features the Groovy language provides, so external scripts and programs can be integrated via this script.

## Coming to details

Let us have a look at the main class first, so you will see it all from startup on. Please be aware that the shown source is a simplified form of our productive code.

### *The application main class*

```
// The main application class
```

```

public class SDEGui {

    public static void main(String[] args) {
        SDEGui sdegui = new SDEGui();
        sdegui.startGui();
    }

    private void startGui() {
        ....
        final SDEGuiWindow window =
SDEGuiWindow.getInstance(); // Create the
whole "GUI"
        window.show();

        Workspace workarea =
SettingManager.getCurrentWorkspace(); //
Create the workarea, the object to be
scripted
        ....

        // starting the Groovy interpreter
        try {
            startScript(workarea, window);
        }
        catch (Exception e) {

JOptionPane.showMessageDialog(window,
"Exception in groovy script connection: " +
e.getMessage(),

"Groovy-error",

```

```

JOptionPane.WARNING_MESSAGE);
    }
}

// Starts the standard Groovy script to
setup additional (customised) gui elements
private void startScript(final Workspace
workarea, final SDEGuiWindow window) {

    ScriptConnector connector = new
ScriptConnector(workarea, window); //
instanciate the connector ...

connector.runGuiComponentScript("plugins.gro
ovy"); // ... and run the
plugin script

    window.show();
}
}

```

#### *The script connector*

Now let's look at the ScriptConnector, as this is the important place of Groovy integration:

```

....
import groovy.lang.Binding;
import groovy.util.GroovyScriptEngine;
import groovy.util.ResourceException;
import groovy.util.ScriptException;

```

....

```
public class ScriptConnector {
    Binding      binding; // The 'binding'
makes instances of the application objects
available as 'variables' in the script
    SDEGuiWindow window; // The main
application window, the GUI in general
    String[]     roots; // A list of
directories to search for Groovy scripts
(think of it as a PATH).
```

```
    public ScriptConnector(Workspace
workarea, SDEGuiWindow window) {
        roots      = new
String[]{System.getProperty("user.home"),
"." }; // The root list is filled with the
locations to be searched for the script
        Binding    scriptenv = new Binding();
// A new Binding is created ...

        scriptenv.setVariable("workarea",
workarea); // ... and filled with two
'variables': the workarea to work on
        scriptenv.setVariable("SDE", this);
// and the current ScriptConnector instance
as API provider.
        this.binding = scriptenv;
        this.window  = window;
    }
```

```
    // Method to show Groovy related
errors/warnings in a dialog window.
    public void showWarning(String message)
{

JOptionPane.showMessageDialog(window,
message, "Groovy-error",
JOptionPane.WARNING_MESSAGE);
    }
```

```
    // This is the main method called from
the application code to start the Groovy
integration
    public void runGuiComponentScript(String
filename) {
        GroovyScriptEngine gse = null;
        try {
            gse = new
GroovyScriptEngine(roots); //
instanciating the script engine ...
        } catch (IOException ioe) {
            ioe.printStackTrace();
            showWarning("I/O-Exception in
starting Groovy engine. Message is:\n"
                        + ioe.getMessage()
                        + "\n" +
prepareStackTrace(ioe));
        }

        if (gse != null) {
```

```

        try {
            gse.run(filename, binding);
// ... and running the specified script
        } catch (ResourceException re)
    {
        re.printStackTrace();

showWarning("ResourceException in calling
groovy script '" + filename +
            "' Message
is:\n" +re.getMessage()
            + "\n" +
prepareStackTrace(re));

        } catch (ScriptException se) {
            se.printStackTrace();

showWarning("ScriptException in calling
groovy script '" + filename +
            "' Message
is:\n" +se.getMessage()
            + "\n" +
prepareStackTrace(se));
        }
    }
}

// prepare a stacktrace to be shown in

```

an output window

```
private String
prepareStackTrace(Exception e) {
    Throwable exc = e;
    StringBuffer output = new
StringBuffer();
    collectTraces(exc, output);
    if (exc.getCause() != null) {
        exc = exc.getCause();
        output.append("caused by:\n");
        output.append(exc.getMessage());
        output.append("\n");
        collectTraces(exc, output);
    }
    return output.toString();
}

private void collectTraces(Throwable e,
StringBuffer output) {
    StackTraceElement[] trace =
e.getStackTrace();
    for (int i=0; i < trace.length; i++)
{
output.append(trace[i].toString());
    output.append("\n");
    }
}

// ----- API to be used
inside scripts -----
```

```
    // create a new dialog to display
textual output from running scripts
    public ScriptOutputDialog
newOutputDialog(String title, String
tabTitle) {
        return window.newOutputDialog(title,
tabTitle);
    }

    // get the panel instance prepared to
contain customised GUI elements, e.g.
buttons
    public DynamicPanel getDynpanel() {
        return window.getDynamicPanel();
    }

    // get the user menu instance to add
custom items and submenus to.
    public JMenu getUsermenu() {
        return window.getSDEUserMenu();
    }

    // create a process to run a shell
command in a given directory
    public Process exec(String command, File
inDir) {
        Process proc = null;
        try {
            proc =
Runtime.getRuntime().exec(command, null,
```

```

inDir);
        } catch (Exception e) {
            displayExecError(e.toString());
        }
        return proc;
    }

    // create a process to run a shell
command
    public Process exec(String command) {
        Process proc = null;
        try {
            proc =
Runtime.getRuntime().exec(command);
        } catch (Exception e) {
            displayExecError(e.toString());
        }
        return proc;
    }

    private void displayExecError(String
message) {
        ScriptOutputDialog win =
window.newOutputDialog("Groovy Error",
"Error during exec");
        win.addTabPane("error");
        win.println("error", message);
    }

```

```
    }  
  
}
```

**Customisation: The script plugin.groovy**

This is only an (senseless) example of how to create custom buttons and menu items, but in combination with the connector class it will give you an idea of how an application can be customised/scripted with Groovy as scripting language.

```
import groovy.swing.SwingBuilder  
  
// -- declare standard elements --  
allButtons = []  
allItems   = []  
builder    = new SwingBuilder()  
  
// USER CODE ----->  
  
// custom methods doing the different tasks  
  
def runDoSomething() {  
    def outp =  
SDE.newOutputDialog("Plugin-Window")  
    outp.show()  
    dir = workarea.workdir  
    Thread.start() {  
        outp.println ("=== ${dir}" )  
        def proc =  
SDE.exec("doSomething.bat", new  
File("${dir}") )
```

```

        outp.useInputStream(proc.in)
        proc.waitFor()
    }
    outp.println("end")
}
}

def showLogfile() {
    def outp =
SDE.newOutputDialog("Plugin-Window")
    outp.show()
    def logfile = new File("logfile.txt")
    logfile.eachLine{ line ->
        outp.println(line)
    }
}

// user gui elements

allButtons << builder.button( text: 'Do
Something', actionPerformed: {
runDoSomething() } )
allButtons << builder.button( text:
'showLogfile', actionPerformed: {
showLogfile() } )

allItems << builder.menuItem( text:
'TestItemOne', actionPerformed: { /* more
code, you know ... */ } )

```

```
allItems << builder.menuItem( text:
'TestItemTwo', actionPerformed: { /* ...
here too ... */ } )

// < -----    USER CODE

// -----    add custom gui elements to the
```

```
dynamic panel and user menu -----  
allButtons.each { SDE.dynpanel.add(it) }  
allItems.each { SDE.usermenu.add(it) }
```

I hope this spontaneous little article could give you a help in Groovy application integration and give a slight idea of what Groovy could do for you.

### Iterator Tricks

One useful Groovy iterator is the SyncIterator, based on Ruby's SyncEnumerator which can iterate across several data structures at the same time. It makes use of `DefaultTypeTransformation.asCollection()` to perform the same coercion that Groovy uses by default to implement methods such as `each()`, `collect()`, and the like...

```
import  
org.codehaus.groovy.runtime.typehandling.DefaultTypeTransformation;  
  
public class SyncIterator implements  
Iterator, Iterable{  
    private theobjects  
  
    SyncIterator(Object[] objects){  
        theobjects=objects.collect{  
            if (it instanceof Iterator) return  
/*from closure*/ it  
            else return /*from closure*/  
DefaultTypeTransformation.asCollection(it).i  
terator()  
        }  
    }  
}
```

```
boolean hasNext(){
    return theobjects.any{it.hasNext()}
}
Object next(){
    if (!hasNext()) throw new
java.util.NoSuchElementException()
    return theobjects.collect{
        try{
            return /*from closure*/
it.next()
        }catch(NoSuchElementException e){
            return /*from closure*/ null
        }
    }
}

Iterator iterator(){
    return this;
}

void remove(){
    throw new
UnsupportedOperationException("remove() not
supported")
}
```

```
}  
  
}
```

Another useful iterator is the Generator, which takes a closure and offers up the stuff it yields as an iterator, without having to keep all of the generated data around in memory. (It's based loosely on Ruby 1.9's Generator, which was also the challenge in [Ruby Quiz #66](#))

```
//JAVA CODE  
import java.util.concurrent.*;  
import java.lang.ref.*;  
import groovy.lang.Closure;  
import java.util.*;  
  
public class Generator<T> implements  
Iterator<T>, Iterable<T>{  
    Semaphore availSemaphore=new  
Semaphore(0);  
    Semaphore emptySemaphore=new  
Semaphore(1);  
  
    //the thread can push one value at at  
time into pushedValue  
    T pushedValue=null;  
  
    //pull value moves it from pushedValue to  
pulledValue  
    //until it is released by next()  
    T pulledValue=null;  
    boolean hasPulledValue=false;  
  
    Thread internalThread;
```

```
Generator(Closure closure){
    internalThread=new
GeneratorThread<T>(this,closure);
    internalThread.setDaemon(true);
    internalThread.start();
}

private void pullValue(){

availSemaphore.acquireUninterruptibly();
    pulledValue=pushedValue;
    pushedValue=null;
    hasPulledValue=true;
    emptySemaphore.release();
}

public boolean hasNext(){
    if (!hasPulledValue)
pullValue();
    return
emptySemaphore.availablePermits() != 2;
}

public T next(){
    if (!hasNext())
throw new NoSuchElementException("Closure
has no more values");
    T retval=pulledValue;
    hasPulledValue=false;
    return retval;
}
```

```

    }

    public void remove(){
        throw new
UnsupportedOperationException(
    "Remove is not supported on generators");
    }

    public Iterator<T> iterator(){
        return this;
    }

    public void finalize(){
        internalThread.interrupt();
    }

    static class GeneratorThread<T> extends
Thread{
        WeakReference<Generator<T>>
generatorRef;
        Closure closure;
        public GeneratorThread(Generator<T>
generator, Closure cl){
            generatorRef=new
WeakReference<Generator<T>>(generator);
            closure=cl;
        }

        public void run(){
            closure.call(new SaveClosure<T>(this));
            Generator generator=generatorRef.get();

```

```

    //NOTE: when the closure completes,
pullValue() will block forever
    //waiting for more available data. This
release() allows it to
    //get in one last time, and read a
variable indicating that the
    //thread has died and isn't producing any
more data. one final
    //pullValue() run will have
emptySemaphore==1 and
    //availSemaphore==1, and it will make
emptySemaphore==2 thus
    //indicating that the thread has died
if (generator!=null){
    generator.availSemaphore.release();
}
    //NOTE: if the generator has been garbage
collected, we don't care
    //about letting the generator pull a
termination condition.
    }
}

```

```

    static class SaveClosure<T> extends
Closure{
        WeakReference<Generator<T>>
generatorRef;
        Semaphore emptySemaphore;
        Semaphore availSemaphore;
        public SaveClosure(GeneratorThread<T>

```

```

gt){
    super(gt,null);
    generatorRef=gt.generatorRef;
    Generator<T> generator=generatorRef.get();
    if (generator!=null){

emptySemaphore=generator.emptySemaphore;

availSemaphore=generator.availSemaphore;
    }else{
        throw new GeneratorDisposedException();
    }
    }

    public void doCall(T value){
try{
    emptySemaphore.acquire();
}catch(InterruptedException e){
    throw new GeneratorDisposedException();
}
Generator<T> generator=generatorRef.get();
if (generator!=null){
    generator.pushedValue=value;
}else{
    throw new GeneratorDisposedException();
}
availSemaphore.release();
    }
}

/**

```

```
* A GeneratorDisposedException is used
to terminate the thread
* that was generating values, once the
Generator has been garbage
* collected.
*/
static public class
GeneratorDisposedException extends
```

```
RuntimeException{  
    }  
}
```

Note that Groovy doesn't have this concept built in. Its MethodClosure to Iterator conversion loads everything into an array all at once, which isn't a particularly good idea you may be running find() on it, and only need to generate the first item to find what you're looking for.

For example, consider the following use of a Generator:

```

//generates an infinite sequence of
Fibonacci numbers
def fibonacci(Closure yield){
    def a=0
    def b=1
    def temp
    while(true){
        yield(b)
        temp=a
        a=b
        b=a+temp
    }
}

//find the first Fibonacci number that's
evenly divisible by 20
println(new Generator(this.&fibonacci).find{
it % 20 == 0})

//BROKEN: the groovy runtime wants to run
fibonacci to termination loading values into
an array.
//this generates an out of memory error.
this.&fibonacci.find{it % 20 == 0}

```

As the Groovy runtime implements this now, you would exhaust all available ram when converting the MethodClosure to an iterator, before find() was ever called. With a Generator, values are only generated on demand.

Disadvantages:

- because of the use of threads, the generator may generate one more value than is actually needed before the garbage collector disposes of the generator.

- Exceptions thrown from the generator method are not propagated back to the caller (again, because the function is called on a different thread)

## Martin Fowler's closure examples in Groovy

[Martin Fowler](#) wrote an [article in his Bliki](#) on [Closures](#). He uses Ruby as demonstration language for closures. On this page (nearly) the same example is written in Groovy:

```
def managers(emps) {
  emps.findAll { e -> e.isManager() }
}
```

```
def highPaid(emps) {
  threshold = 150
  emps.findAll { e -> e.salary > threshold }
}
```

```
def paidMore(amount) {
  { e -> e.salary > amount }
}
```

```
def highPaid = paidMore(150)
println highPaid(emps[0])
```

```
new File(filename).withReader{ reader ->
doSomethingWith(reader) }
```

The whole example with class *Employee* (with an dispensible, but convenient, because boosting readability, *toString()* method), an example list with four employees and some explaining assertions (Dierk would call this *Inline Unittests*):

```
class Employee {
  def name, salary
  boolean manager
  String toString() { return name }
}

def emps = [new Employee(name:'Guillaume',
manager:true, salary:200),
  new Employee(name:'Graeme', manager:true,
salary:200),
  new Employee(name:'Dierk', manager:false,
salary:151),
  new Employee(name:'Bernd', manager:false,
salary:50)]

def managers(emps) {
  emps.findAll { e -> e.isManager() }
}

assert emps[0..1] == managers(emps) //
[Guillaume, Graeme]

def highPaid(emps) {
  threshold = 150
  emps.findAll { e -> e.salary > threshold }
}

assert emps[0..2] == highPaid(emps) //
[Guillaume, Graeme, Dierk]

def paidMore(amount) {
```

```
{ e -> e.salary > amount}
}
def highPaid = paidMore(150)

assert highPaid(emps[0]) // true
assert emps[0..2] == emps.findAll(highPaid)

def filename = 'test.txt'
new File(filename).withReader{ reader ->
doSomethingWith(reader) }

def readersText
def doSomethingWith(reader) { readersText =
reader.text }
```

```
assert new File(filename).text ==
readersText
```

## Other Examples

Some examples and snippets:

- [Scripts](#)
- [Unit tests](#)
- [GroovyAnt](#)
- [GroovySwing](#)
- [Make a builder](#)

## Parsing Groovy Doc Online

Contributing to online documentation website <http://gotapi.com>, we have to generate a xml document describing groovy objects and methods hierarchy and links to javadoc.

This is done with the following groovy script and the excellent HTML parser NekoHTML (see <http://people.apache.org/~andyc/neko/doc/index.html> )

```
package free.cafekiwi.gotapi;
import groovy.xml.MarkupBuilder
/**
 * @author Marc DEXET
 **/
class XmlGotApiGenerator {

    static void main(args) {
        def writer = new FileWriter( new
File('gotapi.xml'))
        def xml = new MarkupBuilder(writer)
        def pages = []

        def apidocRoot =
'http://groovy.codehaus.org/api'
```

```

def parser = new JavaDocParser();

def methodNameFilter = {
  anchor ->
    def name = anchor.'@name'
    name != null && name.indexOf('(') >=0
}

def packageFilter = {
  anchor ->
    def url = anchor.'@href'
    url != null && url.endsWith(
"/package-frame.html")
}

def classFilter = {
  anchor ->
    def url = anchor.'@href'
    url != null && ! url.contains('/')
}

def parseAndGet = {
  url, filter ->
    parser.parse( url
).depthFirst().A.findAll( filter )
}

parseAndGet("$apidocRoot/overview-frame.html
", packageFilter).each {

```

```

    def packBaseURL = it.'@href' -
"/package-frame.html"
    def packPage = new Page( title:
packBaseURL.replace('/', '.') ,
        type: 'package',
        url:
"${apidocRoot}/${it.'@href'}"
        );

    pages << packPage
    println "PackPAGE ${packPage}"

    parseAndGet( packPage.url,
classFilter).each {

        def url = it.'@href'
        def fullUrl =
"${apidocRoot}/${packBaseURL}/${url}"
        def fqN = url.replace('/', '.')-
".html"
        def parts = fqN.tokenize('.')

        def classPage = new Page( title: fqN ,
            type: 'class',
            url: fullUrl
        );

        packPage.children << classPage

        println "Class PAGE: ${classPage}"

```

```

    parseAndGet( classPage.url,
methodNameFilter ).each {
    def methodName = it.'@name'
    def methodPage = new Page( title:
methodName ,
                                type: 'method',
                                url:
classPage.url+'#' +methodName);
    classPage.children << methodPage
    println "Method PAGE: ${methodPage}"
    }
    }
    }
xml.pages() {
    pages.each{
    packIt ->
    xml.page(title: packIt.title , type:
packIt.type, url: packIt.url) {
    packIt.children.each {
    classIt ->
    xml.page(title: classIt.title , type:
classIt.type, url: classIt.url) {
    classIt.children.each {
    methodIt ->
    xml.page(title: methodIt.title ,
type: methodIt.type, url: methodIt.url)
    }
    }
    }
    }
    }
}

```

```

    }
  }
  println 'END'
}

class Page {
  def title
  def type
  def url
  def children = []
  public String toString() {
    "Titre: ${title} Type: ${type} URL:${url}
    \\n\\t
    ${children.collect{it.toString()}+'\\n\\t'
    }"}
  }
}

class JavaDocParser {
  def parser
  /* Constructor */
  JavaDocParser() {
    def nekoparser = new
    org.cyberneko.html.parsers.SAXParser()

    nekoparser.setFeature('http://xml.org/sax/fe
    atures/namespace', false)
    parser = new XmlParser(nekoparser);
  }
}

```

```
def parse(url) {
```

```
    return parser.parse(url)
}
}
```

## Plotting graphs with JFreeChart

Inspired by the excellent [BeanShell example](#), here is the same thing in Groovy (but displaying in a Swing window rather than writing to a file):

```
//require(groupId: 'jfree',
artifactId: 'jfreechart', version: '1.0.5')
//require(groupId: 'jfree',
artifactId: 'jcommon', version: '1.0.9')
import org.jfree.chart.ChartFactory
import org.jfree.chart.ChartPanel
import
org.jfree.data.general.DefaultPieDataset
import groovy.swing.SwingBuilder
import java.awt.*
import javax.swing.WindowConstants as WC

def piedataset = new DefaultPieDataset();
piedataset.with {
    setValue "Apr", 10
    setValue "May", 30
    setValue "June", 40
}

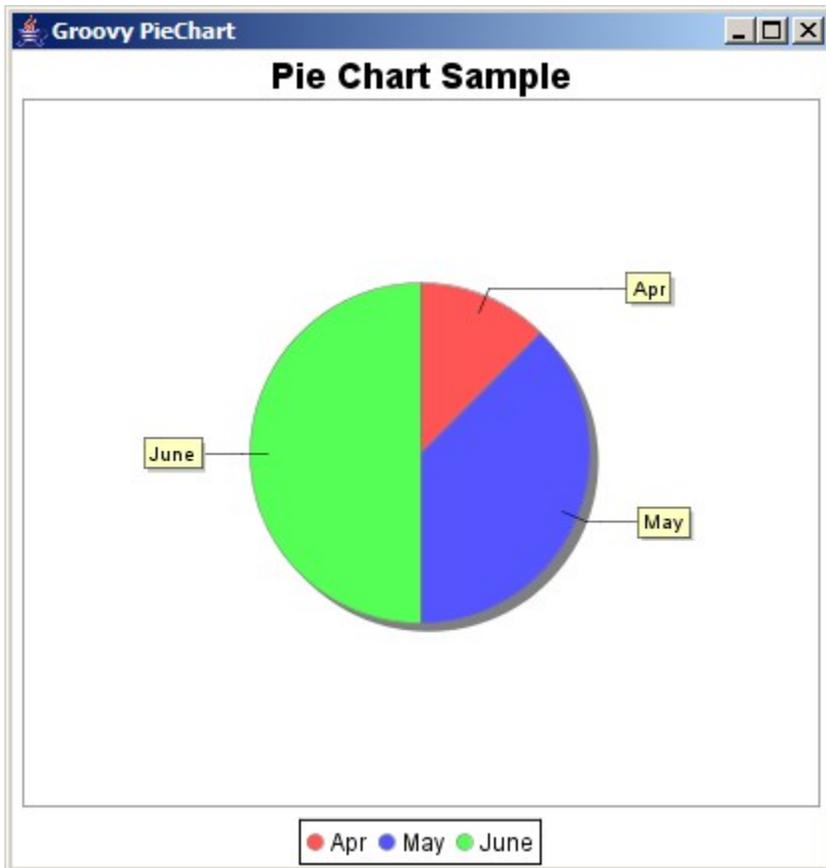
def options = [true, true, true]
```

```
def chart = ChartFactory.createPieChart("Pie
Chart Sample",
    piedataset, *options)
chart.backgroundPaint = Color.white
def swing = new SwingBuilder()
def frame = swing.frame(title:'Groovy
PieChart',

defaultCloseOperation:WC.EXIT_ON_CLOSE) {
    panel(id:'canvas') { widget(new
ChartPanel(chart)) }
```

```
}  
frame.pack()  
frame.show()
```

Here is the result:



Or if you prefer [Line Charts](#):

```
//require(groupId: 'jfree',  
artifactId: 'jfreechart', version: '1.0.5')  
//require(groupId: 'jfree',  
artifactId: 'jcommon', version: '1.0.9')  
import org.jfree.chart.ChartFactory  
import org.jfree.chart.ChartPanel  
import
```

```

org.jfree.data.category.DefaultCategoryDataset
et
import org.jfree.chart.plot.PlotOrientation
as Orientation
import groovy.swing.SwingBuilder
import javax.swing.WindowConstants as WC

def dataset = new DefaultCategoryDataset()
dataset.with{
    addValue 150, "no.1", "Jan"
    addValue 210, "no.1", "Feb"
    addValue 390, "no.1", "Mar"
    addValue 300, "no.2", "Jan"
    addValue 400, "no.2", "Feb"
    addValue 200, "no.2", "Mar"
}

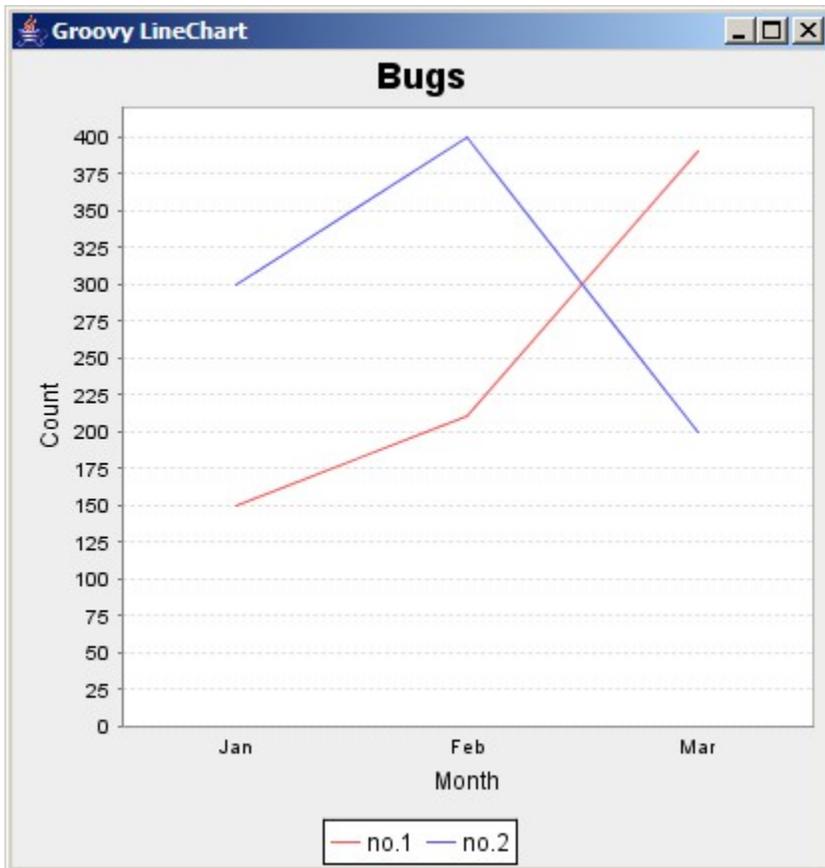
def labels = ["Bugs", "Month", "Count"]
def options = [true, true, true]
def chart =
ChartFactory.createLineChart(*labels,
dataset,
                Orientation.VERTICAL,
*options)
def swing = new SwingBuilder()
def frame = swing.frame(title:'Groovy
LineChart',

defaultCloseOperation:WC.EXIT_ON_CLOSE) {
    panel(id:'canvas') { widget(new
ChartPanel(chart)) }

```

```
}  
frame.pack()  
frame.show()
```

And the result looks like:



Additionally, you might have a look at the [GroovyChart library](#) which provides a dedicated builder interface for building charts.

## PoorMansMixins

**i** In order to prevent confusion with officially supported mixin mechanisms that may be added to Groovy in the future, I have used the term 'gynamo' instead of 'mixin'. It is in essence trying to provide the same functionality.

Gynamo is a lightweight mechanism for extending classes with discrete packages of functionality similar to mixins in languages like Ruby.

## Terms

To make things easier (hopefully) to understand, the following is a definition of terms used:

- Gynamee - A class that will be or has been Gynamized with a Gynamo
- Gynamo - A class that extends the gynamo.Gynamo abstract base class and specifies functionality that will be injected into to Gynamees
- Gynamize - The process of injecting the functionality specified by a Gynamo into a Gynamee

## Example

### An example Gynamee

```
class Square  
{  
  def size  
}
```

### An example Gynamo

```
import gynamo.*  
class AreaGynamo extends Gynamo  
{  
  def getArea = {->  
    return delegate.size * delegate.size  
  }  
}
```

## Gynamization

```
import gynamo.*

Gynamo.gynamize(Square, AreaGynamo)

def s = new Square(size: 10)
assert s.area == 100
```

Hopefully that illustrates the point.

### Download

- [Gynamo 1.0](#) (contains source)

You can also grab the buildable source from the [SVN repository](#).

Gynamo is released under the same license terms as Groovy.

### Documentation

All classes in the Gynamo library are in the package "gynamo".

Gynamos can contain fields (which will be public when injected) and methods (which Gynamos define as closures). All methods and properties are attached via [ExpandoMetaClass](#) so all the usual rules apply there.

Methods and properties are attached via the '=' operator so any existing methods or properties with a matching name will be overridden. This is something to watch out for.

### *Writing a Gynamo*

A Gynamo is any class that extends the gynamo.Gynamo class. Gynamo inheritance is not supported, all Gynamos must directly extend the Gynamo class.

Methods (as closures) or fields to be injected must be accessible via getters. Groovy makes this too easy by generating getters for any fields that you don't specify the visibility of.

```

import gynamo.*
class SomeGynamo extends Gynamo
{
  // Will become a public field on Gynamees
  def afield

  // Will become the method printSomething()
  on Gynamees.
  def printSomething = {->
    println "something"
  }
}

```

You can even add fields and methods statically to a class ...

```

import gynamo.*
class SomeGynamo extends Gynamo
{
  // Will become a public static field on the
  Gynamee's class
  static aStaticField

  // Will become the method aStaticMethod on
  the Gynamee's class.
  static aStaticMethod = {->
    // method code here
  }
}

```

The Gynamo class defines the static method `gynamize` which takes a Gynamee class and a Gynamo class and performs the injection.

```
import gynamo.*  
Gynamo.gynamize(GynameeClass, GynamoClass)
```

Or alternatively using categories ...

```
import gynamo.*  
  
use (Gynamo) {  
  GynameeClass.gynamize(GynamoClass)  
}
```

#### ***Pre and Post Gynamize hooks***

If your Gynamo needs to do anything before or after Gynamization of a class there are optional hooks for it

```

import gynamo.*
class SomeGynamo extends Gynamo
{
    void preGynamize(Class clazz)
    {
        // Could instrospect clazz and throw
        exceptions if there it shouldn't be
        Gynamized with this Gynamo
    }

    // ... some properties or methods to copy
    here ...

    void postGynamize(Class clazz)
    {
        // Could set any default values on clazz
        here or do anything else
    }
}

```

### *Gynamo Properties*

Sometimes you want to add bean like properties to a Gynamee. This is a little awkward but you can make it work using the provided `GynamoPropertyStorage` class. Another caveat is that you have to define the getter and/or setter for each such property. Here is an example ...

```

import gynamo.*
class SomeGynamo extends Gynamo
{
  def getObjectProperty = {->
    return
GynamoPropertyStorage[delegate].objectProper
ty
  }

  def setObjectProperty = {

GynamoPropertyStorage[delegate].objectProper
ty = it
  }

  static getStaticProperty = {->
    return
GynamoPropertyStorage[delegate].staticProper
ty
  }

  static setStaticProperty = {

GynamoPropertyStorage[delegate].staticProper
ty = it
  }
}

```

GynamoPropertyStorage is pretty simple. It contains a synchronized [WeakHashMap](#). Take a look at the [source](#) for details.

You would use the above like so ...

```
import gynamo.*

use(Gynamo) {
  SomeClass.gynamize(SomeGynamo)
}

def o1 = new SomeClass()
def o2 = new SomeClass()

o1.objectProperty = "o1"
o2.objectProperty = "o2"

assert o1.objectProperty == "o1"
assert o1.objectProperty == "o2"

// Can't use property notation with static
properties added dynamically due to groovy
bug
SomeClass.setStaticProperty("static")
assert SomeClass.getStaticProperty() ==
"static"
```

#### *Gynamo dependencies*

If you have a gynamo that depends on another Gynamee being injected then you can specify that with annotations

...

```

import gynamo.*
@GynamoDependency (SomeOtherGynamo)
class SomeGynamo extends Gynamo
{
    ...
}

```

The @GynamoDependency annotation takes a single parameter of type Class which must be another Gynamo. The specified Gynamo will be injected automatically **\*before\*** this Gynamo gets injected. Therefore, the methods and fields of the dependency are available on the Gynamnee at time of Gynamization.

If your Gynamo depends on multiple Gynamos, you can use the @GynamoDependencies annotation to specify a list of Gynamos that need to be injected before.

```

import gynamo.*
@GynamoDependencies ([SomeOtherGynamo,
SomeOtherOtherGynamo] )
class SomeGynamo extends Gynamo
{
    ...
}

```

**\*Note:\*** Be careful of circular dependencies. The dependency handling is pretty simple so if you do have a a circular dependency, your code will blow up with a StackOverflowError. If anyone has any suggestions on how this can be improved please let me know.

## Reading from a Blob

### Reading a Blob!

Note: tested with Oracle DB 9.x and a thin driver 9.x !

```

import groovy.sql.Sql

println "---- A working test of writing and
then reading a blob into an Oracle DB ---"
sql =
Sql.newInstance("jdbc:oracle:thin:@pignut:15
21:TESTBNDY", "userName",
                "paSSword",
                "oracle.jdbc.OracleDriver")

rowTest = sql.firstRow("select binarydata
from media where mediaid = 11122345")
blobTest = (oracle.sql.BLOB)rowTest[0]

byte_stream_test =
blobTest.getBinaryStream()
if( byte_stream_test == null ) { println
"Test: Received null stream!" }

byte[] byte_array_test = new byte[10]
int bytes_read_test =
byte_stream_test.read(byte_array_test)

print "Read $bytes_read_test bytes from the
blob!"

sql.connection.close()

```

Uses a table:

```
CREATE TABLE MEDIA
(
    MEDIAID NUMBER(22) NOT NULL,
    BINARYDATA BLOB NOT NULL
);
```

```
CREATE SEQUENCE SEQ_MEDIAID
INCREMENT BY 1
START WITH 100
ORDER;
```

Copying A Blob to A File!

```

...
    byte_stream_test =
blobTest.getBinaryStream()
    if( byte_stream_test == null ) { println
"Test: Received null stream!" }

    blob_size = blobTest.length()
    println "Blob size: $blob_size"

    byte[] byte_array_test = new
byte[blob_size]
    int bytes_read_test =
byte_stream_test.read(byte_array_test)
    println "Read $bytes_read_test from the
blob!"

    // Write to a file
    def fos= new
FileOutputStream('c:\\Jornada\\auxil\\output
.jpg')
    fos.write(byte_array_test);
    fos.close()
...

```

## Recipes For File

Groovy recipes for every day!

Today is the File Day !

*List my imported packages*

**Today:** I get a list of imported package into my old groovy scripts !

```

//Pattern for groovy script
def p = ~/.*\.groovy/
new File( 'd:\\scripts' ).eachFileMatch(p) {
    f ->
        // imports list
        def imports = []
        f.eachLine {
            // condition to detect an import
instruction
            ln -> if ( ln =~ '^import .*' ) {
                imports << "${ln - 'import '}"
            }
        }
        // print thmen
        if ( ! imports.empty ) {
            println f
            imports.each{ println "    $it" }
        }
    }
}

```

Output

```
D:\groovy getImports.groovy
d:\scripts\testCom.groovy
    org.codehaus.groovy.scriptom.ActiveXProxy
d:\scripts\testDurableSubscriber.groovy
    javax.jms.*;

org.apache.activemq.ActiveMQConnectionFactory
Y;

org.apache.activemq.command.ActiveMQTopic;
d:\scripts\testJmsBroker.groovy
    javax.jms.*;

org.apache.activemq.ActiveMQConnectionFactory
Y;

org.apache.activemq.command.ActiveMQTopic;
```

*Clean old files*

**today:** Oh jeez! I get on my SysAdmin's nerves, file System /data is full! .. I have to clean all these useless daily reports !

```

def yesterday = ( new Date() ).time -
1000*60*60*24

def cleanThem = { prefix ->
    new
File( '/data/waporwaresystem/reports' ).eachFi
leMatch( ~".*${prefix}.*xml" ) { f ->
    if ( f.lastModified() <= yesterday ) {
        f.delete()
    }
}
}

[ 'sales-carambar_', 'stock-scoubidou_',
'stock_freztagad_',
'coffee.vs.tea_stats_' ].each( cleanThem )

```

Saved! My SysAdmin loves me, it's sure 😊

*Recursively deleting files and directories.*

**Today** is a recursive day ! We have a bunch of files and directories to delete, let's go!

the Ant way

Don't forget this old java swiss knife...

```

new AntBuilder().delete(dir: "D:/tmp/test")

```

the Groovy Way

```

// Create a ref for closure
def delClos

// Define closure
delClos = { println "Dir
${it.canonicalPath}";
            it.eachDir( delClos );
            it.eachFile {
                println "File
${it.canonicalPath}";
                it.delete()
            }
            it.delete()
}

// Apply closure
delClos( new File("D:/tmp/test") )

```

An Even Groovier Way

Using the Groovy JDK extensions to [File](#)

```

new File("D:/tmp/test").deleteDir()

```

## Regex search for a class in the JVM runtime classpath

### Description

The below code searches through the classpath of a groovy script or a running groovy application and prints out classes which match the given regular expression. This is useful when you hit some variation of [JAR hell](#) and want to figure out which files contribute a specific class to your groovy classpath, be it a standalone script, a compiled joint java/groovy project, a grails project or something else.

Credits for the recursive classpath code go to 'blindgaenger' at:

The code has been modified to do a regex search as per below.

## Code

### test.groovy

```
import java.util.zip.*

def printClassPath(classLoader, pattern) {
    println "$classLoader"

    classLoader.URLs.findAll {
        it.toExternalForm().endsWith(".jar")
    }.each { url ->
        def f = new File(url.toURI())
        try {
            boolean firstTime = true
            // Iterate through all zip entries
            matching our regular expression
            // print out the jar url and the entry
            name as we go
            new ZipFile(f).entries().findAll {
                it.name =~ pattern }.each { ZipEntry entry
                ->
                    if (firstTime) {
                        println "    > $url"
                        firstTime = false
                    }
                    println "        > ${entry.name}"
                }
            } catch (Throwable e) {
```

```
        println "Error opening jar file ${f},
message: ${e.message}"
    }
}

// if we have a parent, make a recursive
call to search for the class there also
    if (classLoader.parent)
printClassPath(classLoader.parent, pattern)
}

// call the method with the current
classpath, look for classes matching 'http'
// (?i) switches on case insensitive regex
matching
// the / .. / is a groovy string syntax
suitable for regex handling
```

```
printClassPath this.class.classLoader,  
/(?i)http/
```

To try this out, paste the above code into 'test.groovy' and execute as follows:

```
$ groovy test.groovy  
groovy.lang.GroovyClassLoader$InnerLoader@6d  
79953c  
groovy.lang.GroovyClassLoader@7f033a6f  
org.codehaus.groovy.tools.RootLoader@2352544  
e  
>  
file:/usr/local/Cellar/groovy/2.1.1/libexec/  
lib/ant-1.8.4.jar  
>  
org/apache/tools/ant/taskdefs/condition/Http  
.class  
>  
file:/usr/local/Cellar/groovy/2.1.1/libexec/  
lib/groovy-servlet-2.1.1.jar  
>  
groovy/servlet/AbstractHttpServlet.class  
>  
file:/usr/local/Cellar/groovy/2.1.1/libexec/  
lib/ivy-2.2.0.jar  
>  
org/apache/ivy/plugins/resolver/util/ApacheH  
ttpURLLister.class  
>  
org/apache/ivy/util/url/BasicURLHandler$Http
```

**Status.class**

>

**org/apache/ivy/util/url/HttpClientHandler\$1.class**

>

**org/apache/ivy/util/url/HttpClientHandler\$HttpClientHelper.class**

>

**org/apache/ivy/util/url/HttpClientHandler\$HttpClientHelper2x.class**

>

**org/apache/ivy/util/url/HttpClientHandler\$HttpClientHelper3x.class**

>

**org/apache/ivy/util/url/HttpClientHandler\$IvyCredentialsProvider.class**

>

**org/apache/ivy/util/url/HttpClientHandler.class**

>

**file:/usr/local/Cellar/groovy/2.1.1/libexec/lib/jsp-api-2.0.jar**

> **javax/servlet/jsp/HttpJspPage.class**

>

**file:/usr/local/Cellar/groovy/2.1.1/libexec/lib/servlet-api-2.4.jar**

> **javax/servlet/http/**

> **javax/servlet/http/Cookie.class**

> **javax/servlet/http/HttpServlet.class**

>

**javax/servlet/http/HttpServletRequest.class**

```
>
javax/servlet/http/HttpServletRequestWrapper
.class
>
javax/servlet/http/HttpServletResponse.class
>
javax/servlet/http/HttpServletResponseWrappe
r.class
  > javax/servlet/http/HttpSession.class
  >
javax/servlet/http/HttpSessionActivationListe
ner.class
  >
javax/servlet/http/HttpSessionAttributeListe
ner.class
  >
javax/servlet/http/HttpSessionBindingEvent.c
lass
  >
javax/servlet/http/HttpSessionBindingListene
r.class
  >
javax/servlet/http/HttpSessionContext.class
  >
javax/servlet/http/HttpSessionEvent.class
  >
javax/servlet/http/HttpSessionListener.class
  > javax/servlet/http/HttpUtils.class
  >
javax/servlet/http/LocalStrings.properties
  >
```

```
javax/servlet/http/LocalStrings_es.properties
```

```
>
```

```
javax/servlet/http/LocalStrings_fr.properties
```

```
>
```

```
javax/servlet/http/LocalStrings_ja.properties
```

```
>
```

```
javax/servlet/http/NoBodyOutputStream.class
```

```
>
```

```
javax/servlet/http/NoBodyResponse.class
sun.misc.Launcher$AppClassLoader@1f3e8d89
sun.misc.Launcher$ExtClassLoader@43be2d65
```

Author: Matias Bjarland, Iteego Inc

**Search one or more jar files for a specified file or directory**

This script searches one or more jar or zip files for a specified file/directory.

```
import java.util.jar.*

/**
 * Searches entries (file name and
 * directory, if any) in one or more named
 * compressed (jar/zip) files, or in all
 * compressed files in a named directory.
 * To specify a search that includes special
 * characters, like a period, use
 * a backslash: \.xml
 */
if( args.size() < 2){
    println "Required parameters:
searchString filePath [filePath]\n"
    println "NOTE: filePath may be a
directory, which will search all jar/zip"
    println "    files in all subdirectories"
    return
}
def searchstr = args[0]

args[1..-1].each{searchInputFile(searchstr,
```

```

new File(it))}

def searchInputFile(text, inputFile){
  def filePattern = ~/*.*(jar|zip)$/
  if(inputFile.isDirectory()){
    inputFile.eachFileRecurse{
      if(!it.isDirectory() &&
it.getName() =~ filePattern)

searchCompressedFile(text, it)
    }
  }else{
    if(inputFile.getName() =~
filePattern){
      searchCompressedFile(text,
inputFile)
    }
  }
}

def searchCompressedFile(text, file){
  try{
    new
JarFile(file).entries().each{ entry ->
      if ( entry.name =~ text){
        println "\n$entry.name :
$file.canonicalPath"
      }
    }
  }catch(Throwable t){
    println "\nFailed to open

```

```
$file.canonicalPath: ${t.toString()}"  
    }  
}
```

## Simple file download from URL

This method takes a string containing the URL of the file to be downloaded. It will create a file in the current folder whose name is the same as the remote file.

```
def download(address)  
{  
    def file = new  
FileOutputStream(address.tokenize("/")[-1])  
    def out = new BufferedOutputStream(file)  
    out << new URL(address).openStream()  
    out.close()  
}
```

If proxy configuration is needed, call this:

```
System.properties.putAll(  
[ "http.proxyHost" : "proxy-host" ,  
"http.proxyPort" : "proxy-port" , "http.proxyUse  
rName" : "user-name" ,  
"http.proxyPassword" : "proxy-passwd" ] )
```

before the call.

Here is another approach using categories. The default left shift operators acts on text streams only. This category overrides the left shift operator making it a binary copy and uses the URL type as the source for the data.

```
package tests.io;

class FileBinaryCategoryTest extends
GroovyTestCase
{
    void testDownloadBinaryFile()
    {
        def file = new File("logo.gif")

        use (FileBinaryCategory)
        {
            file <<
"http://www.google.com/images/logo.gif".toUR
L()
        }

        assert file.length() > 0

        file.delete()
    }
}

class FileBinaryCategory
{
    def static leftShift(File a_file, URL
a_url)
    {
        def input
        def output
```

```

try
{
    input = a_url.openStream()
    output = new BufferedOutputStream(new
FileOutputStream(a_file))

    output << input
}
finally
{
    input?.close()
    output?.close()
}
}
}

```

A little less verbose FileBinaryCategory

```

class FileBinaryCategory{
    def static leftShift(File file, URL
url){
        url.withInputStream {is->
            file.withOutputStream {os->
                def bs = new
BufferedOutputStream( os )
                bs << is
            }
        }
    }
}

```

```

    }
  }
}

```

## Solving Sudoku

This page describes a Groovy solution for [ShortestSudokuSolver](#). Check the link for all the details, but basically the puzzle state is fed in as a String. Each line of the script must be no more than 80 characters in length.

The solution (184 characters plus 2 newlines):

```

def r(a){def i=a.indexOf(48);if(i<0)print a
else(('1'..'9')-(0..80).collect{j->
g={(int)it(i)==(int)it(j)};g{it/9}|g{it%9}|g
{it/27}&g{it%9/3}?a[j]:'0'}).each{
r(a[0..<i]+it+a[i+1..-1])}}

```

Notes:

- The script could be 25 characters shorter if Groovy supported integer division.
- The script executes more efficiently if you use logical operators '&&' and '||' instead of '&' and '|' because short-circuiting kicks in at the expense of 3 characters.
- The script would be 1 character longer if for clarity you wanted to use '0' instead of the first 48.
- The script would be 2 characters longer if you want to use `println` rather than `print` if you are fussy about the formatting
- To make the function stop as soon as it finds the first solution (proper puzzles will only have one solution), the part before the `else` becomes `{print a;System.exit(1)}`

Add the following line to the script to solve a partially complete puzzle (should take just a few seconds):

```

r
'2003751696392184575719643821524968733487529
16796831245900100500800007600400089001'

```

Alternatively, add the following line to the script to solve a puzzle from scratch (may take 30-60 minutes or more):

**r**

```
'2003700090092000070010040020500008000080009  
00006000040900100500800007600400089001'
```

The expected output is:

```
28437516963921845757196438215249687334875291  
6796831245967143528813527694425689731
```

Here is the more usual representation of the puzzle:

2			3	7				9
		9	2					7
		1			4			2
	5					8		
		8				9		
		6					4	
9			1			5		
8					7	6		
4				8	9			1

You can run it from the command-line by adding `;r args[0]` to the end of the script (saved in a file called `sudoku.groovy`) and then invoking:

```
> groovy sudoku.groovy  
20037000900920000700100400205000080000800090  
0006000040900100500800007600400089001
```

If you use `.groovy` as your script file extension, you can even leave off the extension as follows:

```
> groovy sudoku
20037000900920000700100400205000080000800090
0006000040900100500800007600400089001
```

A slightly longer version using a matcher and no inner closure (209 characters plus 2 newlines):

```
def r(a){def m=a=~'0';if(m.find()){int
i=m.start();(('1'..'9')-(0..80).collect{
j->int
q=j/9,r=i/9,u=q/3,v=r/3,w=j%9/3,x=i%9/3;q==r
||j%9==i%9||u==v&&w==x?
a[j]:'0'})}.each{r(a[0..<i]+it+a[i+1..-1])}}e
lse print a}
```

Or without the matcher (193 characters plus two newlines):

```
def r(a){int i=a.indexOf(48);if(i<0)print a
else(('1'..'9')-(0..80).collect{j->
int
q=j/9,r=i/9,u=q/3,v=r/3,w=j%9/3,x=i%9/3;q==r
||i%9==j%9||u==v&&w==x?a[j]:'0'
}).each{r(a[0..<i]+it+a[i+1..-1])}}
```

Also see [another version](#) for a more understandable (though much longer) algorithm - but it also does a lot more.

## StreamingMarkupBuilder and IMAP client example

The following groovy code example shows how get email messages from an **IMAP** server and transform the message structure and data into **XML**. It took me a ridiculously long time to get this working, so I thought it would be a good idea to share the code with others. ;-) Processing complex messages with javax.mail API is no fun and there is a lot in StreamingMarkupBuilder (and groovy) which simply seems to get over my head.

The code below has been structured so that the IMAP client implementation is nicely separated from the markup building code using something, which to me looks a lot like the *old school* call-back mechanism used in the C

language. Now, how groovy is that?

Any comments are welcome, of course!

```
//export
CLASSPATH=$HOME/src/javamail-1.4.3/lib/imap.jar:$HOME/src/javamail-1.4.3/lib/mailapi.jar

import groovy.xml.StreamingMarkupBuilder
import javax.activation.*
import javax.mail.*
import javax.mail.internet.*
import javax.mail.search.*
import java.util.Properties

class MailContentProcessor {
    def build(builder, String content) {
        builder.string(content)
    }

    def build(builder, Multipart mp) {
        builder.multipart {
            (1..mp.getCount()).each { n ->
                def p = mp.getBodyPart(n - 1)
                part {
                    contentType(p.getContentType())
                    def a
                    def d = p.getDisposition()
                    if (d) {
                        disposition(d)
                        a =
d.toLowerCase().equals(BodyPart.ATTACHMENT)
                    }
                }
            }
        }
    }
}
```

```

    }
    if (a) {
        fileName(p.getFileName())
        size(p.getSize())
        // Add code to save attachment if
desired...
    } else {
        build(builder, p.getContent())
    }
}
}
}
}

def build(builder, Message m) {

builder.message(id:m.getHeader("Message-ID")
[0]) {
    sender(m.sender)
    subject(m.subject)
    contentType(m.getContentType())
    build(builder, m.getContent())
}
}
} // class

class ImapSslClient {
    def hostname
    def username
    def password

```

```
def getStore = {
    // configure the jvm to use the jsse
security.
    java.security.Security.addProvider(new
com.sun.net.ssl.internal.ssl.Provider());

    Properties props = new Properties([

"mail.imap.socketFactory.class":
"javax.net.ssl.SSLSocketFactory",

"mail.imap.socketFactory.fallback":
"false",

"mail.imap.socketFactory.port": "993"
    ]
    )

    def cons = System.console()
    while (!password) {
        def m = "password for ${username} on
${hostname}: "
        password =
cons.readPassword(m).toString()
    }
    def name = new
URLName("imap://${username}:${password}@${ho
stname}")

Session.getInstance(props).getStore(name)
}
```

```

def processMessages(callBack,
callBackParam) {
  def inbox;
  def store;

  try {
    store = getStore()
    print "3."
    store.connect()
    print "2."
    inbox = store.getFolder("INBOX")
    print "1."
    inbox.open(Folder.READ_ONLY)
    def terms = new FlagTerm(new
Flags(Flags.Flag.DELETED), false)
    def l = inbox.search(terms)
    print "0."
    l.each() { m -> callBack(m,
callBackParam) }
  } finally {
    if(inbox) {
inbox.close(false) // do not expunge
    }
    if (store) {
store.close()
    }
    println "Done"
  }
}
} // class

```

```
if (args.length < 3) {
  println "Usage: mailtoxml <hostname>
<username> <output.xml>"
  System.exit(1)
}

def proc = new MailContentProcessor()

def myMessageCallback = { m, b ->
  print "."
  proc.build(b, m)
}

def builder = new StreamingMarkupBuilder()

new FileOutputStream(args[2]) <<
builder.bind { b ->
  messageList {
    new ImapSslClient(hostname: args[0],
      username:
args[1]).processMessages(myMessageCallback,
b)
  }
}
```

```
}
```

```
// Teppo Peltonen 2010
```

## SwingBuilder with custom widgets and observer pattern

In this tutorial you will learn how to:

- use custom widgets with SwingBuilder,
- implement observer pattern (also known as subject-observer pattern) in Swing & Groovy,
- use action listeners with SwingBuilder,
- build simple currency converter 😊.

Swing introduced modified **MVC** pattern (for more details see <http://java.sun.com/products/jfc/tsc/articles/architecture/>). To update model I'll use **observer pattern** (if you're not familiar with it, see [http://en.wikipedia.org/wiki/Observer\\_pattern](http://en.wikipedia.org/wiki/Observer_pattern)) which is directly supported in Java by `Observable` class and `Observer` interface in `java.util` package.

For the example let's choose currency converter application from **Java Generics and Collections** book by *Maurice Naftalin* and *Philip Wadler* (<http://www.oreilly.com/catalog/javagenerics/>) from chapter 9.5. It will show explicitly the benefits from using dynamic language like Groovy over static typed Java with generic observer pattern introduced in the book. If you would like to see implementation of generic observer pattern you can download examples from the book website and have a look.

OK. Let's start with the model:

```
class Model extends Observable {
    static CURRENCY = [ "USD", "EURO", "YEN"
]

    private Map rates = new HashMap()
    private long value

    void initialize(initialRates) {
        (0..CURRENCY.size() - 1).each {
            setRate(CURRENCY[it],
initialRates[it])
        }
    }
}
```

```
// setting rate for currency
void setRate(currency, f) {
    rates.put(currency, f);
    setChanged();
    notifyObservers(currency);
}

// setting new value for currency
void setValue(currency, double newValue)
{
    value = Math.round(newValue /
rates[currency]);
    setChanged();
    notifyObservers(null);
}

// getter for value for particular
currency
def getValue(currency) {
```

```
        value * rates[currency]
    }
}
```

The converter model allows conversions over three different currencies. As you can see it extends Observable class to provide Model class observable behaviour (for more details see [java.util.Observable](#)).

Now let's create two custom widgets for displaying rate and value.

```
class RateView extends JTextField implements
Observer {
    private Model model;
    private currency;

    public void setModel(Model model) {
        this.model?.removeObserver(this)
        this.model = model
        model.addObserver(this)
    }

    public void update(Observable o, Object
currency) {
        if (this.currency == currency)
            text = String.format("%15.2f",
model.rates[currency])
    }
}

class ValueView extends JTextField
implements Observer {
    private Model model
    private currency
```

```
public void setModel(Model model) {
    this.model?.removeObserver(this)
    this.model = model
    model.addObserver(this)
}

public void update(Observable o, Object
currency) {
    if (currency == null ||
this.currency == currency)
        text = String.format("%15.2f",
```

```
model.getValue(this.currency));
    }
}
```

These classes extends `JTextField` to hold model and currency which is representing. They also implement `Observer` interface to be noticed when the model is changed. As you can see in update method there are not class casts required although it receives `Object`, because as dynamic nature of Groovy. Also in `setModel` method safe dereferencing is shown to protect from throwing `NullPointerException` when initially model is `null`.

Now let's put it all together.

```
swing = new SwingBuilder()
model = new Model()

frame = swing.frame(title: "Groovy
SwingBuilder MVC Demo", layout: new
GridLayout(4, 3), size: [300, 150],
    defaultCloseOperation:
WindowConstants.EXIT_ON_CLOSE) {

    label("currency")
    label("rate")
    label("value")

    for (c in Model.CURRENCY) {
        label(c)
        widget(new RateView(), model:
model, currency: c,
            action:
swing.action(closure: { event ->
event.source.model.setRate(event.source.curr
ency, event.source.text.toDouble());
```

```
        )))  
        widget(new ValueView(), model:  
model, currency: c, action:  
swing.action(closure: {event ->  
  
event.source.model.setValue(event.source.cur  
rency, event.source.text.toDouble());  
        )))  
    }  
}
```

```
frame.show()  
model.initialize([1.0, 0.83, 0.56]);
```

Frame is constructed by using `swing.frame()`. To frame there are provided `title`, `layout`, `defaultCloseOperation`, `size` properties. You can think of it like creating a new instance of `JFrame` and invoking methods `setTitle()`, `setLayout()`, `setDefaultCloseOperation()`, `setSize()`. Then 12 components are added to frame:

- JLabel components using `label("label's text")`,
- RateView components using `widget()` builder method and setting `model`, `currency` attributes,
- ValueView components in the same way like RateView.

When new rate or value is entered all action listeners of that component are noticed with `actionPerformed()` method ([java.awt.ActionListener](#)). To construct classes which implements `ActionListener` interface `SwingBuilder` provides **action()** builder method. One of this method's attributes is closure when we are able to provide our closure with application logic. The closure argument has `ActionEvent` type.

Download the source code of the example: [SwingBuilderObserver.groovy](#)

## Tomcat tools

You ever want to run multiple instance of tomcats with the same install base, but tired of copy and setup `server.xml` file? Attached are couple Groovy scripts that create new server instances, and a tool to quickly setup a new Groovlet webapp.

### NewTomcatInstance.groovy

Assume you have installed Tomcat6 or Tomcat5 in `/opt/tomcat` directory.

```
ztoy:~/s zemian$ ./NewTomcatInstance.groovy
/opt/tomcat mytomcat
Create new instance dir
/opt/tomcat/instances/mytomcat
Create server.xml file with httpPort 8081,
httpPort 8081
Create tomcat-user.xml file with manager
role user.
Create catalina-mytomcat.sh file
Create ROOT webapp
Create ROOT webapp web.xml
Create ROOT webapp index.jsp
Done.
To Start Server:
/opt/tomcat/bin/catalina-mytomcat.sh start
To View Browser: http://127.0.0.1:8081
```

Run it again to create another instance with mytomcat2, and it should configure to port 8082 and so on...

Each new server instance will contain a ROOT webapp that lists all other webapps for quick links.

Also, the server instance is configured with Tomcat manager webapp enabled with a user: admin. If you are in this stage, you ought to know where to look for your password. 😊

#### Security

As stated in Tomcat documentation, enabling Tomcat manager is considered a potential security hole as it enables authenticated users to have total control over your webapps. It's enabled in this script for the sake of easy setup and quick management.

To remove a previous installed instance

```
ztoy:~/s zemian$  
./delete_tomcat_instance.groovy /opt/tomcat  
mytomcat
```

### NewWebapp.groovy

This script will create a new webapp directory structure with all the Groovlet setup ready.

```
ztoy:~/s zemian$ ./NewWebapp.groovy  
/opt/tomcat/instances/mytomcat/webapps  
mywebapp  
Create webapp  
Copy groovy jar to lib  
    [copy] Copying 1 file to  
/opt/tomcat/instances/mytomcat/webapps/myweb  
app/WEB-INF/lib  
Create webapp web.xml  
Create webapp index.gsp  
Create webapp mysqlreport.groovy  
Create webapp mysqlreport.gsp  
Done.
```

Start your server and you have a webapp ready to go!

The mysqlreport.groovy is a updated version of Andrew Glover's <http://www.ibm.com/developerworks/java/library/j-pg03155/>

And you will need mysql jdbc driver jar copy into mywebapp/WEB-INF/lib to work.

### Unsign Jar Files (Recursively)

```
ant = new AntBuilder();  
tmpDir = "tmpDir"
```

```
new File(args[0]).eachFileRecurse({file->  
    if(file.name.endsWith(".jar")) {  
        ant.sequential {  
            mkdir(dir:tmpDir)        }  
    }  
})
```

```
    echo "Unsigning file: $file"
    unjar(src:file, dest:tmpDir)
    delete {
        fileset(dir:tmpDir, includes:"META-INF/*.DSA,META-INF/*.SF,META-INF/*.RSA")
    }
    jar(destFile:file, baseDir:tmpDir)
    delete(dir:tmpDir)
}
}
})
```

## Using Groovy with Boost

[Boost](#) is a framework for working with Java. This page describes how to use the IoC part of Boost called [Spider](#) with Groovy.

Spider has a very interface oriented view of the world. If you are into *Interface oriented design* (e.g. Ken Pugh's book) and use fine-grained interfaces, you will feel right at home. So first, let's create a Java interface:

```
// Java
package foo;

public interface MeTwo {
    void reportStuff();
}
```

and a corresponding implementation:

```
// Java
package foo;

public class DefaultMeTwo implements MeTwo {
    public void reportStuff() {
        System.out.println("Reporting
stuff");
    }
}
```

We could have just as easily used Groovy. As an example, here's another interface (this time Groovy):

```
package foo

interface IGetInjected {
    void doStuff()
}
```

And its Groovy implementation:

```
package foo

class DefaultIGetInjected implements
IGetInjected {
    void doStuff() {
        println("Doing stuff")
    }
}
```

Now, let's create a class that uses these implementations.

```
package foo

import
au.net.netstorm.boost.spider.api.entry.Go

@PackageScope
class PleaseDontGo implements Go {
    IGetInjected yippee
    MeTwo rippa

    void go(String[] args) {
        yippee.doStuff()
        rippa.reportStuff()
    }
}
```

The `go()` method is the Spider's answer to Java's `main()` method, i.e. it is the entry point for IoC managed webs of classes. Also, `@PackageScope` is an AST macro. The Spider does field injection based on package scoped fields. However, Groovy normally turns packages scoped fields into properties and we don't want that here. The AST macro tells the Groovy compiler to leave the fields in the `PleaseDontGo` class alone and not try to turn them into properties. We could have placed the annotation individually on fields if we preferred. Placing it at the class level makes it apply to all package scoped fields.

Now, let's bootstrap the IoC container so that we can run our application:

```
package foo
```

```
import
```

```
au.net.netstorm.boost.spider.ioc.BoostWeb
```

```
import
```

```
au.net.netstorm.boost.spider.api.entry.SpiderMain
```

```
def main = new SpiderMain(BoostWeb)
```

```
main.main(PleaseDontGo)
```

Here, `BoostWeb` provides some nice defaults for us to use. Running this gives:

```
Doing stuff
```

```
Reporting stuff
```

We can also override the Web. For instance, we can supply our own web:

```

package foo

import
au.net.netstorm.boost.spider.api.config.web.
Web
import
au.net.netstorm.boost.spider.api.config.wire
.Wire

class TangledWeb implements Web {
    @PackageScope Wire wire
    void web() {
        wire.ref({ println 'Reporting
stuff2' } as MeTwo).to(MeTwo)
    }
}

```

Which tells the IoC framework to use the supplied closure whenever it needs to inject a `MeTwo` implementation. Note again the use of the `PackageScope` AST macro to override Groovy's default property behavior. With this in place, our main script becomes:

```

def main = new SpiderMain(BoostWeb,
TangledWeb)
main.main(PleaseDontGo)

```

And the output will be:

```

Doing stuff
Reporting stuff2

```

You can also use Boost's [testing facilities](#). An example (Foo and Person classes not shown) to give you a flavor of these kinds of tests:



```
@PackageScope
class StringJoinerTest extends
GroovyLifecycleTestCase implements
HasFixtures, InjectableSubject,
InjectableTest, LazyFields {
    StringJoiner subject
    private Person pa, pb
    Foo fooA, fooB
    Foo foolMock, foo2Mock

    void fixtures() {
        pa = new Person(first:'f1',
last:'l1')
        pb = new Person(first:'f2',
last:'l2')
    }

    void testLookup() {
        expect.oneCall(foolMock, "FOO",
"toUpperCase")
        expect.oneCall(foo2Mock, "BAR",
"toUpperCase")
        assert subject.join(pa.first,
pb.last) == "f1l2"
        assert subject.join(fooA.bar,
fooB.bar) == "barbar"
        assert subject.joinUpper(foolMock,
foo2Mock) == "FOOBAR"
    }
}
```

Here, `fooA`, `fooB` and `subject` will be auto created with default values, `foo1Mock` and `foo2Mock` will be auto created as mocks, `pa` and `pb` are set using the `fixtures()` method. `GroovyLifecycleTestCase` is a slight variation of `LifecycleTestCase` which is built-in to Boost. The built-in one is a little too opinionated about the methods Groovy adds under the covers to its classes. The result is very compact test code albeit looking a little magical until you get used to this style of testing.

## Using Hibernate with Groovy

This example uses Hibernate with Groovy using Groovy Grapes.

```
package demo

import javax.persistence.*
import org.hibernate.cfg.*

// javax.transaction jta.jar added manually
to ivy repo
@Grapes([
    @Grab(group='org.hibernate',
module='hibernate-annotations',
version='3.4.0.GA'),
    @Grab(group='org.slf4j',
module='slf4j-simple', version='1.4.2'),
    @Grab(group='hsqldb', module='hsqldb',
version='1.8.0.7'),
    @Grab(group='javassist',
module='javassist', version='3.4.GA'),
])
@Entity class Book {
    @Id @GeneratedValue(strategy =
 GenerationType.AUTO)
    public Long id
    public String author
    public String title
```

```
String toString() { "$title by $author"
}
}

def hibProps = [
    "hibernate.dialect":
"org.hibernate.dialect.HSQLDialect",
    "hibernate.connection.driver_class":
"org.hsqldb.jdbcDriver",
    "hibernate.connection.url":
"jdbc:hsqldb:mem:demodb",
    "hibernate.connection.username": "sa",
    "hibernate.connection.password": "",
    "hibernate.connection.pool_size": "1",
    "hibernate.connection.autocommit":
"true",
    "hibernate.cache.provider_class":
"org.hibernate.cache.NoCacheProvider",
    "hibernate.hbm2ddl.auto": "create-drop",
    "hibernate.show_sql": "true",
    "hibernate.transaction.factory_class":
"org.hibernate.transaction.JDBCTransactionFactory",

    "hibernate.current_session_context_class":
"thread"
]

def configureHibernate(props) {
    def config = new
AnnotationConfiguration()
```

```
    props.each { k, v ->
config.setProperty(k, v) }
    config.addAnnotatedClass(Book)
    return config
}

def factory =
configureHibernate(hibProps).buildSessionFactory()

// store some books
def session = factory.currentSession
def tx = session.beginTransaction()
session.save(new Book(author:'Dierk et al',
title:'Groovy in Action'))
session.save(new Book(author:'Craig',
title:'Spring in Action'))
tx.commit()

// find some books
session = factory.currentSession
tx = session.beginTransaction()
def books = session.createQuery("from
Book").list()
```

```
println 'Found ' + books.size() + ' books:'
books.each { println it }
tx.commit()
```

Which produces this output (logging not shown):

```
Found 2 books:
Groovy in Action by Dierk et al
Spring in Action by Craig
```

We can get a little more sophisticated and show a One-to-One relationship as follows:

```
@Entity class Book {
    @Id @GeneratedValue(strategy =
GenerationType.AUTO)
    public Long id
    @OneToOne
    public Author author
    public String title
    String toString() { "$title by
$author.name" }
}

@Entity class Author {
    @Id @GeneratedValue(strategy =
GenerationType.AUTO)
    public Long id
    public String name
}
```

Where the `configureHibernate` method needs an additional line:

```
...
config.addAnnotatedClass(Book) //
existing
config.addAnnotatedClass(Author) // new
...
```

And creation would become:

```
// store some books
def session = factory.currentSession
def tx = session.beginTransaction()
def a1 = new Author(name:'Dierk et al')
session.save(a1)
session.save(new Book(author:a1,
title:'Groovy in Action'))
def a2 = new Author(name:'Craig')
session.save(a2)
session.save(new Book(author:a2,
title:'Spring in Action'))
tx.commit()
```

We can get even fancier with a one-to-many association (one way to do it):

```
@Entity class Book {
    @Id @GeneratedValue(strategy =
GenerationType.AUTO)
    public Long id
    @OneToMany(cascade=CascadeType.ALL)
    public Set<Author> authors
    public String title
    String toString() { "$title by
${authors.name.join(', ')}" }
}

@Entity class Author {
    @Id @GeneratedValue(strategy =
GenerationType.AUTO)
    public Long id
    public String name
}
```

With creation code like this:

```

// store some books
def session = factory.currentSession
def tx = session.beginTransaction()
def names1 = ['Dierk', 'Guillaume', 'Jon',
'Andy', 'Paul']
def names2 = ['Craig']
def storeAuthors = { names, hib ->
    Set result = []
    names.each{ def a = new Author(name:it);
hib.save(a); result << a }
    result
}
def authors1 = storeAuthors(names1, session)
def authors2 = storeAuthors(names2, session)
session.save(new Book(authors:authors1,
title:'Groovy in Action'))
session.save(new Book(authors:authors2,
title:'Spring in Action'))
tx.commit()

```

Which has output like this (logging not shown):

```

Found 2 books:
Groovy in Action by Paul, Jon, Guillaume,
Dierk, Andy
Spring in Action by Craig

```

Using JGoodies Animation with Groovy

```

// require(url:'jgoodies.com',

```

```
jar:'animation', version:'1.2.0')
// require(url:'jgoodies.com', jar:'forms',
version:'1.1.0')

// Based on the jgoodies animation tutorial
class:
//
com.jgoodies.animation.tutorial.intro.BasicT
extLabelIntro

import static java.awt.Color.*
import java.awt.Font
import javax.swing.*

import static
com.jgoodies.animation.Animations.*
import static
com.jgoodies.animation.animations.BasicTextA
nimation.defaultFade as fadeText
import static
com.jgoodies.animation.animations.BasicTextA
nimations.defaultFade as fadeTexts
import com.jgoodies.animation.*
import
com.jgoodies.animation.components.BasicTextL
abel
import
com.jgoodies.forms.builder.PanelBuilder
import
com.jgoodies.forms.layout.CellConstraints
import com.jgoodies.forms.layout.FormLayout
```

```

class AnimateAction extends AbstractAction {
    def animation
    void
    actionPerformed(java.awt.event.ActionEvent
e) {
        animation.addAnimationListener([
            animationStarted: { ae ->
enabled = false },
            animationStopped: { ae ->
enabled = true }
        ] as AnimationListener)
        new Animator(animation, 30
/*fps*/).start()
    }
}

def buildPanel(labels) {
    def layout = new
FlowLayout('fill:pref:grow',
'fill:pref:grow, p, p')
    def builder = new PanelBuilder(layout)
    def cc = new CellConstraints()
    builder.add(buildPreviewPanel(labels),
cc.xy(1, 1))
    builder.addSeparator(' ',
cc.xy(1, 2))
    builder.add(buildToolsPanel(labels),
cc.xy(1, 3))
    return builder.panel
}

```

```

def buildPreviewPanel(labels) {
    def layout = new
    BorderLayout('fill:200dlu:grow',
    'fill:100dlu:grow')
    def panel = new JPanel(layout)
    def cc = new CellConstraints()
    panel.background = WHITE
    panel.add(labels[0], cc.xy(1, 1))
    panel.add(labels[1], cc.xy(1, 1))
    return panel
}

def buildToolsPanel(labels) {
    def layout = new
    BorderLayout('right:pref:grow', 'pref')
    def builder = new PanelBuilder(layout)
    builder.setDefaultDialogBorder()
    def cc = new CellConstraints()
    def action = new
    AnimateAction(animation:createAnimation(labels))
    action.putValue('Name', 'Animate')
    builder.add(new JButton(action),
    cc.xy(1, 1))
    return builder.panel
}

def createAnimation(labels) {
    Animation[] animations = [
        pause(1000),

```

```

        fadeText(labels[0], 2500, 'Welcome
To', DARK_GRAY),
        pause(1000),
        fadeText(labels[0], 3000, 'JGoodies
Animation', DARK_GRAY),
        pause(1000),
        fadeTexts(labels[0], labels[1],
2000, -100,
                'An open source framework|for
time-based|' +
                'real-time animations|in Java
and Groovy.',
                DARK_GRAY),
        pause(1000),
        fadeTexts(labels[0], labels[1],
3000, 500, 'Main Features:', DARK_GRAY),
        pause(1000),
        fadeTexts(labels[0], labels[1],
1750, 0,
                'Seamless|flexible|and powerful
integration|' +
                'with Java and Groovy.|Small
library size.',
                DARK_GRAY),
        pause(1500)
    ]
    sequential(animations)
}

def buildLabel(font) {
    def label = new BasicTextLabel(' ')

```

```

    label.font = font
    label.opaque = false
    return label
}

def centerOnScreen(component) {
    def paneSize = component.size
    def screenSize =
component.toolkit.screenSize
    int x = (screenSize.width -
paneSize.width) / 2
    int y = (screenSize.height -
paneSize.height) * 0.45
    component.setLocation(x, y)
}

def font = new Font('Tahoma', Font.BOLD, 18)
def label1 = buildLabel(font)
def label2 = buildLabel(font)
def frame = new JFrame()
frame.title = 'Groovy/JGoodies Animation
Demo'
frame.defaultCloseOperation =
WindowConstants.EXIT_ON_CLOSE
def panel = buildPanel([label1, label2])
frame.contentPane.add(panel)
frame.pack()
centerOnScreen(frame)
frame.visible = true

```

[JScience](#) is a comprehensive Java library for the scientific community encompassing various areas of science including math, physics, sociology, biology, astronomy, economics.

```
/*
// require(name:'JScience', version:'3.2',
url:'http://jscience.org/jscience-vm14.jar')
import
org.jscience.mathematics.numbers.Complex
import static
org.jscience.mathematics.numbers.Complex.I
import static
org.jscience.mathematics.numbers.Complex.val
ueOf as c
import
org.jscience.mathematics.functions.Polynomial
import static
org.jscience.mathematics.functions.Polynomial.valueOf as p
import
org.jscience.mathematics.functions.Variable
*/

// require(name:'JScience', version:'4.3.1',
url:'http://jscience.org/jscience-4.3.1-bin.
zip')
import
org.jscience.mathematics.number.Complex
import static
org.jscience.mathematics.number.Complex.I
import static
```

```

org.jscience.mathematics.number.Complex.valueOf as c
import
org.jscience.mathematics.function.Polynomial
import static
org.jscience.mathematics.function.Polynomial
.valueOf as p
import
org.jscience.mathematics.function.Variable

// Defines two local variables (x, y).
def varX = new Variable.Local<Complex>("x")
def varY = new Variable.Local<Complex>("y")

use (JScienceCategory) {
  def ONE = Complex.ONE
  def TWO = c(2, 0)
  //  $f(x) = ix^2 + 2x + 1$ 
  def x = p(ONE, varX)
  def fx = I * x ** 2 + TWO * x + ONE
  println fx
  println fx ** 2
  println fx.differentiate(varX)
  println fx.integrate(varY)
  println fx.compose(fx)

  // Calculates expression.
  varX.set(c(2, 3))
  println fx.evaluate()
}

```

```
class JScienceCategory {
    static power(Polynomial p, int n) {
        p.pow(n)
    }
    static multiply(Complex c, Polynomial p)
{
        p.times(c)
    }
    static multiply(Polynomial p, Complex c)
{
```

```

        p.times(c)
    }
}

```

Running this script yields:

```

[0.0 + 1.0i]x2 + [2.0 + 0.0i]x + [1.0 +
0.0i]
[-1.0 + 0.0i]x4 + [0.0 + 4.0i]x3 + [4.0 +
2.0i]x2 + [4.0 + 0.0i]x + [1.0 + 0.0i]
[0.0 + 2.0i]x + [2.0 + 0.0i]
[0.0 + 1.0i]x2y + [2.0 + 0.0i]xy + [1.0 +
0.0i]y
[0.0 - 1.0i]x4 + [-4.0 + 0.0i]x3 + [-2.0 +
6.0i]x2 + [4.0 + 4.0i]x + [3.0 + 1.0i]
-7.0 + 1.0i

```

## Using MarkupBuilder for Agile XML creation

Two principles of Agile development are *DRY* (don't repeat yourself) and *merciless refactoring*. Thanks to excellent IDE support it isn't too hard to apply these principles to coding Java and Groovy but it's a bit harder with XML.

The good news is that Groovy's Builder notation can help. Whether you are trying to refactor your Ant build file(s) or manage a family of related XML files (e.g. XML request and response files for testing Web Services) you will find that you can make great advances in managing your XML files using builder patterns.

**Scenario:** Consider we have a program to track the sales of copies of [GINA](#) 😊. Books leave a warehouse in trucks. Trucks contain big boxes which are sent off to various countries. The big boxes contain smaller boxes which travel to different states and cities around the world. These boxes may also contain smaller boxes as required. Eventually some of the boxes contain just books. Either GINA or some potential upcoming Groovy titles. Suppose the delivery system produces XML files containing the items in each truck. We are responsible for writing the system which does some fancy reporting.

If we are a vigilant tester, we will have a family of test files which allow us to test the many possible kinds of XML files we need to deal with. Instead of having to manage a directory full of files which would be hard to maintain if the delivery system changed, we decide to use Groovy to generate the XML files we need. Here is our first attempt:

```

import groovy.xml.MarkupBuilder

```

```
def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
xml.truck(id:'ABC123') {
    box(country:'Australia') {
        box(country:'Australia',
state:'QLD') {
            book(title:'Groovy in Action',
author:'Dierk König et al')
            book(title:'Groovy in Action',
author:'Dierk König et al')
            book(title:'Groovy for VBA Macro
writers')
        }
        box(country:'Australia',
state:'NSW') {
            box(country:'Australia',
state:'NSW', city:'Sydney') {
                book(title:'Groovy in
Action', author:'Dierk König et al')
                book(title:'Groovy for COBOL
Programmers')
            }
            box(country:'Australia',
state:'NSW', suburb:'Albury') {
                book(title:'Groovy in
Action', author:'Dierk König et al')
                book(title:'Groovy for
Fortran Programmers')
            }
        }
    }
}
```

```

    }
    box(country: 'USA') {
        box(country: 'USA', state: 'CA') {
            book(title: 'Groovy in Action',
author: 'Dierk König et al')
            book(title: 'Groovy for Ruby
programmers')
        }
    }
    box(country: 'Germany') {
        box(country: 'Germany',
city: 'Berlin') {
            book(title: 'Groovy in Action',
author: 'Dierk König et al')
            book(title: 'Groovy for PHP
Programmers')
        }
    }
    box(country: 'UK') {
        box(country: 'UK', city: 'London') {
            book(title: 'Groovy in Action',
author: 'Dierk König et al')
            book(title: 'Groovy for Haskel
Programmers')
        }
    }
}

println writer.toString()

```

There is quite a lot of replication in this file. Lets refactor out two helper methods `standardBook1` and `standardB`

ook2 to remove some of the duplication. We now have something like this:

```
import groovy.xml.MarkupBuilder

// standard book
def standardBook1(builder) {
    builder.book(title:'Groovy in Action',
    author:'Dierk König et al') }
// other standard books
def standardBook2(builder, audience) {
    builder.book(title:"Groovy for ${audience}")
}
def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
xml.truck(id:'ABC123') {
    box(country:'Australia') {
        box(country:'Australia',
state:'QLD') {
            standardBook1(xml)
            standardBook1(xml)
            standardBook2(xml, 'VBA Macro
writers')
        }
        box(country:'Australia',
state:'NSW') {
            box(country:'Australia',
state:'NSW', city:'Sydney') {
                standardBook1(xml)
                standardBook2(xml, 'COBOL
Programmers')
            }
        }
    }
}
```

```
        box(country: 'Australia',
state: 'NSW', suburb: 'Albury') {
            standardBook1(xml)
            standardBook2(xml, 'Fortran
Programmers')
        }
    }
}
box(country: 'USA') {
    box(country: 'USA', state: 'CA') {
        standardBook1(xml)
        standardBook2(xml, 'Ruby
Programmers')
    }
}
box(country: 'Germany') {
    box(country: 'Germany',
city: 'Berlin') {
        standardBook1(xml)
        standardBook2(xml, 'PHP
Programmers')
    }
}
box(country: 'UK') {
    box(country: 'UK', city: 'London') {
        standardBook1(xml)
        standardBook2(xml, 'Haskel
Programmers')
    }
}
```

```
}
```

```
println writer.toString()
```

Next, let's refactor out a few more methods to end up with the following:

```
import groovy.xml.MarkupBuilder

// define standard book and version allowing
multiple copies
def standardBook1(builder) {
  builder.book(title:'Groovy in Action',
  author:'Dierk König et al') }
def standardBook1(builder, copies) {
  (0..<copies).each{ standardBook1(builder) }
}
// another standard book
def standardBook2(builder, audience) {
  builder.book(title:"Groovy for ${audience}")
}
// define standard box
def standardBox1(builder, args) {
  def other = args.findAll{it.key !=
  'audience'}
  builder.box(other) {
  standardBook1(builder);
  standardBook2(builder, args['audience']) }
}
// define standard country box
def standardBox2(builder, args) {
  builder.box(country:args['country']) {
```

```

        if (args.containsKey('language')) {
            args.put('audience',
args['language'] + ' programmers')
            args.remove('language')
        }
        standardBox1(builder, args)
    } }

```

```

def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
xml.truck(id:'ABC123') {
    box(country:'Australia') {
        box(country:'Australia',
state:'QLD') {
            standardBook1(xml, 2)
            standardBook2(xml, 'VBA Macro
writers')
        }
        box(country:'Australia',
state:'NSW') {
            [Sydney:'COBOL',
Albury:'Fortran'].each{ city, language ->
                standardBox1(xml,
[country:'Australia', state:'NSW',
                    city:"${city}",
audience:"${language} Programmers"])
            } } }
        standardBox2(xml, [country:'USA',
state:'CA', language:'Ruby'])
        standardBox2(xml, [country:'Germany',

```

```
city:'Berlin', language:'PHP'])
  standardBox2(xml, [country:'UK',
city:'London', language:'Haskel'])
```

```
}
```

```
println writer.toString()
```

This is better. If the format of our XML changes, we will minimise the changes required in our builder code. Similarly, if we need to produce multiple XML files, we can add some for loops, closures or if statements to generate all the files from one or a small number of source files.

We could extract out some of our code into a helper method and the code would become:

```
import groovy.xml.MarkupBuilder

def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
def standard = new
StandardBookDefinitions(xml)
xml.truck(id: 'ABC123') {
    box(country: 'Australia') {
        box(country: 'Australia',
state: 'QLD') {
            standard.book1(2)
            standard.book2('VBA Macro
writers')
        }
        box(country: 'Australia',
state: 'NSW') {
            [Sydney: 'COBOL',
Albury: 'Fortran'].each{ city, language ->

standard.box1(country: 'Australia',
state: 'NSW',
                    city: "${city}",
```

```
audience:"${language} Programmers")
  }   }   }
  standard.box2(country:'USA', state:'CA',
language:'Ruby')
  standard.box2(country:'Germany',
city:'Berlin', language:'PHP')
  standard.box2(country:'UK',
city:'London', language:'Haskell')
```

```
}
```

```
println writer.toString()
```

So far we have just produced the one XML file. It would make sense to use similar techniques to produce all the XML files we need. We can take this in several directions at this point including using GStrings, using database contents to help generate the content or making use of templates.

We won't look at any of these, instead we will just augment the previous example just a little more. First we will slightly expand our helper class. Here is the result:

## StandardBookDefinitions.groovy

```
import groovy.xml.MarkupBuilder

class StandardBookDefinitions {
    private def builder
    StandardBookDefinitions(builder) {
        this.builder = builder
    }
    def removeKey(args, key) { return
args.findAll{it.key != key} }
    // define standard book and version
allowing multiple copies
    def book1() { builder.book(title:'Groovy
in Action', author:'Dierk König et al') }
    def book1(copies) { (0..<copies).each{
book1() } }
    // another standard book
    def book2(audience) {
builder.book(title:"Groovy for ${audience}")
}
    // define standard box
```

```

def box1(args) {
  def other = removeKey(args,
'audience')
  builder.box(other) { book1();
book2(args['audience']) }
}
// define standard country box
def box2(args) {
  builder.box(country:args['country'])
{
  if
(args.containsKey('language')) {
    args.put('audience',
args['language'] + ' programmers')
    args.remove('language')
  }
  box1(args)
} }
// define deep box
def box3(args) {
  def depth = args['depth']
  def other = removeKey(args, 'depth')
  if (depth > 1) {
    builder.box(other) {
      other.put('depth', depth -
1)
      box3(other)
    }
  } else {
    box2(other)
  }
} }

```

```
// define deep box
def box4(args) {
  builder.box(country: 'South Africa'){
    (0..<args[ 'number' ]).each{
book1() }
}
```

```
    }  
  }  
}
```

And now we will use this helper class to generate a family of related XML files. For illustrative purposes, we will just print out the generated files rather than actually store the files.

```
import groovy.xml.MarkupBuilder  
  
def writer = new StringWriter()  
xml = new MarkupBuilder(writer)  
standard = new StandardBookDefinitions(xml)  
def shortCountry = 'UK'  
def longCountry = 'The United Kingdom of  
Great Britain and Northern Ireland'  
def shortState = 'CA'  
def longState = 'The State of Rhode Island  
and Providence Plantations'  
def countryForState = 'USA'  
  
def generateWorldOrEuropeXml(world) {  
  xml.truck(id: 'ABC123') {  
    if (world) {  
      box(country: 'Australia') {  
        box(country: 'Australia',  
state: 'QLD') {  
          standard.book1(2)  
          standard.book2('VBA  
Macro writers')  
        }  
        box(country: 'Australia',
```

```

state: 'NSW') {
    [Sydney: 'COBOL',
Albury: 'Fortran'].each{ city, language ->

standard.box1(country: 'Australia',
state: 'NSW',
                                city: "${city}",
audience: "${language} Programmers")
    } }
    standard.box2(country: 'USA',
state: 'CA', language: 'Ruby')
    }
    standard.box2(country: 'Germany',
city: 'Berlin', language: 'PHP')
    standard.box2(country: 'UK',
city: 'London', language: 'Haskel')
    }
}

def generateSpecialSizeXml(depth, number) {
    xml.truck(id: 'DEF123') {
        standard.box3(country: 'UK',
city: 'London', language: 'Haskel',
depth: depth)
        standard.box4(country: 'UK',
city: 'London', language: 'Haskel',
number: number)
        box(country: 'UK') {} // empty box
    }
}

```

```
def generateSpecialNamesXml(country, state)
{
    xml.truck(id: 'GHI123') {
        if (state) {
            box(country:country,
state:state){ standard.book1() }
        } else {
            box(country:country){
standard.book1() }
        }
    }
}
```

```
generateWorldOrEuropeXml(true)
generateWorldOrEuropeXml(false)
generateSpecialSizeXml(10, 10)
generateSpecialNamesXml(shortCountry, '')
generateSpecialNamesXml(longCountry, '')
generateSpecialNamesXml(countryForState,
shortState)
```

```
generateSpecialNamesXml (countryForState,  
longState)  
println writer.toString()
```

This will be much more maintainable over time than a directory full of hand-crafted XML files.

Here is what will be produced:

```
<truck id='ABC123'>  
  <box country='Australia'>  
    <box state='QLD' country='Australia'>  
      <book title='Groovy in Action'  
author='Dierk König et al' />  
      <book title='Groovy in Action'  
author='Dierk König et al' />  
      <book title='Groovy for VBA Macro  
writers' />  
    </box>  
    <box state='NSW' country='Australia'>  
      <box city='Albury' state='NSW'  
country='Australia'>  
        <book title='Groovy in Action'  
author='Dierk König et al' />  
        <book title='Groovy for Fortran  
Programmers' />  
      </box>  
      <box city='Sydney' state='NSW'  
country='Australia'>  
        <book title='Groovy in Action'  
author='Dierk König et al' />  
        <book title='Groovy for COBOL  
Programmers' />
```

```
    </box>
  </box>
</box>
<box country='USA'>
  <box state='CA' country='USA'>
    <book title='Groovy in Action'
author='Dierk König et al' />
    <book title='Groovy for Ruby
programmers' />
  </box>
</box>
<box country='Germany'>
  <box city='Berlin' country='Germany'>
    <book title='Groovy in Action'
author='Dierk König et al' />
    <book title='Groovy for PHP
programmers' />
  </box>
</box>
<box country='UK'>
  <box city='London' country='UK'>
    <book title='Groovy in Action'
author='Dierk König et al' />
    <book title='Groovy for Haskel
programmers' />
  </box>
</box>
</truck>
<truck id='ABC123'>
  <box country='Germany'>
    <box city='Berlin' country='Germany'>
```

```
<book title='Groovy in Action'
author='Dierk König et al' />
  <book title='Groovy for PHP
programmers' />
    </box>
  </box>
  <box country='UK'>
    <box city='London' country='UK'>
      <book title='Groovy in Action'
author='Dierk König et al' />
      <book title='Groovy for Haskell
programmers' />
        </box>
      </box>
    </truck>
    <truck id='DEF123'>
      <box language='Haskell' city='London'
country='UK'>
        <box language='Haskell' city='London'
country='UK'>
          <box language='Haskell' city='London'
country='UK'>
            <box language='Haskell'
city='London' country='UK'>
              <box language='Haskell'
city='London' country='UK'>
                <box language='Haskell'
city='London' country='UK'>
                  <box language='Haskell'
```



```
author='Dierk König et al' />
  <book title='Groovy in Action'
author='Dierk König et al' />
  <book title='Groovy in Action'
author='Dierk König et al' />
  <book title='Groovy in Action'
author='Dierk König et al' />
  <book title='Groovy in Action'
author='Dierk König et al' />
  <book title='Groovy in Action'
author='Dierk König et al' />
</box>
<box country='UK' />
</truck>
<truck id='GHI123'>
  <box country='UK'>
    <book title='Groovy in Action'
author='Dierk König et al' />
  </box>
</truck>
<truck id='GHI123'>
  <box country='The United Kingdom of Great
Britain and Northern Ireland'>
    <book title='Groovy in Action'
author='Dierk König et al' />
  </box>
</truck>
<truck id='GHI123'>
  <box state='CA' country='USA'>
    <book title='Groovy in Action'
author='Dierk König et al' />
```

```
</box>
</truck>
<truck id='GHI123'>
  <box state='The State of Rhode Island and
  Providence Plantations' country='USA'>
    <book title='Groovy in Action'
```

```
author='Dierk König et al' />
  </box>
</truck>
```

Things to be careful about when using markup builders is not to overlap variables you currently have in scope. The following is a good example

```
import groovy.xml.MarkupBuilder

def book = "MyBook"

def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
xml.shelf() {
  book(name:"Fight Club") {

  }
}

println writer.toString()
```

When run this will actually get the error

```
aught: groovy.lang.MissingMethodException:  
No signature of method:  
java.lang.String.call() is applicable for  
argument types: (java.util.LinkedHashMap,  
HelloWorld$_run_closure1_closure2) values:  
{"name": "Fight Club"},
```

This is because we have a variable above called book, then we are trying to create an element called book using the markup. Markups will always honor for variables/method names in scope first before assuming something should be interpreted as markup. But wait, we want a variable called book AND we want to create an xml element called book! No problem, use delegate variable.

```
import groovy.xml.MarkupBuilder  
  
def book = "MyBook"  
  
def writer = new StringWriter()  
def xml = new MarkupBuilder(writer)  
xml.shelf() {  
    delegate.book(name: "Fight Club") {  
  
    }  
}  
  
println writer.toString()
```

**Using Regex Patterns to find word phone numbers**

Running this script:

```

import java.util.regex.Pattern

keypadMap = [0:"", 1:"", 2:"abc", 3:"def",
4:"ghi",
              5:"jkl", 6:"mno", 7:"pqrs",
8:"tuv", 9:"wxyz"]
dictionary = new File("words")
regexBuilder = { digit -> '[' +
keypadMap[digit.toInteger()] + ']' }
patternBuilder = { number ->
    Pattern.compile('^' +
number.collect(regexBuilder).join() + '$') }

def matchWords(phNumber) {
    def p = patternBuilder(phNumber)
    def matches = []
    dictionary.eachLine{ line -> if
(p.matcher(line).matches()) matches << line
}
    if (matches)
        println "Matches for $phNumber: " +
matches.join(' ')
    else
        println "No matches found for
$phNumber"
}

def matchPairs(phNumber) {
    def p1 = patternBuilder(phNumber[0..2])
    def p2 = patternBuilder(phNumber[3..6])
}

```

```

def first = [], second = []
dictionary.eachLine { line ->
    if (p1.matcher(line).matches())
first << line
    if (p2.matcher(line).matches())
second << line
}
if (first && second)
    println "$phNumber matches word
pairs: " +
        [first,
second].combinations().collect{ it.join '-'
}.join(', ')
    else
        println "Couldn't find word pair for
$phNumber"
}

[ '2345678', '2455466', '2276228',
'6662342', '2476642',
'2742742', '7338433', '7678243',
'3742253', '7264253',
'7683353', '7684453', '7837453',
'7378453', '3749953',
'7877363', '2668363', '5367273',
'7377473', '7877673',
'7278873', '2646283', '3377483'
].each{ matchWords it }
println '-----'

```

```
[ '8765432', '9689474', '2896673',  
'6847687', '9687283', '2283278'  
].each{ matchPairs it }
```

against the attached [words](#) file yields these results:

```
No matches found for 2345678  
Matches for 2455466: billion  
Matches for 2276228: acrobat  
Matches for 6662342: monadic nomadic  
Matches for 2476642: chromic chronic  
Matches for 2742742: aphasia aphasic  
Matches for 7338433: refugee seethed  
Matches for 7678243: portage postage  
Matches for 3742253: dribble friable  
Matches for 7264253: panicle sanicle  
Matches for 7683353: snuffle souffle  
Matches for 7684453: smuggle snuggle  
Matches for 7837453: puerile sterile  
Matches for 7378453: reptile servile  
Matches for 3749953: drizzle frizzle  
Matches for 7877363: supreme suspend  
Matches for 2668363: contend convene  
Matches for 5367273: jeopard leopard  
Matches for 7377473: reprise respire  
Matches for 7877673: purpose suppose  
Matches for 7278873: pasture rapture  
Matches for 2646283: animate cognate  
Matches for 3377483: deprive despite ferrite  
-----  
Couldn't find word pair for 8765432
```

9689474 matches word pairs: you-wish

2896673 matches word pairs: buy-more,  
buy-nose

6847687 matches word pairs: mug-pour,  
mug-soup, mug-sour

9687283 matches word pairs: you-pate,  
you-pave, you-rate, you-rave, you-save,  
you-scud

2283278 matches word pairs: act-dart,

```
bat-dart, cat-dart, act-east, bat-east,  
cat-east, act-fast, bat-fast, cat-fast
```

## Using the Delegating Meta Class

This is an example of how to replace a MetaClass to adjust the default behavior. Each groovy object has a metaClass that is used to manage the dynamic nature of the language. This class intercepts calls to groovy objects to ensure that the appropriate grooviness can be added. For example, when an object is constructed, the MetaClass's invokeConstructor() is called. One feature of the invokeConstructor allows us to create groovy objects using a map argument to set the properties of the object (new X([prop1: value1, prop2: value2])).

These solutions perform complete replacements, where as a more scoped solution can be found at [Using the Proxy Meta Class](#).

### InvokeHelper Solution

This technique installs the meta class at runtime using the InvokerHelper to gain access to the registry which allows us to change the meta class instance that is in use. The behaviour for objects created before the change depends on whether they are Java or Groovy objects. Groovy objects "cache" their metaclass, so once they're created, changing the registry has no effect. However, Java objects don't have a cache, so accesses to them always start with a trip to the registry, meaning any registry changes affect existing and new instances alike.

This sample code overrides the invokeMethod method to augment the behavior but there are other options that you can choose from like set and getAttribute, invokeStaticMethod and invokeConstructor. The complete list can be found in the Groovy's source release in "src/main/groovy/lang/DelegatingMetaClass.java".

```
import  
org.codehaus.groovy.runtime.InvokerHelper  
  
class DelegatingMetaClassInvokeHelperTest  
extends GroovyTestCase  
{  
    void testReplaceMetaClass()  
    {  
        /*  
         * Constructing first instance  
before meta class replacment  
         * is made.  
         */  
        def firstInstance = "first"
```

```

        assertEquals "first",
firstInstance.toString()

        def myMetaClass = new
MyDelegatingMetaClass(String.class)

InvokerHelper.metaRegistry.setMetaClass(Stri
ng.class, myMetaClass)

        /*
         * Constructing second instance
after meta class replacment
         * is made.
         */
        def secondInstance = "second"

        /*
         * Since we are replacing a meta
class at the class level
         * we are changing the behavior of
the first and second
         * instance of the string.
         */
        assertEquals "changed first",
firstInstance.toString()
        assertEquals "changed second",
secondInstance.toString()
    }
}

class MyDelegatingMetaClass extends

```

```
groovy.lang.DelegatingMetaClass
{
    MyDelegatingMetaClass(final Class
aclass)
    {
        super(aclass);
        initialize()
    }

    public Object invokeMethod(Object
a_object, String a_methodName, Object[]
a_arguments)
    {
        return "changed
${super.invokeMethod(a_object, a_methodName,
```

```
a_arguments) } "  
    }  
}
```

### Package Name Convention Solution

This second solution offers a more consistent augmentation of existing classes. There are no risks of unpredictable results from methods. The idea is that any package.class can have a custom meta class loaded at startup time by placing it into a well known package with a well known name.

```
groovy.runtime.metaclass.[YOURPACKAGE].[YOURCLASS]MetaClass
```

So your class Foo in package "bar" could have a custom meta class FooMetaClass in package "groovy.runtime.metaclass.bar".

The following example shows how we can change the behavior of the String class. Firstly the custom meta class, similar to the implementation above except that it needs a MetaClassRegistry argument in its constructor.

```
package groovy.runtime.metaclass.java.lang

class StringMetaClass extends
groovy.lang.DelegatingMetaClass
{
    StringMetaClass(MetaClass delegate)
    {
        super(delegate);
    }

    public Object invokeMethod(Object
a_object, String a_methodName, Object[]
a_arguments)
    {
        return "changed
${super.invokeMethod(a_object, a_methodName,
a_arguments)}"
    }
}
```

The actual class that uses the enhanced features is now very simple. Notice that there are no extra imports or any work with the meta class. The mere package and name of the class tells the groovy runtime to use the custom meta class.

```

class DelegatingMetaClassPackageImpliedTest
extends GroovyTestCase
{
    void testReplaceMetaClass()
    {
        assertEquals "changed hello world",
"hello world".toString()
    }
}

```

### Precedence

So what would happen if you used both techniques. Assume that the package convention class exists in your class path and you create and set another meta class. The answer is that the last setMetaClass that you did applies to the usages of all instance of the effected type.

### Using the Eclipse Modeling Framework (EMF)

Access to models generated with the [Eclipse Modeling Framework](#) is easy in Groovy with the help of Groovy Beans and GPath.

As an example we use the [EMF tutorial](#). This model contains three classes Book, Writer and Library and an enumeration BookCategory. From this model EMF generates Java code. There are two special classes: a package class and a factory class. We need the package class for reading the model. We have to instantiate it and load a file with data as following.

```

LibraryPackage.eINSTANCE
def resource = new
XMIResourceImpl(URI.createURI('hardboiled.li
brary'))
resource.load(null)
Library library = (Library)
resource.contents\[0\]           // get the
root element

```

Now we are able to query the model using standard Groovy. For example

```
for ( book in library.books ) {  
    println book.author.name + ', ' +  
    book.title + ', ' + book.category + ', ' +  
    book.pages  
}
```

prints out all books. We can print out all the books with less than 240 pages with the following statement.

```
println library.books.grep { it.pages < 240  
}.title.join(", ")
```

All the objects in an EMF model are constructed with methods from a factory (LibraryFactory in this example). The [Groovy EMF Builder](#) provides an interface for constructing models and model elements. It takes an EMF factory as an argument. In the following snippet three objects are created in the model: a Library, a Writer and a Book.

```

def builder = new EMFBuilder(LibraryFactory)
def writer
def library = builder.Library( name :
'Hardboiled Library') {
    writers {
        writer = Writer( name : 'Raymond
Chandler')
    }
    books {
        Book ( title: 'The Big Sleep',
pages: 234, category:
BookCategory.MYSTERY_LITERAL, author:
writer)
    }
}
}

```

The braces indicate the containment relationships writers and books of the class Library. See the homepage of the [Groovy EMF Builder](#) for further details.

## Using the Eclipse UML2 Framework

The [Eclipse UML2](#) project provides an implementation of the [Unified Modeling Language \(UML\) 2.1](#) metamodel in Java. The fact that the underlying technology is the [Eclipse Modeling Framework \(EMF\)](#) makes it possible to use the [EMFBuilder](#) with the UML2 metamodel.

The [introductory article](#) by Ken Hussey explains how to create UML2 models with the Eclipse editor and how to create them programmatically with Java code. Here we will use Groovy and the [EMFBuilder](#).

We use the UMLFactory for the EMFBuilder.

```

def builder = new EMFBuilder(UMLFactory)

```

We create a model as the root node and then we create two primitive types and store them in Groovy variables because we have to reference them later on.

```
def epo2Model = builder.Model(name: 'epo2')
{
  packagedElement {
    def intPrimitiveType = PrimitiveType(name:
'int')
    def stringPrimitiveType =
PrimitiveType(name: 'String')
```

We define an enumeration OrderStatus with three literals.

```
def orderStatusEnumeration =
Enumeration(name: 'OrderStatus') {
  ownedLiteral {
    EnumerationLiteral(name: 'Pending')
    EnumerationLiteral(name: 'Back Order')
    EnumerationLiteral(name: 'Complete')
  }
}
```

The following code snippet shows the definition of the classes Address and USAddress. All the attributes are defined as a Property. The primitive types stringPrimitiveType and intPrimitiveType defined above are used. The class USAddress is a subclass of the abstract class Address. This is expressed with the Generalization object.

```

def addressClass = Class(name: 'Address'
,isAbstract: true) {
  ownedAttribute {
    Property(name: 'name', type:
stringPrimitiveType, lower: 0, upper: 1)
    Property(name: 'country', type:
stringPrimitiveType, lower: 0, upper: 1)
  }
}
def usAddressClass = Class(name:
'USAddress') {
  generalization {
    Generalization(general: addressClass)
  }
  ownedAttribute {
    Property(name: 'street', type:
stringPrimitiveType, lower: 0, upper: 1)
    Property(name: 'city', type:
stringPrimitiveType, lower: 0, upper: 1)
    Property(name: 'state', type:
stringPrimitiveType, lower: 0, upper: 1)
    Property(name: 'zip', type:
intPrimitiveType, lower: 0, upper: 1)
  }
}

```

Compare this code to the original code in the article! The code is much more concise and is a direct representation of the UML2 diagram. No auxiliary methods are needed.

See the homepage of the [UML2 Builder](#) for further details.

**Using the Proxy Meta Class**

The [Using the Delegating Meta Class](#) page talks about techniques to change the behavior of existing classes by replacing the meta class in use. However, the Delegating Meta Class effects all loaded classes.

The Proxy Meta Class allows us to replace the meta class in use by a class but in a well defined scope. This technique would be more appropriate for temporary behavior replacements such as

This example shows how we can create a proxy meta class for the String class and intercept its method invocations.

```
import
org.codehaus.groovy.runtime.InvokerHelper

class ProxyMetaClassTest extends
GroovyTestCase
{
    void testProxyMetaClass()
    {
        def proxy =
ProxyMetaClass.getInstance(String.class);
        proxy.interceptor = new
MyInterceptor()

        def text = "hello world"

        assertEquals "hello world",
text.toString()

        proxy.use {
            assertEquals "changed hello
world", text.toString()
        }

        assertEquals "hello world",
text.toString()
    }
}
```

```
}
```

```
class MyInterceptor implements
```

```
groovy.lang.Interceptor
```

```
{
```

```
    Object beforeInvoke(Object a_object,  
String a_methodName, Object[] a_arguments)
```

```
    {
```

```
    }
```

```
    boolean doInvoke()
```

```
    {
```

```
        return true
```

```
    }
```

```
    Object afterInvoke(Object a_object,  
String a_methodName, Object[] a_arguments,  
Object a_result)
```

```
    {
```

```
        return "changed ${a_result}"
    }
}
```

For more detail on using the ProxyMetaClass, see <http://groovy.codehaus.org/Using+the+Proxy+Meta+Class+in+depth>.

## Windows Look And Feel for groovyConsole

Add the following lines:

```
set JAVA_OPTS=%JAVA_OPTS%
-Dswing.aatext=true
set JAVA_OPTS=%JAVA_OPTS%
-Dswing.defaultlaf=com.sun.java.swing.plaf.w
indows.WindowsLookAndFeel
```

just after the line

```
if "%JAVA_OPTS%" == "" set
JAVA_OPTS="-Xmx128m"
```

in `groovyConsole.bat`. The first line turns on text antialiasing and the second turns on the Windows Look And Feel. You can do the same in all the other `.bat` files as well. You may find the antialiasing doesn't improve the appearance of the fonts very much, in which case remove it.

## WrappingGroovyScript

### Abstract

The script `GroovyWrapper` described on this page, creates a self-executable jar from a groovy script.

### Description

The script `GroovyWrapper` allows you to distribute your groovy script as a standalone self-executable jar.

The generated jar contains the compiled groovy script, and the groovy embeddable jar. Moreover the jar's main-class is set to the compiled groovy script.

Having this self-executable jar, you can start the groovy script, like `java -jar MyScriptAsJar.jar`

## Groovy Wrapper Script

The Groovy Wrapper Script below is quite simple.

First it parses its command line arguments using groovy's CliBuilder.

- The argument `-m` defines the groovy script name, w/o `{}.class` suffix
- The optional argument `-d` allows defines the jar fileame, by default the jar filename is build as `{}groovy-script-name.jar`{}.
- The argument `-c` compile the groovy script in the GroovyWrapper. Use this argument if your groovy script is not compiled by some build process.

Next it checks the arguments, compiles the script if specified.

Then the environment variable `GROOVY_HOME` is looked up for calculating the file location of the groovy embeddable jar.

Finally GroovyWrapper builds the jar using AntBuilder.

The source of the GroovyWrapper script:

```
/*
 * Copyright 2002-2007 the original author
 or authors.
 *
 * Licensed under the Apache License,
 Version 2.0 (the "License");
 * you may not use this file except in
 compliance with the License.
 * You may obtain a copy of the License at
 *
 *
 http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or
 agreed to in writing, software
 * distributed under the License is
 distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
```

**KIND, either express or implied.**

**\* See the License for the specific language governing permissions and**

**\* limitations under the License.**

**\*/**

**/\*\***

**\* Wrap a script and groovy jars to an executable jar**

**\*/**

**def cli = new CliBuilder()**

**cli.h( longOpt: 'help', required: false, 'show usage information' )**

**cli.d( longOpt: 'destfile', argName: 'destfile', required: false, args: 1, 'jar destination filename, defaults to {mainclass}.jar' )**

**cli.m( longOpt: 'mainclass', argName: 'mainclass', required: true, args: 1, 'fully qualified main class, eg. HelloWorld' )**

**cli.c( longOpt: 'groovyc', required: false, 'Run groovyc' )**

**//-----**  
**-----**

**def opt = cli.parse(args)**

**if (!opt) { return }**

**if (opt.h) {**  
**cli.usage();**  
**return**

**}**

```
def mainClass = opt.m
def scriptBase = mainClass.replace( '.', '/'
)
def scriptFile = new File( scriptBase +
'.groovy' )
if (!scriptFile.canRead()) {
    println "Cannot read script file:
'${scriptFile}'"
    return
}
def destFile = scriptBase + '.jar'
if (opt.d) {
    destFile = opt.d
}

//-----
-----

def ant = new AntBuilder()

if (opt.c) {
    ant.echo( "Compiling ${scriptFile}" )

org.codehaus.groovy.tools.FileSystemCompiler
.main( [ scriptFile ] as String[] )
}

def GROOVY_HOME = new File(
System.getenv( 'GROOVY_HOME' ) )
if (!GROOVY_HOME.canRead()) {
    ant.echo( "Missing environment variable
```

```
GROOVY_HOME: '${GROOVY_HOME}'" )
    return
}

ant.jar( destfile: destFile, compress: true,
index: true ) {
    fileset( dir: '.', includes: scriptBase +
'*.class' )

        zipgroupfileset( dir: GROOVY_HOME,
includes: 'embeddable/groovy-all-*.jar' )
        zipgroupfileset( dir: GROOVY_HOME,
includes: 'lib/commons*.jar' )
        // add more jars here

    manifest {
        attribute( name: 'Main-Class', value:
mainClass )
    }
}
```

```
ant.echo( "Run script using: \'java -jar  
{destFile} ...\'" )
```

## Example

An example for using GroovyWrapper:

Say you have groovy script HelloWorld.groovy, use GroovyWrapper for building HelloWorld.jar, as follows:

```
| $ groovy GroovyWrapper -c -m HelloWorld
```

GroovyWrapper will compile the script HelloWorld.groovy to HelloWorld.class, and creates a self-executable jar HelloWorld.jar.

Now you can use the HelloWorld.jar for launching the HelloWorld script, simply by running:

```
| $ java -jar HelloWorld.jar
```

## Writing to a Blob

### Writing to Blob!

Note: tested with Oracle DB 9.x and a thin driver 9.x !

```
import groovy.sql.Sql

println "---- A working test of writing and  
then reading a blob into an Oracle DB ----"
sql =
Sql.newInstance("jdbc:oracle:thin:@pignut:15  
21:TESTBNDY", "userName",
                "paSSword",
                "oracle.jdbc.OracleDriver")
sql.execute(
    "INSERT INTO MEDIA VALUES  
(SEQ_MEDIAID.NextVal, empty_blob())");

sql.connection.autoCommit = false
```

```
try {
    row = sql.firstRow("select
SEQ_MEDIAID.CurrVal from Dual")
    mediaID = row[0]

    row = sql.firstRow("select binarydata
from media where mediaid = ? for
update",[mediaID])

    my_blob = (oracle.sql.BLOB)row[0]
    if( my_blob == null ) println "my_blob
is null!"

    // write the array of binary data to a
BLOB
    ostream =
my_blob.getBinaryOutputStream();

    // read data into a byte array
    data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] as
byte[];
    ostream.write(data);
    ostream.close()

    sql.commit()
} catch (Exception e) {
    println "Failed: $e"
    sql.rollback()
```

```
} finally {  
    sql.connection.close()  
}
```

Uses a table:

```
CREATE TABLE MEDIA  
(  
    MEDIAID NUMBER(22) NOT NULL,  
    BINARYDATA BLOB NOT NULL  
);  
  
CREATE SEQUENCE SEQ_MEDIAID  
INCREMENT BY 1  
START WITH 100  
ORDER;
```

. Copying a file to Blob!

```

...
    // write the array of binary data to a
    BLOB
    ostream =
my_blob.getBinaryOutputStream();

    def fis= new
FileInputStream('c:\\Jornada\\auxil\\010306_
115332.jpg')
    println "File size= ${fis.available()}"
    byte[] data = new byte[fis.available()]
    fis.read(data)
    fis.close()

    ostream.write(data);
    ostream.close()

    sql.commit()
    ...

```

## Yaml and Groovy

[Yaml](#) is a straightforward machine parsable data serialization format designed for human readability and interaction with scripting languages. One of its reasons for popularity is that XML can sometimes be quite verbose and cumbersome. Groovy's XML capabilities make XML less verbose (e.g. through its builder notation for creating XML) and less cumbersome (e.g. its excellent XML processing [capabilities](#)). Even so, there may be times when you need to or prefer to use Yaml over XML. Here are some examples of how to use Groovy with the [JYaml](#) library.

First, reading simple lists:

```
//
require(url:'http://jyaml.sourceforge.net',
jar:'jyaml.jar', version:'1.0')
import org.ho.yaml.Yaml

def input = '''
- 1
- apple
- orange
'''

assert Yaml.load(input) == [1, 'apple',
'orange']
```

Reading maps:

```
// ...
input = '''
Name: Andre Agassi
Height: 5'11'' (180 cm)
Birthplace: "Las Vegas, Nevada, USA"
Turned Pro: 1986
'''

def player = Yaml.load(input)
assert player.Birthplace == "Las Vegas,
Nevada, USA"
assert player['Turned Pro'] == 1986
```

Reading into classes:

```
// ...
class Staff {
  def firstname, lastname, position
}

input = '''
firstname: John
lastname: Connor
position: Resistance Leader
'''

Staff s = Yaml.loadType(input, Staff)
println s.dump()
// => <Staff@c05d3b firstname=John
lastname=Connor position=Resistance Leader>
```

Writing lists:

```
// ...
def list = ['a, b', 123456]
println Yaml.dump(list)
```

This produces:

```
---
- "a, b"
- 123456
```

**New Documents**

[New User Guide](#)

- [Downloading, Installing and Running](#)
- [Up and Running](#)

## New User Guide

### Getting Started

#### [Downloading, Installing and Running](#)

- [Download](#)
- [Installing Groovy](#)
- [Running](#)

#### [Up and Running](#)

- [Quick Start](#)
- [Beginners Tutorial](#)
  - [Getting started](#)
  - [Code as data, or closures](#)
  - [Classes and Objects](#)
  - [Regular expressions basics](#)
  - [Capturing regex groups](#)
  - [Groovy SQL](#)
- [Feature Overview](#)
- [Differences from Java](#)
- [Differences from Ruby](#)

### Some of the Groovy Modules

- - Ant Scripting
  - Groovlets
  - Groovy Beans
  - Groovy Templates
  - GroovyMarkup
  - CLIBuilder

### The basics

#### Language basics

- Collections
- Closures
- Control Structures
  - Logical Branching
  - Looping
- Builders
- Bitwise Operations
- Statements
- Strings
- Groovy Math
- Input Output
- Logging
- Operator Overloading
- Regular Expressions
- GPath

- Scoping and the Semantics of "def"
- Scripts and Classes
- Things to remember
- Groovy Categories
- Migration From Classic to JSR syntax

### **Groovy from the Command-Line**

- Groovy CLI
- Groovy Console

### **Integrating Groovy into your build**

- Groovy Ant Task
- Groovyc Ant Task
- Ant Task Troubleshooting
- Compiling Groovy

### **Working with Java**

- Bean Scripting Framework
- Embedding Groovy

### **Working with XML**

- Processing XML
  - Creating XML using Groovy's MarkupBuilder
  - Creating XML using Groovy's StreamingMarkupBuilder
  - Creating XML with Groovy and DOM
  - Creating XML with Groovy and DOM4J
  - Creating XML with Groovy and JDOM
  - Creating XML with Groovy and XOM
  - Reading XML using Groovy's DOMCategory
  - Reading XML using Groovy's XmlParser
  - Reading XML using Groovy's XmlSlurper
  - Reading XML with Groovy and DOM
  - Reading XML with Groovy and DOM4J
  - Reading XML with Groovy and Jaxen
  - Reading XML with Groovy and JDOM
  - Reading XML with Groovy and SAX
  - Reading XML with Groovy and StAX
  - Reading XML with Groovy and XOM
  - Reading XML with Groovy and XPath
  - XML Example

### **More Advanced Topics**

#### **DSLs and Functional Programming**

- Functional Programming
- Writing Domain-Specific Languages

#### **Creating a Builder**

- BuilderSupport
- Make a builder

- Leveraging Spring
- Mixed Java and Groovy Applications
- Security
- Advanced OO
  - Groovy way to implement interfaces

## Testing with Groovy

- Groovy Mocks — used to assist (typically unit) testing of classes in isolation
  - Developer Testing using Closures instead of Mocks
  - Developer Testing using Maps and Expandos instead of Mocks
  - Groovy Mocks and Java Mocks - An Extended Example
- Integrating TPTP — some hints for using the Eclipse TPTP facilities with Groovy
- Test Combinations — some hints for using Groovy to assist generate test data in particular all combinations and all pair combinations
  - Effectiveness of testing combinations with all pairs
- Test Coverage — is a useful measure of the effectiveness of unit tests and can be derived for Groovy tests
  - Code Coverage with Cobertura
- Testing Web Applications — how to use NekoHTML, HtmlUnit, Watij and WebTest to test web applications
- Testing Web Services — how to test Web Services using Groovy directly and in conjunction with WebTest and SoapUI
- Unit Testing — Groovy simplifies JUnit testing, making it more Groovy, in several ways

## Downloading, Installing and Running

### Downloading, Installing and Running

In this section we look at how to get up and running with Groovy.

#### *Download*

**In this download area, you will be able to download the distribution (binary and source), the Windows installer (for some of the versions) and the documentation for Groovy.**

For a quick and effortless start on Mac OSX, Linux or Cygwin, you can use [GVM](#) (the Groovy enVironment Manager) to download and configure any Groovy version of your choice. Basic [instructions](#) can be found below.

## ***Groovy 2.1***

Groovy 2 is the latest official major [version of Groovy](#).

Learn more about Groovy 2.1 in the [release notes](#).

#### Groovy 2.1.5

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

#### Groovy 2.1.4

**Download zip:** [Binary Release](#) | [Source Release](#)  
**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)  
**Download documentation:** [JavaDoc and zipped online documentation](#)  
**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

#### Groovy 2.1.3

**Download zip:** [Binary Release](#) | [Source Release](#)  
**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)  
**Download documentation:** [JavaDoc and zipped online documentation](#)  
**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

#### Groovy 2.1.2

**Download zip:** [Binary Release](#) | [Source Release](#)  
**Download Windows-Installer:** [Binary Release \(Module Versions\)](#)  
**Download documentation:** [JavaDoc and zipped online documentation](#)  
**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

## ***Groovy 2.0***

Groovy 2.0 is the [minor version](#) before Groovy 2.1.

Learn more about Groovy 2.0 in the [release notes](#).

#### Groovy 2.0.8

**Download zip:** [Binary Release](#) | [Source Release](#)  
**Download documentation:** [JavaDoc and zipped online documentation](#)  
**Combined binary / source / documentation bundle:** [Distribution bundle](#)

You can have a look at the [JIRA release notes](#).

If you plan on using invokedynamic support, [read those notes](#).

#### ***Groovy 1.8 (Maintenance Release)***

Groovy 1.8 is the latest maintenance release branch of the Groovy 1 major version.

Learn more about the Groovy 1.8 in the [release notes](#).

Groovy 1.8.9

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

You can have a look at the [JIRA release notes](#).

## Snapshots

For those who want to test the very latest versions of Groovy and live on the bleeding edge, you can use our [snapshot builds](#). As soon as a build succeeds on our continuous integration server a snapshot is deployed to Codehaus' snapshot repository.

## Older Releases

All maintenance of the Groovy 1.5.X, 1.6.X and 1.7.X series has now stopped. These are the last released versions.

### ***Groovy 1.7.11***

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

You can have a look at the [JIRA release notes](#).

Groovy 1.6.9

**Download zip:** [Binary Release \(JDK 1.4 version\)](#) | [Source Release](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

You can have a look at the [JIRA release notes](#).

Groovy 1.5.8

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download Windows-Installer:** [Binary Release](#)

**Download unofficial Fedora/RHEL/CentOS package:** [Binary Release](#) | [Source Release](#)

**Download documentation:** [JavaDoc and zipped online documentation](#)

The release notes are [Release Notes](#).

### ***Legacy Groovy 1.0***

If you still need to use the old Groovy 1.0, here are the links you may be interested in:

**Download zip:** [Binary Release](#) | [Source Release](#)

**Download unofficial Ubuntu/Debian package:** [Binary Release](#)

**Download Windows-Installer:** [Binary Release](#)

**Download tar/gz:** [Binary Release](#) | [Source Release](#)

**Download Javadoc:** [Javadoc zip](#)

[Once you've downloaded the distribution, please read the installation instructions.](#)

Other versions can be found in the [distributions archive](#).

## Maven Repositories

If you wish to embed Groovy in your application, you may just prefer to point to your favourite maven repositories or the [codehaus maven repository](#). You should consult the individual pom files for the exact details but here is a summary of the main supported artifactIds and groupIds:

### Groovy 2.0.X and newer

Gradle	Maven	Explanation
'org.codehaus.groovy:groovy:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Just the core of groovy without the modules (see below). Treats Antlr, ASM, etc. as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars.
'org.codehaus.groovy:groovy-\$module:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-\$module&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	"\$module" stands for the different optional groovy modules "ant", "bsf", "console", "docgenerator", "groovydoc", "groovysh", "jmx", "json", "jsr223", "servlet", "sql", "swing", "test", "testng" and "xml". Example: <pre>&lt;artifactId&gt;groovy-sql&lt;/artifactId&gt;</pre>
'org.codehaus.groovy:groovy-all:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-all&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	The core plus all the modules. Also includes jarjar versions of Antlr, ASM, Commons-CLI and Retrotranslator runtime. Allows you or your other dependencies (e.g. Hibernate) to use other versions of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.

To use the [InvokeDynamic](#) version of the jars just append ':indy' for Gradle or `<classifier>indy</classifier>` for Maven.

### Groovy 1.6.X - 1.8.X

Gradle	Maven	Explanation
--------	-------	-------------

'org.codehaus.groovy:groovy:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Treats Antlr, ASM, etc. as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'org.codehaus.groovy:groovy-all:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-all&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Includes jarjar versions of Antlr, ASM, Commons-CLI and Retrotranslator runtime. Allows you or your other dependencies (e.g. hibernate) to use other versions of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.

### Groovy 1.1.X - 1.5.X

Gradle	Maven	Explanation
'org.codehaus.groovy:groovy:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Treats Antlr and ASM as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'org.codehaus.groovy:groovy-all:x.y.z'	<pre>&lt;groupId&gt;org.codehaus.groovy&lt;/groupId&gt; &lt;artifactId&gt;groovy-all&lt;/artifactId&gt; &lt;version&gt;x.y.z&lt;/version&gt;</pre>	Includes jarjar versions of Antlr, ASM and Commons-CLI. Allows you or your other dependencies (e.g. hibernate) to use other versions of these jars. Optional dependencies are marked as optional. You may need to include some of the optional dependencies to use some features of Groovy, e.g. AntBuilder, GroovyMBeans, etc.

'org.codehaus.groovy:groovy-all-minimal:x.y.z'	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy-all-minimal</artifactId> <version>x.y.z</version>	Includes jarjar versions of Antlr and ASM. Does not include Commons-CLI or any optional dependencies. Not suitable by itself if you want to use any tools which do command-line processing, e.g. groovy-c, GroovyShell, ...
--	---	---

### Groovy 1.0

Available in the [Maven 1](#) and [Maven 2](#) repositories.

Gradle	Maven	Explanation
'groovy:groovy:x.y.z'	<groupId>groovy</groupId> <artifactId>groovy</artifactId> <version>1.0</version>	Treats Antlr and ASM as standard dependencies. Only useful if you happen to also use the same versions of these jars yourself as it will save you having two copies of these jars. Dependencies mandated for all optional parts of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'groovy:groovy-all:x.y.z'	<groupId>groovy</groupId> <artifactId>groovy-all</artifactId> <version>1.0</version>	Includes jarjar versions of Antlr and ASM. Allows you or your other dependencies (e.g. hibernate) to use other versions of these jars. Dependencies mandated for all optional parts of Groovy, e.g. AntBuilder, GroovyMBeans, etc.
'groovy:groovy-all-minimal:x.y.z'	<groupId>groovy</groupId> <artifactId>groovy-all-minimal</artifactId> <version>1.0</version>	Includes jarjar versions of Antlr and ASM. Does not include any optional dependencies.

For historical purposes, the releases candidates for 1.0 are also available by using the appropriate version instead of 1.0 in the above version tag.

### Snapshot Releases

In addition to the stable and milestone releases you can find intermediate SNAPSHOT releases at the [codehaus snapshot maven repository](#).

### GVM (the Groovy enVironment Manager)

This tool makes installing Groovy on any Bash platform (Mac OSX, Linux, Cygwin, Solaris or FreeBSD) very easy.

Simply open a new terminal and enter:

```
$ curl -s get.gvmtool.net | bash
```

Follow the instructions on-screen to complete installation.

Open a new terminal or type the command:

```
$ source "$HOME/.gvm/bin/gvm-init.sh"
```

Then install the latest stable Groovy:

```
$ gvm install groovy
```

After installation is complete and you've made it your default version, test it with:

```
$ groovy -version
```

That's all there is to it!

### **Other ways to get Groovy**

If you're on MacOS and have [Homebrew](#) installed, you can run "brew install groovy" to install Groovy.

If you're on MacOS and have [MacPorts](#) installed, you can run "sudo port install groovy" to install the latest Groovy release.

If you're on Windows, you can also use the [NSIS Windows installer](#).

You may download other distributions of Groovy from [this site](#).

If you prefer to live on the bleeding edge, you can also grab the [source code from SVN](#).

If you are an IDE user, you can just grab the latest [IDE plugin](#) and follow the plugin installation instructions.

### ***Installing***

## **These instructions describe how to install a binary distribution of Groovy.**

- first, [Download](#) a binary distribution of Groovy and unpack it into some file on your local file system
- set your `GROOVY_HOME` environment variable to the directory you unpacked the distribution
- add `GROOVY_HOME/bin` to your `PATH` environment variable
- set your `JAVA_HOME` environment variable to point to your JDK. On OS X this is `/Library/Java/Home`, on other unices its often `/usr/java` etc. If you've already installed tools like Ant or Maven you've probably already done this step.

You should now have Groovy installed properly. You can test this by typing the following in a command shell:

```
groovysh
```

Which should create an interactive groovy shell where you can type Groovy statements. Or to run the [Swing interactive console](#) type:

```
groovyConsole
```

To run a specific Groovy script type:

```
groovy SomeScript.groovy
```

#### Alternative Installation on Mac OS X

Set up [MacPorts](#)

Open Terminal

```
sudo port install groovy
```

#### *Running*

**Groovy scripts are a number of statements and class declarations in a text file. Groovy scripts can be used similarly to other scripting languages. There are various ways of running Groovy scripts**

#### *Using the interactive console*

Groovy has a Swing interactive console that allows you to type in commands and execute them rather like using an SQL query tool. History is available and such like so you can move forwards and backwards through commands etc.

If you [install](#) a binary distribution of Groovy then you can run the Groovy Swing console by typing this on the command line.

```
groovyConsole
```

For a command line interactive shell type

```
groovysh
```

To see how to add things to the classpath see below.

### ***Running Groovy scripts from your IDE***

There is a helper class called [GroovyShell](#) which has a `main(String[])` method for running any Groovy script. You can run any groovy script as follows

```
java groovy.lang.GroovyShell  
foo/MyScript.groovy [arguments]
```

You can then run the above Groovy `main()` in your IDE to run or debug any Groovy script.

### ***Running Groovy scripts from the command line***

There are shell scripts called 'groovy' or 'groovy.bat' depending on your platform which is part of the Groovy runtime. Once the runtime is [installed](#) you can just run groovy like any other script...

```
groovy foo/MyScript.groovy [arguments]
```

If you are using Groovy built from CVS Head (after Beta-5, see below if you want to upgrade), apart from Groovy scripts, you may also now run different kind of classes from the command-line.

- Classes with a main method of course,
- Classes extending `GroovyTestCase` are run with JUnit's test runner,
- Classes implementing the `Runnable` interface are instantiated either with a constructor with `String[]` as argument, or with a no-args constructor, then their `run()` method is called.

To work from the latest and greatest Groovy see [Building Groovy from Source](#). Once built you'll then have a full binary distribution made for you in `groovy/target/install`. You can then add `groovy/target/install/bin` to your path and you can then run groovy scripts easily from the command line.

To see how to add things to the classpath see below.

### ***Creating Unix scripts with Groovy***

You can write unix scripts with Groovy and execute them directly on the command line as if they were normal unix shell scripts. Providing you have installed the Groovy binary distribution (see above) and 'groovy' is on your PATH then the following should work.

The following is a sample script, which you should copy and save as `helloWorld.groovy`.

```
#!/usr/bin/env groovy
println("Hello world")
for (a in this.args) {
    println("Argument: " + a)
}
```

Then to run the script from the command line, just make sure the script is executable then you can call it.

```
chmod +x helloWorld
./helloWorld
```

Note that you will not be able to provide any arguments to groovy when running with the shebang(#!) on Linux because the arguments are treated as part of the name of the command to run. Mac OS X will interpret additional arguments correctly.

#### ***Adding things to the classpath***

When running command line scripts or interactive shells you might want to add things to your classpath such as JDBC drivers or JMS implementations etc. To do this, you have a few choices:

- Add things to your CLASSPATH environment variable
- Pass -classpath (or -cp) into the command you used to create the shell or run the script
- It's also possible to create a `~/groovy/lib` directory and add whatever jars you need in there.
- If the jars you need are in a Maven or Ivy repository, you can "grab" them with [Grape](#).

#### ***Increasing Groovy's JVM Heap Size***

To increase the amount of memory allocated to your groovy scripts, set your JAVA\_OPTS environment variable. `JAVA_OPTS="-Xmx..."`

#### ***Avoiding PermGen out of memory***

Groovy creates classes dynamically, but the default Java VM does not GC the PermGen. If you are using Java 6 or later, add `-XX:+CMSClassUnloadingEnabled -XX:+UseConcMarkSweepGC`. `UseConcMarkSweepGC` is needed to enable `CMSClassUnloadingEnabled`.

## **Up and Running**

### **Up and Running**

#### ***Quick Start***

#### **Getting started**

## Getting Started

### *Setting up your Java environment*

Groovy requires Java, so you need to have a version available (while groovy 1.6 supported JDK 1.4 or greater, for groovy 1.7 onwards, minimum JDK 1.5 is needed). Here are the steps if you don't already have Java installed:

- Get the suitable Java distribution from the <http://java.sun.com> website.
- Run the installer.
- Set the JAVA\_HOME environment variables. On Windows, follow these steps:
  - Open the System control panel
  - Click the Advanced tab
  - Click the Environment Variables button
  - Add a new System variable with the name JAVA\_HOME and the value of the directory Java was installed in (mine is C:\Program Files\Java\jdk1.5.0\_04)
  - Optionally add %JAVA\_HOME%\bin to your system path  
(Note: as an alternative to setting a system environment variable, you can create yourself a '.bat' or '.cmd' file which sets the variable. You then need to run that batch file in any console window in which you wish to run Java and double clicking on '.bat' or '.cmd' files containing Java invocation instructions won't work. If you are unsure about what this means, follow the earlier instructions.)

### *Setting up your Groovy environment*

Download the Groovy installer or binaries from the [downloads](#) page and follow the installation instructions. (There is currently an issue where you cannot have spaces in the path where Groovy is installed under windows. So, instead of accepting the default installation path of "c:\Program Files\Groovy" you will want to change the path to something like "c:\Groovy")

OR

- Get a copy of the Groovy distribution from the [website](#), and copy it to some place on your hard drive.
- Unzip the groovy archive to some logical place on your hard drive, I have mine in C:\dev\groovy-2.0.5
- Set the GROOVY\_HOME environment variables. On Windows, follow these steps:
  - Optionally add a new System variable with the name GROOVY\_HOME and the value of the directory groovy was installed in (mine is C:\dev\groovy-2.0.5)
  - Start a command prompt, and type "set" and hit return to see that your environment variables were set correctly.
- Optionally add %GROOVY\_HOME%\bin to your system path
- Try opening groovyConsole.bat by double clicking on the icon in the bin directory of the Groovy distribution. If it doesn't work, open a command prompt, and change to the bin directory and run it from there to see what the error message is. If it is complaining about not finding parts of Groovy, then setting GROOVY\_HOME may help,

### *Setting up optional jar files*

You may wish to obtain optional jar files, either corresponding to Groovy modules (see module documentation for details) or corresponding to other Java classes you wish to make use of from Groovy. Some possibilities are listed below:

Name	From	Description
jtids- <i>version</i> .jar	<a href="http://jtids.sourceforge.net">http://jtids.sourceforge.net</a>	Database driver for SQL Server and/or Sybase

hsqldb-version.jar	<a href="http://www.hsqldb.org/">http://www.hsqldb.org/</a>	Database driver for HSQLDB, a 100% Java database
--------------------	---	--

The recommended way for making Groovy be aware of your additional jar files is to place them in a predefined location. Your Groovy install should include a file called `groovy-starter.conf`. Within that file, make sure a line such as

```
load ${user.home}/.groovy/lib/*
```

is not commented out. The `user.home` system property is set by your operating system. (Mine is `C:\Document and Settings\paul`. Now simply place your jar files into the `.groovy/lib` directory. (Note: as an alternative, you can set up a `CLASSPATH` variable and make sure it mentions all of your additional jar files, otherwise Groovy works fine with an empty or no `CLASSPATH` variable.)

### *Hello, World*

In the top part of the window of the `groovyConsole`, type the following

```
println "Hello, World!"
```

And then type `<CTRL-R>`.

Notice that the text gets printed out in the OS console window (the black one behind the `groovyConsole` window) and the bottom part of the `groovyConsole` says:

```
groovy> println "Hello, World!"  
null
```

The line starting with `"groovy>"` is just the text of what the console processed. The `"null"` is what the expression "evaluated to". Turns out the expression to print out a message doesn't have any "value" so the `groovyConsole` printed `"null"`.

Next try something with an actual value. Replace the text in the console with:

```
123+45*67
```

or your favorite arithmetic expression, and then type `<CTRL-R>` (I'm going to stop telling you to hit `<CTRL-R>`, I

think you get the idea). Now the "value" printed at the bottom of the groovyConsole has more meaning.

### *Variables*

You can assign values to variables for later use. Try the following:

```
x = 1
println x

x = new java.util.Date()
println x

x = -3.1499392
println x

x = false
println x

x = "Hi"
println x
```

### *Lists and Maps*

The Groovy language has built-in support for two important data types, lists and maps (Lists can be operated as arrays in Java language). Lists are used to store ordered collections of data. For example an integer list of your favorite integers might look like this:

```
myList = [1776, -1, 33, 99, 0, 928734928763]
```

You can access a given item in the list with square bracket notation (indexes start at 0):

```
println myList[0]
```

Should result in this output:

```
1776
```

You can get the length of the list with the "size" method:

```
println myList.size()
```

Should print out:

```
6
```

But generally you shouldn't need the length, because unlike Java, the preferred method to loop over all the elements in an list is to use the "each" method, which is described below in the "Code as Data" section.

Another native data structure is called a map. A map is used to store "associative arrays" or "dictionaries". That is unordered collections of heterogeneous, named data. For example, let's say we wanted to store names with IQ scores we might have:

```
scores = [ "Brett":100, "Pete":"Did not  
finish", "Andrew":86.87934 ]
```

Note that each of the values stored in the map is of a different type. Brett's is an integer, Pete's is a string, and Andrew's is a floating point number. We can access the values in a map in two main ways:

```
println scores["Pete"]  
println scores.Pete
```

Should produce the output:

```
Did not finish  
Did not finish
```

To add data to a map, the syntax is similar to adding values to an list. For example, if Pete re-took the IQ test and got a 3, we might:

```
scores["Pete"] = 3
```

Then later when we get the value back out, it will be 3.

```
println scores["Pete"]
```

should print out 3.

Also as an aside, you can create an empty map or an empty list with the following:

```
emptyMap = [:]  
emptyList = []
```

To make sure the lists are empty, you can run the following lines:

```
println emptyMap.size()  
println emptyList.size()
```

Should print a size of 0 for the List and the Map.

### ***Conditional Execution***

One of the most important features of any programming language is the ability to execute different code under different conditions. The simplest way to do this is to use the "if" construct. For example:

```
amPM =
Calendar.getInstance().get(Calendar.AM_PM)
if (amPM == Calendar.AM)
{
    println("Good morning")
} else {
    println("Good evening")
}
```

Don't worry too much about the first line, it's just some code to determine whether it is currently before noon or after. The rest of the code executes as follows: first it evaluates the expression in the parentheses, then depending on whether the result is "true" or "false" it executes the first or the second code block. See the section below on boolean expressions.

Note that the "else" block is not required, but the "then" block is:

```
amPM =
Calendar.getInstance().get(Calendar.AM_PM)
if (amPM == Calendar.AM)
{
    println("Have another cup of coffee.")
}
```

### ***Boolean Expressions***

There is a special data type in most programming languages that is used to represent truth values, "true" and "false". The simplest boolean expressions are simply those words. Boolean values can be stored in variables, just like any other data type:

```
myBooleanVariable = true
```

A more complex boolean expression uses one of the boolean operators:

```
* ==
* !=
* >
* >=
* <
* <=
```

Most of those are probably pretty intuitive. The equality operator is "==" to distinguish from the assignment operator "=". The opposite of equality is the "!=" operator, that is "not equal". So some examples:

```
titanicBoxOffice = 1234600000
titanicDirector = "James Cameron"

trueLiesBoxOffice = 219000000
trueLiesDirector = "James Cameron"

returnOfTheKingBoxOffice = 752200000
returnOfTheKingDirector = "Peter Jackson"

theTwoTowersBoxOffice = 581200000
theTwoTowersDirector = "Peter Jackson"

titanicBoxOffice > returnOfTheKingBoxOffice
// evaluates to true
titanicBoxOffice >= returnOfTheKingBoxOffice
// evaluates to true
titanicBoxOffice >= titanicBoxOffice
// evaluates to true
titanicBoxOffice > titanicBoxOffice
```

```
// evaluates to false
titanicBoxOffice + trueLiesBoxOffice <
returnOfTheKingBoxOffice +
theTwoTowersBoxOffice // evaluates to false

titanicDirector > returnOfTheKingDirector
// evaluates to false, because "J" is before
"p"
titanicDirector < returnOfTheKingDirector
// evaluates to true
titanicDirector >= "James Cameron"
```

```
// evaluates to true
titanicDirector == "James Cameron"
// evaluates to true
```

Boolean expressions are especially useful when used in conjunction with the "if" construct. For example:

```
if (titanicBoxOffice + trueLiesBoxOffice >
returnOfTheKingBoxOffice +
theTwoTowersBoxOffice)
{
    println(titanicDirector + " is a better
director than " + returnOfTheKingDirector)
}
```

An especially useful test is to test whether a variable or expression is null (has no value). For example let's say we want to see whether a given key is in a map:

```
suvMap = ["Acura MDX": "\$36,700", "Ford
Explorer": "\$26,845"]
if (suvMap["Hummer H3"] != null)
{
    println("A Hummer H3 will set you back
"+suvMap["Hummer H3"]);
}
```

Generally null is used to indicate the lack of a value in some location.

### Debugging and Troubleshooting Tips

- Print out the class of a variable that you're interested in with `myVar.getClass()`. Then look up the documentation for that class.
- If you're having trouble with a complex expression, pare it down to a simpler expression and evaluate that. Then build up to your more complex expression.

- Try restarting the groovyConsole (this will clear out all the variables so you can start over.
- Look for the topic you're interested in in the Groovy [User Guide](#)

### If you are a Java developer

- you might want to check on the [Differences from Java](#)
- also there a few [Things to remember](#)

Code as data

### Closures

One of the things that makes Groovy different than most compiled languages is that you can create functions that are first class objects. That is you can define a chunk of code and then pass it around as if it were a string or an integer. Check out the following code:

```
square = { it * it }
```

The curly braces around the expression "it \* it" tells the Groovy compiler to treat this expression as code. In the software world, this is called a "closure". In this case, the designator "it" refers to whatever value is given to the function. Then this compiled function is assigned to the variable "square" much like those above. So now we can do something like this:

```
square(9)
```

and get the value 81.

This is not very interesting until we find that we can pass this function "square" around as a value. There are some built in functions that take a function like this as an argument. One example is the "collect" method on arrays. Try this:

```
[ 1, 2, 3, 4 ].collect(square)
```

This expression says, create an array with the values 1,2,3 and 4, then call the "collect" method, passing in the closure we defined above. The collect method runs through each item in the array, calls the closure on the item, then puts the result in a new array, resulting in:

```
[ 1, 4, 9, 16 ]
```

For more methods you can call with closures as arguments, see the [Groovy GDK documentation](#).

By default closures take 1 parameter called "it", you can also create closures with named parameters. For example

the method `Map.each()` can take a closure with two variables, to which it binds the key and associated value:

```
printMapClosure = { key, value -> println  
key + "=" + value }  
[ "Yue" : "Wu", "Mark" : "Williams", "sudha"  
: "Kumari" ].each(printMapClosure)
```

Produces:

```
Yue=Wu  
Mark=Williams  
Sudha=Kumari
```

#### *More Closure Examples*

Here are a few more closure examples. This first one shows a couple of things. First, the closure is interacting with a variable outside itself. That is, the closure's purpose is to put together the parts of a stock order held in the array `orderParts`, by adding (appending) it to the variable `fullString`. The variable `fullString` is not in the closure. The second thing that is different about this example is that the closure is "anonymous", meaning that it is not given a name, and is defined in the place where the `each` method is called.

```
fullString = ""  
orderParts = ["BUY", 200, "Hot Dogs", "1"]  
orderParts.each {  
  fullString += it + " "  
}  
  
println fullString
```

You can probably guess what this prints out.

The next example is another anonymous closure, this time, summing up the values stored in a map.

```
myMap = [ "China": 1 , "India" : 2, "USA" :
3]

result = 0
myMap.keySet().each( { result+= myMap[it] }
)
println result
```

### Dealing with Files

Reading data from files is relatively simple. First create a text file, and call it myfile.txt. It doesn't matter what's in it, just type some random text into it and save it on your C: drive in the \temp directory. Then type the following code in the groovyConsole:

```
myFileDirectory = "C:\\temp\\"
myFileName = "myfile.txt"
myFile = new File(myFileDirectory +
myFileName)

printFileLine = { println "File line: " + it
}

myFile.eachLine( printFileLine )
```

This should print out every line in the file prefixed with "File line: ". The first two lines of the code simply declare variables to specify where the file is located. The variable names don't have any special significance, and as you can see, all we do is combine them when we use them. Note that because the backslash character has special meaning in groovy, you have to use two of them to tell it that you "really" mean a backslash.

The next line that starts "myFile =" creates a new File object. An object is simply a collection of related methods and data. For example, a file object might have data describing its location, in this case "C:\temp\myfile.txt", and maybe a method to delete the file if it exists. In this case the only method we are going to use is the eachLine method, which we call in the last line of code. The line before that is a simple closure definition, that you have seen several times by this point.

### Dealing with strings

Strings in Groovy have all the same functionality of Java strings. That is, a Groovy string is just a Java string with a few extra things added to it. Because of that, we can refer to the [Java documentation for the String class](#) to find out some of the interesting things we can do with it. For example, look in the section entitled "Method Summary" at the description for the "split" method. This method does something very useful, which is to split a string based on a regular expression. We will talk more about regular expressions later, but for now the only thing you have to know is that the simplest regular expression is a single character. So let's say that we want to split up the components of the date "2005-07-04", so that we can add one to the year to get the date of next fourth of July. We might:

```
stringDate = "2005-07-04"  
dateArray = stringDate.split("-") // split()  
uses regex's, so you need to escape chars  
such as a "." -> "\\."  
year = dateArray[0].toInteger()  
year = year + 1  
newDate = year + "-" + dateArray[1] + "-" +  
dateArray[2]
```

This code brings together a bunch of things we have talked about before. There are two new things, first is the use of the split method on a String. Second is the call of toInteger() on a String. This call to toInteger simply tells Groovy that you want to treat that data as a number rather than a String. See what happens if you run the same code without ".toInteger()" at the end of the third line.

Another thing you might notice is that toInteger is not listed in the Java documentation for string. That is because it is one of the extra features that Groovy has added to Strings. You can also take a look at the documentation for the [Groovy extensions to Java objects](#).

- [Classes and Objects](#)
- [Regular expressions basics](#)
- [Capturing regex groups](#)
- [Groovy SQL](#)
- [Feature Overview](#)
- [Differences from Java](#)
- [Differences from Ruby](#)

## **TMPGroovy Categories**

There are many situations where you might find that it would be useful if a class not under your control had additional methods that you define. We call these Categories. There are a few categories that are included in the system for adding functionality to classes that make them more usable within the Groovy environment.

- [DOMCategory](#)
- [ServletCategory](#)

The first category allows you to treat DOM objects as arrays and maps so that you can use them in conjunction with the Groovy path expression language and treat them like JavaBeans. Here is an example from the tests of using the DOMCategory:

## DOMTest.groovy

```
import groovy.xml.*;

def html = DOMBuilder.newInstance().html {
  head {
    title (class:"mytitle", "Test")
  }
  body {
    p (class:"mystyle", "This is a test.")
  }
}

use (groovy.xml.dom.DOMCategory.class) {
  assert html.head.title.textContent ==
"Test";
  assert html.body.p.textContent == "This is
a test.";
  assert html.find { it.tagName == "body"
}.tagName == "body";
  assert
html.getElementsByTagName("*").findAll {
  if (it["@class"] != "") { return it }
}.size() == 2;
}

try {
  html.head;
} catch(MissingPropertyException mpe) {
  println "Categories wear off";
}
```

As you can see here we are treating DOM objects just as if they were JavaBeans and are accessing them with GPath. The ServletCategory is similarly used when we want to treat the attributes of Servlet API objects as if they were properties since they don't follow the typical conventions for JavaBeans or Maps either. In the GroovyServlet that lets you use scripts as servlets we call GroovyCategorySupport from Java in order to make it possible to use property accessors against the request:

## GroovyServlet.java

```
Closure closure = new Closure(gse) {  
    public Object call() {  
        try {  
            return ((GroovyScriptEngine)  
getDelegate()).run(scriptUri, binding);  
        } catch (ResourceException e) {  
            throw new RuntimeException(e);  
        } catch (ScriptException e) {  
            throw new RuntimeException(e);  
        }  
    }  
};  
GroovyCategorySupport.use(ServletCategory.cl  
ass, closure);
```

This allows users to access things like Session attributes and request Attributes by name instead of through the API within their Groovy servlet scripts. For example, without this you would have to do:

```
if (session.getAttribute("count") == null)  
then session.setAttribute("count", 1);
```

With this you can say it more tersely as:

```
if (session.count == null) session.count =  
1;
```

In order to create your own Categories and extend classes yourself you'll need to understand what the "use" keyword expects to be defined within the class you pass to it. To add a method to a class T, simply define a new class with a static method that whose first parameter is of type T. Here is a simple example from the tests:

### CategoryTest.groovy

```
class StringCategory {  
    static String lower(String string) {  
        return string.toLowerCase();  
    }  
}  
  
use (StringCategory.class) {  
    println "TeSt".lower();  
}
```

This code will print out the string "test". This facility is extremely powerful and essentially lets you change the way any class in the system works when it is called from Groovy code. Note though that you can't add methods to classes, pass them to Java code, and expect the Java code to be able to call them. Since most people use statically typed Java with little reflection I doubt this case would come up much anyway.

Here is an example of using this as an end user in order to add methods to Apple's own NSDictionary and NSArray class in order to manipulate their Cocoa objects as if they were native Groovy objects:

### bookmarks.groovy

```
#!/Users/sam/bin/groovy  
// Put /System/Library/Java in your  
CLASSPATH  
import groovy.xml.*;
```

```
import groovy.xml.dom.*;
import java.io.*;
import com.apple.cocoa.foundation.*;

class PropertyListCategory {
    static Object get(NSDictionary
dictionary, String key) {
        return
dictionary.objectForKey(key);
    }
    static Object getAt(NSArray array,
int i) {
        return
array.objectAtIndex(i);
    }
    static void each(NSArray array,
Closure closure) {
        for (i in
0..array.count()-1) {

closure.call(array[i]);
        }
    }
}

filename =
"${System.getProperty("user.home")}/Library/
Safari/Bookmarks.plist";
data = new NSData(new File(filename));
errorString = new String[1];
format = new int[1];
```

```
plist =
NSPropertyListSerialization.propertyListFrom
Data(data,

NSPropertyListSerialization.PropertyListImmu
table, format, errorString);

if (errorString[0]) {
    println "Error: ${errorString[0]}";
    System.exit(1);
}

def getURLs(NSArray array, list) {
    array.each {
        getURLs(it, list);
    }
}

def getURLs(NSDictionary dict, list) {
    if (dict.Children != null)
getURLs(dict.Children, list);
    if (dict.URIDictionary != null) {

list.add([title:dict.URIDictionary.title,
url:dict.URLString]);
    }
}

def getURLs(NSDictionary dict) {
    use (PropertyListCategory.class) {
        def list = [];
```

```
        getURLs(dict, list);  
    }  
    return list;
```

```
}
```

```
println getURLs(plist);
```

Notice how we can even create Category classes in Groovy code. They essentially look just like DefaultGroovyMethods. Define them by creating a static method that takes the type you want to extend, then the additional parameters that the new method will take.

## Community and Support

### Getting in touch

There are various ways of communicating with the Groovy community.

- subscribe to the [mailing lists](#) and take part in any conversations
- add some comments to the [wiki](#)
- report bugs and file new feature requests on our [bug tracker](#)
- learn about the [roadmap](#) and [changelog](#)

Other topics:

- [Support](#)
- [Contributing](#)
- [Forums and Newsgroups](#)
- [Mailing Lists](#)
- [User Groups](#)
- [Modules](#)
  - [COM Scripting](#)
    - [All About Arrays](#)
    - [Change Log](#)
    - [COM Data Types in Scriptom](#)
    - [COM Events](#)
    - [COM Methods and Properties in Scriptom](#)
    - [Passing Values by Reference \(in-out\)](#)
    - [Scriptom Archive](#)
      - [Copy of COM Scripting](#)
      - [Scriptom 1.2](#)
    - [Scriptom Articles](#)
      - [Bridging the Gap Between Java and .NET with Groovy and Scriptom](#)
      - [Getting Started with COM](#)
      - [Using Scriptom to Automate Microsoft Excel](#)
    - [The Least You Need to Know about COM](#)
  - [Gant](#)
  - [GFreeMarker](#)
  - [Google Data Support](#)
  - [Gram](#)
  - [GraphicsBuilder](#)
    - [Alphabetical Index](#)
    - [GraphicsBuilder - Animation](#)
      - [GraphicsBuilder - Animation - Bind](#)

- [GraphicsBuilder - GDK](#)
- [GraphicsBuilder - Groodles](#)
  - [GraphicsBuilder - Groodle1 - Screencast](#)
  - [GraphicsBuilder - Groodle2 - Screencast](#)
  - [GraphicsBuilder - Groodle3 - Screencast](#)
- [GraphicsBuilder - Groups](#)
- [GraphicsBuilder - Misc](#)
  - [GraphicsBuilder - Misc - Antialias](#)
  - [GraphicsBuilder - Misc - Clip](#)
  - [GraphicsBuilder - Misc - Color](#)
  - [GraphicsBuilder - Misc - Draw](#)
  - [GraphicsBuilder - Misc - Font](#)
  - [GraphicsBuilder - Misc - Image](#)
  - [GraphicsBuilder - Misc - RenderingHint](#)
- [GraphicsBuilder - Outlines](#)
  - [GraphicsBuilder - Outlines - CubicCurve](#)
  - [GraphicsBuilder - Outlines - Line](#)
  - [GraphicsBuilder - Outlines - Polyline](#)
  - [GraphicsBuilder - Outlines - QuadCurve](#)
- [GraphicsBuilder - Paints](#)
  - [GraphicsBuilder - Paints - BorderPaint](#)
  - [GraphicsBuilder - Paints - ColorPaint](#)
  - [GraphicsBuilder - Paints - GradientPaint](#)
  - [GraphicsBuilder - Paints - LinearGradient](#)
  - [GraphicsBuilder - Paints - MultiPaint](#)
  - [GraphicsBuilder - Paints - Paint](#)
  - [GraphicsBuilder - Paints - RadialGradient](#)
  - [GraphicsBuilder - Paints - TexturePaint](#)
- [GraphicsBuilder - Shapes](#)
  - [GraphicsBuilder - Area - Add](#)
  - [GraphicsBuilder - Area - Intersect](#)
  - [GraphicsBuilder - Area - Shape](#)
  - [GraphicsBuilder - Area - Subtract](#)
  - [GraphicsBuilder - Area - Xor](#)
  - [GraphicsBuilder - Shapes - Arc](#)
  - [GraphicsBuilder - Shapes - Arrow](#)
  - [GraphicsBuilder - Shapes - Circle](#)
  - [GraphicsBuilder - Shapes - Cross](#)
  - [GraphicsBuilder - Shapes - Donut](#)
  - [GraphicsBuilder - Shapes - Ellipse](#)
  - [GraphicsBuilder - Shapes - Morph](#)
  - [GraphicsBuilder - Shapes - Path](#)
  - [GraphicsBuilder - Shapes - Pin](#)
  - [GraphicsBuilder - Shapes - Polygon](#)
  - [GraphicsBuilder - Shapes - Rays](#)
  - [GraphicsBuilder - Shapes - Rect](#)
  - [GraphicsBuilder - Shapes - RegularPolygon](#)
  - [GraphicsBuilder - Shapes - RoundRect](#)
  - [GraphicsBuilder - Shapes - Star](#)
  - [GraphicsBuilder - Shapes - Text](#)
  - [GraphicsBuilder - Shapes - Triangle](#)
  - [GraphicsBuilder - Shapes - Xpath](#)
- [GraphicsBuilder - Strokes](#)

- [GraphicsBuilder - Strokes - BasicStroke](#)
  - [GraphicsBuilder - Strokes - CompositeStroke](#)
  - [GraphicsBuilder - Strokes - CompoundStroke](#)
  - [GraphicsBuilder - Strokes - Shapetroke](#)
  - [GraphicsBuilder - Strokes - Stroke](#)
  - [GraphicsBuilder - Strokes - TextStroke](#)
  - [GraphicsBuilder - Strokes - WobbleStroke](#)
  - [GraphicsBuilder - Strokes - ZigzagStroke](#)
- [Graphicsbuilder - SVG](#)
- [GraphicsBuilder - Swing](#)
- [GraphicsBuilder - Transformations](#)
- [GraphicsBuilder - Tutorials](#)
- [Grapplet](#)
- [Griffon](#)
  - [Download Griffon](#)
  - [Greet](#)
  - [Griffon JavaOne 2008 Dev Notes](#)
  - [Griffon Quick Start](#)
  - [Griffon RoadMap](#)
  - [Installing Griffon](#)
  - [UberBuilder](#)
- [Groosh](#)
- [Groovy Jabber-RPC](#)
- [GroovyJMS](#)
  - [GroovyJMS Design Docs](#)
  - [GroovyJMS Docs](#)
  - [GroovyJMS Reference Links](#)
  - [GroovyJMS Roadmap and Planning](#)
  - [GroovyJMS - v0.1 Docs and Example](#)
  - [Groovy Messaging Service API](#)
- [GroovyLab](#)
- [Groovy Monkey](#)
- [GroovyRestlet](#)
  - [GroovyRestlet User Guide](#)
  - [RestletTutorialExamples](#)
  - [SpringIntegrationExamples](#)
- [Groovy Science](#)
- [Groovy SOAP](#)
- [GroovySWT](#)
  - [A sample application](#)
  - [Binding](#)
  - [Changelog](#)
  - [groovy-swt - Comparing the java and the groovy code](#)
- [GroovyWS](#)
  - [Consuming a web service](#)
    - [Publishing a web service](#)
  - [Currency rate calculator](#)
  - [GroovyWS grape config file](#)
  - [GroovyWS installation](#)
  - [Publishing a web-service](#)
  - [TerraServer-USA by Microsoft](#)
  - [Using the Aegis mapping](#)
  - [Using the client with complex objects](#)

- [Using the Grails XFire plugin and GroovyWS](#)
- [Using WSCClient in Grails](#)
- [Using WS-Security](#)
- [WSCClient configuration](#)
- [GSP](#)
- [GSQL](#)
- [HTTP Builder](#)
- [JideBuilder](#)
- [MetaBuilder](#)
- [Native Launcher](#)
- [Proxy-o-Matic](#)
- [Windows NSIS-Installer](#)
- [Windows Services](#)
- [WingSBuilder](#)
- [XMLRPC](#)
- [GroovyFX](#)
  - [Layout in GroovyFX](#)
  - [SceneGraphBuilder](#)
  - [SceneGraphBuilder - containers](#)
    - [SceneGraphBuilder.anchorPane](#)
    - [SceneGraphBuilder.borderPane](#)
    - [SceneGraphBuilder.borderPane.bottom](#)
    - [SceneGraphBuilder.borderPane.center](#)
    - [SceneGraphBuilder.borderPane.left](#)
    - [SceneGraphBuilder.borderPane.right](#)
    - [SceneGraphBuilder.borderPane.top](#)
    - [SceneGraphBuilder.flowPane](#)
    - [SceneGraphBuilder.gridPane](#)
    - [SceneGraphBuilder.gridPane.column](#)
    - [SceneGraphBuilder.gridPane.row](#)
    - [SceneGraphBuilder.group](#)
    - [SceneGraphBuilder.hbox](#)
    - [SceneGraphBuilder.scene](#)
    - [SceneGraphBuilder.stackPane](#)
    - [SceneGraphBuilder.tilePane](#)
    - [SceneGraphBuilder.vbox](#)
  - [SceneGraphBuilder - controls](#)
    - [SceneGraphBuilder.accordion](#)
    - [SceneGraphBuilder.button](#)
    - [SceneGraphBuilder.checkBox](#)
    - [SceneGraphBuilder.choiceBox](#)
    - [SceneGraphBuilder.graphic](#)
    - [SceneGraphBuilder.hyperlink](#)
    - [SceneGraphBuilder.label](#)
    - [SceneGraphBuilder.listView](#)
    - [SceneGraphBuilder.passwordBox](#)
    - [SceneGraphBuilder.progessIndicator](#)
    - [SceneGraphBuilder.progressBar](#)
    - [SceneGraphBuilder.radioButton](#)
    - [SceneGraphBuilder.scrollBar](#)
    - [SceneGraphBuilder.scrollPane](#)
    - [SceneGraphBuilder.separator](#)
    - [SceneGraphBuilder.slider](#)

- [SceneGraphBuilder.splitPane](#)
- [SceneGraphBuilder.splitPane.dividerPosition](#)
- [SceneGraphBuilder.tableView](#)
- [SceneGraphBuilder.tableView.tableColumn](#)
- [SceneGraphBuilder.tabPane](#)
- [SceneGraphBuilder.tabPane.tab](#)
- [SceneGraphBuilder.textArea](#)
- [SceneGraphBuilder.textBox](#)
- [SceneGraphBuilder.textField](#)
- [SceneGraphBuilder.titledPane](#)
- [SceneGraphBuilder.titledPane.content](#)
- [SceneGraphBuilder.titledPane.title](#)
- [SceneGraphBuilder.toggleButton](#)
- [SceneGraphBuilder.toolBar](#)
- [SceneGraphBuilder.tooltip](#)
- [SceneGraphBuilder - effects](#)
  - [SceneGraphBuilder.blend](#)
  - [SceneGraphBuilder.blend.bottomInput](#)
  - [SceneGraphBuilder.blend.topInput](#)
  - [SceneGraphBuilder.bloom](#)
  - [SceneGraphBuilder.boxBlur](#)
  - [SceneGraphBuilder.colorAdjust](#)
  - [SceneGraphBuilder.colorInput](#)
  - [SceneGraphBuilder.displacementMap](#)
  - [SceneGraphBuilder.dropShadow](#)
  - [SceneGraphBuilder.gaussianBlur](#)
  - [SceneGraphBuilder.glow](#)
  - [SceneGraphBuilder.imageInput](#)
  - [SceneGraphBuilder.innerShadow](#)
  - [SceneGraphBuilder.lighting](#)
  - [SceneGraphBuilder.lighting.bumpInput](#)
  - [SceneGraphBuilder.lighting.contentInput](#)
  - [SceneGraphBuilder.lighting.distant](#)
  - [SceneGraphBuilder.lighting.point](#)
  - [SceneGraphBuilder.lighting.spot](#)
  - [SceneGraphBuilder.motionBlur](#)
  - [SceneGraphBuilder.perspectiveTransform](#)
  - [SceneGraphBuilder.reflection](#)
  - [SceneGraphBuilder.sepiaTone](#)
  - [SceneGraphBuilder.shadow](#)
- [SceneGraphBuilder - input](#)
  - [empty](#)
  - [SceneGraphBuilder.onKeyPressed](#)
  - [SceneGraphBuilder.onKeyReleased](#)
  - [SceneGraphBuilder.onKeyTyped](#)
  - [SceneGraphBuilder.onMouseClicked](#)
  - [SceneGraphBuilder.onMouseDragged](#)
  - [SceneGraphBuilder.onMouseEntered](#)
  - [SceneGraphBuilder.onMouseExited](#)
  - [SceneGraphBuilder.onMouseMove](#)
  - [SceneGraphBuilder.onMousePressed](#)
  - [SceneGraphBuilder.onMouseReleased](#)
- [SceneGraphBuilder - nodes](#)

- [SceneGraph.text](#)
- [SceneGraphBuilder.imageView](#)
- [SceneGraphBuilder.imageView.image](#)
- [SceneGraph.node](#)
- [SceneGraph.container](#)
- [SceneGraphBuilder - paint](#)
  - [Scenegraph.hsb](#)
  - [Scenegraph.linearGradient](#)
  - [Scenegraph.radialGradient](#)
  - [Scenegraph.rgb](#)
  - [Scenegraph.stop](#)
- [SceneGraphBuilder - shapes](#)
  - [SceneGraphBuilder.arc](#)
  - [SceneGraphBuilder.circle](#)
  - [SceneGraphBuilder.cubicCurve](#)
  - [SceneGraphBuilder.ellipse](#)
  - [SceneGraphBuilder.line](#)
  - [SceneGraphBuilder.path](#)
  - [SceneGraphBuilder.path.arcTo](#)
  - [SceneGraphBuilder.path.closePath](#)
  - [SceneGraphBuilder.path.cubicCurveTo](#)
  - [SceneGraphBuilder.path.hLineTo](#)
  - [SceneGraphBuilder.path.lineTo](#)
  - [SceneGraphBuilder.path.moveTo](#)
  - [SceneGraphBuilder.path.quadCurveTo](#)
  - [SceneGraphBuilder.path.vLineTo](#)
  - [SceneGraphBuilder.polygon](#)
  - [SceneGraphBuilder.polyline](#)
  - [SceneGraphBuilder.quadCurve](#)
  - [SceneGraphBuilder.rectangle](#)
  - [SceneGraphBuilder.SVGPath](#)
- [SceneGraphBuilder - stage](#)
  - [SceneGraphBuilder.popup](#)
  - [SceneGraphBuilder.stage](#)
- [SceneGraphBuilder - transforms](#)
  - [SceneGraphBuilder.affine](#)
  - [SceneGraphBuilder.rotate](#)
  - [SceneGraphBuilder.scale](#)
  - [SceneGraphBuilder.shear](#)
  - [SceneGraphBuilder.translate](#)
- [SceneGraphBuilder - web](#)
  - [SceneGraph.htmlEditor](#)
  - [SceneGraph.webEngine](#)
  - [SceneGraph.webView](#)
- [TimelineBuilder](#)
  - [TimelineBuilder.timeline](#)
  - [TimelineBuilder.timeline.at](#)
  - [TimelineBuilder.timeline.at.change](#)
- [Gaelyk](#)
- [GMaven](#)
- [Groovy Transforms](#)
  - [Scalify](#)
- [Related Projects](#)

## Support

- [Mailing Lists](#)
- [Issues](#)
- [Change log](#)
- [Roadmap](#)

## Community

- [Contributing](#)
- [User Groups](#)
- [Forums and Newsgroups](#)
- [Roadmap, Discussions and Proposals](#)
- [Related Projects](#)
- [Modules](#)

## Contributing

There are many ways you can help make Groovy a better piece of software - and we can use all the help we can get. Please dive in and help! 😊

Try surfing the documentation - if something is confusing or not clear, let us know - or better still fix it for us. All of this website is maintained in [a wiki](#), so please go right ahead and fix things if they are wrong or contribute new documentation.

Download the code & try it out and see what you think. Browse the source code. Got an itch to scratch, want to tune some operation or add some feature?

Want to do some hacking on Groovy? Try surfing the issue tracker ([see below](#)) for open issues or features that need to be implemented, take ownership of an issue and try to fix it.

- [If you find a bug or a problem](#)
- [Contributing Code](#)
- [Using the issue tracker](#)
- [Improvements that need your help](#)

### If you find a bug or a problem

Please raise a new issue in our issue tracker ([see below](#)).

If you can create a JUnit test case (either via Groovy or Java code) then your issue is more likely to be resolved quicker. Take a look at some of the [existing unit tests cases](#), find one and modify it to try reproduce your problem.

Then we can add your issue to [Git](#) and then we'll know when its really fixed and we can ensure that the problem stays fixed in future releases.

### Contributing Code

The preferred way of contributing code to Groovy is by submitting pull request to [the project on Github](#).

The best workflow for this is to raise a new issue in the tracker ([see below](#)), fix the code in a branch on your forked project, and submit a pull request via the Github UI. Be sure to mention the JIRA issue id in the pull request, and add the pull request id to the JIRA ticket so that both are tied together. This workflow is described with more detail [in a blog post here](#).

Instead of pull request we also accept patches. Please attach these to a JIRA issue

Whenever possible please submit enhancements and bugfixes with accompanying unit tests.

## Using the issue tracker

Due to JIRA spam-bots you have to register with Codehaus Xircles before accessing the [issue tracker](#). Follow these steps:

1. Open the [Xircles Signup](#) page.
2. Pick a username/password.
3. Get the verification email and verify yourself.
4. You are now logged into [xircles.codehaus.org](#). Note that there is no link to this page from the [codehaus.org](#) main page.
5. Open the [issue tracker \(JIRA\)](#) page.
6. In the upper-right corner, click "log in" (you must log into JIRA separately from Xircles).
7. Use your Xircles username/password. You should now be logged into JIRA.

See <http://groovy.329449.n5.nabble.com/How-can-register-in-Groovy-s-JIRA-td5713775.html> for additional details/help.

## Improvements that need your help

Here's a [list of some issues contributors could have a look at](#), to get started contributing to Groovy. You can also contribute to unassigned issues/enhancements without the "contrib" tag, just add the comment about your intentions in this issue and start working on it.

The following table is a summary of the fields where we would like to improve Groovy but do not find enough time to implement those features. For each of them, you will find a short description, an idea of the skills you need to implement them and the level of difficulty. Of course, getting into Groovy Core can be sometimes hard, so we should always work as a team and each subject below would be followed by a mentor.

Feature	Description	Skills	Difficulty	Mentor	Status
---------	-------------	--------	------------	--------	--------

Grammar rewrite	The current Groovy grammar is written using ANTLR 2. For maintenance reasons, we would like to migrate the grammar to ANTLR 3 or ANTLR 4. Originally planned for Groovy 3, we will probably not be able to do that if no one helps. It is an important background task which covers the grammar of Groovy, the grammar of Java and a good knowledge of the Groovy AST (for backwards compatibility).	Grammar writing (Antlr helps a lot!), parsers, Groovy AST (nice to have)	Hard	Jochen Theodorou	Delayed
Bytecode optimizations	Since Groovy 2.0, we have two flavours of bytecode: the dynamic bytecode, corresponding to the main and legacy Groovy, and a "static" bytecode, triggered using the @CompileStatic annotation, which is closer to what Java produces. Both need specific bytecode optimizations.	ASM, JVM Bytecode	Hard	Cédric Champeau	In progress

<p>Javascript backend</p>	<p>Groovy runs on the JVM, but we would also like to see a Javascript backend. Of course, a prototype would have limited capabilities compared to what the JVM one offers, mostly because the Groovy language is not only a language but also a rich API relying internally on JDK APIs.</p>	<p>Javascript, Groovy compiler internals</p>	<p>Hard</p>	<p>Guillaume Laforge</p>	<p>Idea</p>
<p>Lambda support</p>	<p>JDK 8 will offer support for lambdas. Lambdas are close to what Groovy offers with closures, but the syntax, implementation and more importantly semantics are quite different. A future version of Groovy should be able to support both closures (Groovy) and lambdas (JDK 8).</p>	<p>JVM bytecode, grammar</p>	<p>Hard</p>	<p>Jochen Theodorou</p>	<p>Not started</p>

<p>Lazy datastructures and comprehensions</p>	<p>The idea would be to have Groovy support generators, lazy evaluations, ... See for example <a href="http://blog.bloidea.com/post/22117894718/groovy-stream-a-lazy-generator-class-for-groovy">http://blog.bloidea.com/post/22117894718/groovy-stream-a-lazy-generator-class-for-groovy</a></p> <p>It is also important to take a look at what APIs are being written in JDK 8 because most of the APIs written for the new lambdas are specifically lazy.</p>	<p>Algorithmics</p>	<p>Medium</p>	<p>Paul King</p>	<p>Prototype</p>
<p>New MOP</p>	<p>The new MOP (Meta Object Protocol) is a key feature of Groovy 3. The objective is to rewrite the MOP totally, in order to provide a more consistent, powerful and performant MOP. See <a href="http://groovy.codehaus.org/MOP+2.0+ideas">http://groovy.codehaus.org/MOP+2.0+ideas</a></p>	<p>Language theory</p>	<p>Medium</p>	<p>Jochen Theodorou</p>	<p>In progress</p>

Bugfixes	<p>As the language is widely used, we have a lot of bug reports. Unfortunately, the team is too small to tackle them all. If you find a bug that you think you can work on, do not hesitate to help!</p>		Easy to hard	any member of Groovy core	In progress
Modularization	<p>Groovy 2 came with the concept of modularization and is now splitted into modules. Each module is a logical group of features (for example, XML, JSON, Swing, ...). While Groovy core was reduced by half, there's still plenty of room for improvement and further modularization.</p>	Architecture	Easy	Paul King	In progress

Documentation	<p>On every project, documentation is never enough, nor up-to-date. Groovy falls into that trap too. A very good way to get into Groovy is to start documentation, fix documentation or improve javadocs. There are also side projects like a new website, a new Groovy Web Console which require some time but were never started.</p>		Easy	Guillaume Laforge	In progress
Specification	<p>Help joining the specification effort. The Groovy language has evolved a lot and there's no complete specification available yet.</p>		Medium	Guillaume Laforge	Standby

Security	Help us improving java.security support in Groovy. We have, for example, unit tests failing periodically because of security tests (Security Manager related). No one in the team is an expert in that domain so any help would be very appreciated!	Java security	Medium	Cédric Champeau	Standby
----------	--	---------------	--------	-----------------	---------

Groovy for Android	<p>A lot of us would like to see Groovy run on Android. There are some technical challenges but the good news is that some hackers (Marcin Erdmann and Erik Pragt) managed to do it for Groovy 1.7. It's not production ready yet, and we would like to have, at some point, support in the Groovy distribution itself (aka, no patch required). Running Groovy is one thing, doing something with it is another, so a side project for this is definitely to make Groovy a language of choice to build Android applications. We think some projects like Griffon, for example, have a strong interest in that, by making the design of UIs much easier.</p> <p>See <a href="http://skillsmatter.com/podcast/groovy-grails/groovy-android">http://skillsmatter.com/podcast/groovy-grails/groovy-android</a></p>	Android, JVM	Medium	Cédric Champeau	Prototype
--------------------	---	--------------	--------	-----------------	-----------

Groovy Console	The Groovy Console is a very nice tool. However, there are several things that can be improved: syntax highlighting, especially for large scripts, which is too slow; the AST Browser which can be improved in several ways (displaying node metadata, more focused bytecode view, ...)	Swing, Groovy	Easy to medium		In progress
----------------	---	---------------	----------------	--	-------------

## Forums and Newsgroups

Web forums and usenet newsgroups where you can get help. If a forum supports more than one language it is listed under its primary language. Other languages are listed in parentheses.

### English

- The English Groovy [Groovy-Nabble](#)

### Polish

- The Polish Groovy and Grails forum [forum.groovydev.org](http://forum.groovydev.org)

### Spanish

- Spanish web forum [groovy.org.es](http://groovy.org.es)

### French

- The french [Groovy](#) and [Grails](#) forums

## Mailing Lists



### Important notice to access the mailing-lists

Here are the public mailing lists that have been set up for the project. In order to subscribe/unsubscribe you first need to get a special link from [http://xircles.codehaus.org/manager\\_email](http://xircles.codehaus.org/manager_email). Once you have got this link you can manage your subscriptions by going to the Groovy project page <http://xircles.codehaus.org/projects/groovy>.

**announce@groovy.codehaus.org**

is a low volume list for announcements about new releases or major news

<b>dev@groovy.codehaus.org</b>	is a medium volume list useful for those interested in ongoing developments
<b>eclipse-plugin-user@groovy.codehaus.org</b>	is a low volume list for questions and problems using Groovy-Eclipse
<b>jsr@groovy.codehaus.org</b>	is a low volume list discussing standardisation of the language through the JCP
<b>scm@groovy.codehaus.org</b>	is a high volume list that logs commits and issues
<b>user@groovy.codehaus.org</b>	is a high volume list for questions and general discussion about Groovy
<b>Portuguese / Spanish</b>	is a list for Spanish and Portuguese speakers ( <a href="#">subscribe</a> <a href="#">unsubscribe</a> <a href="#">archive</a> )

You can find a great archive support on [Nabble](#) and at [MarkMail](#).

## User Groups

[Web Forums](#) can now be found on a separate page.

### Asia

- [Indian Groovy & Grails User Group](#)
- [Japan Grails/Groovy User Group](#)
- [Philippine Groovy/Grails Users Group](#)
- [Java User Group - Chennai](#)

### Africa

- none

### North America

- [Vancouver Groovy and Grails User Group](#) (British Columbia, Canada)
- [Groovy Users of Minnesota](#)
- [Chicago Groovy Users Group](#)
- [Inland Empire Groovy Users Group](#) (Southern California)
- [New York Groovy / Grails Meetup](#)
- [Central Pa Java Users Group](#)

### South America

- [Grails Brasil - Groovy and Grails User Group](#)

### Europe

- [London Groovy and Grails User Group](#)
- German user group [Special Interest Group \(SIG\) "Groovy" in der JUGS \(Java User Group Stuttgart\)](#)
- There are plans to open a second German User Group in Munich, we will use this Wiki page to inform you

about first dates. So far this is the plan:

- proposed meeting space is Actionality, Schellingstraße 45, Munich. Exit the underground station "Universität" and walk down the "Schellingstraße" to get there. Open for other places in Munich.
- Topics would include anything around Groovy: Groovy discussions (Features), Web Frameworks using Groovy (Grails, Aero, Rife), anything you want to talk about
- propose schedule: once a month, evening from 18:00 to 19:00 or open end - we still have to agree on exakt dates
- [Paris Groovy Grails User Group](#)
- There is also a focus on GROOVY in the [JUGHH \(Java User Group Hamburg\)](#)
- A first meeting is coming up in [April 2008](#). Maybe this would be the first step to a user group in Hamburg.
- Check out our meeting dates. We're preparing a GROOVY and GRAILS topic for one of our next meetings.
- [Italian User Group](#)
- [Belgian Grails and Groovy User Group](#): First meeting held at [ixor](#) on [2009 March 19](#)
- [Germany: Munich Groovy, Grails & Griffon User Group](#)
- [Slovak Groovy User Group](#)
- [Croatian Groovy & Grails User Group](#)
- Norway: [Trondheim Groovy Group](#)
- Norway: [Oslo Groovy Meetup](#)
- [Madrid GUG: Groovy/Grails & Griffon User Group](#) from Spain

## Australia

- [Melbourne Groovy User Group](#) held at the [Aegeon](#) offices on the 1st Monday of each Month.
- [Groovy & Grails Queensland](#)

## Modules

The following modules and contributions are currently available:

- [COM Scripting](#) — script Windows ActiveX and COM components with Groovy
- [Gant](#)
- [GFreeMarker](#) — an integration of the FreeMarker template engine for Groovy
- [Google Data Support](#) — makes using the Google Data APIs easier from within Groovy
- [Gram](#) — a simple xdoclet-like tool for processing doclet tags or Java 5 annotations
- [GraphicsBuilder](#) — GraphicsBuilder is a Groovy builder for Java 2D
- [Grapplet](#)
- [Griffon](#) — Dekstop Enhancements for Groovy
- [Groosh](#) — Provides a shell-like capability for handling external processes.
- [Groovy Jabber-RPC](#) — allows you to make XML-RPC calls using the Jabber protocol
- [GroovyJMS](#)
- [GroovyLab](#) — Provides a domain specific language (DSL) for math engineering (matlab-like syntax).
- [Groovy Monkey](#) — is a dynamic scripting tool for the Eclipse Platform
- [GroovyRestlet](#) — Groovy DSL for constructing Restlet application
- [Groovy Science](#)
- [Groovy SOAP](#) — create a SOAP server and make calls to remote SOAP servers using Groovy
- [GroovySWT](#) — a wrapper around SWT, the eclipse Standard Widget Toolkit
- [GroovyWS](#) — GroovySOAP replacement that uses CXF and Java5 features
- [GSP](#) — means GroovyServer Pages, which is similar to JSP (JavaServer Pages)
- [GSQL](#) — supports easier access to databases using Groovy
- [HTTP Builder](#) — provides a convenient builder API for complex HTTP requests
- [JideBuilder](#) — JideBuilder is a Groovy builder for the open source JIDE Common Layer

- [MetaBuilder](#) — MetaBuilder is a builder that builds builders.
- [Native Launcher](#) — a native program for launching groovy scripts
- [Proxy-o-Matic](#) — Proxy-o-Matic lets you create dynamic proxies fast and in an homogeneous way
- [Windows NSIS-Installer](#) — a Windows-specific installer for Groovy
- [Windows Services](#) — framework for Groovy-based WinNT (Windows) Services
- [WingsBuilder](#) — WingsBuilder is a Groovy builder for the wingS Framework
- [XMLRPC](#) — allows you to create a local XML-RPC server and/or to make calls on remote XML-RPC servers
- [GroovyFX](#) — GroovyFX provides a Groovy binding for JavaFX 2.0.
- [Gaelyk](#) — Gaelyk is a lightweight Groovy toolkit for developing and deploying Groovy applications on Google App Engine.
- [GMaven](#)
- [Groovy Transforms](#) — Provides additional AST Transformations
- [Grails](#) — a Groovy-based web framework inspired by Ruby on Rails
- [GORM](#) — the Grails Object-Relational Mapping persistence framework
- [Griffon](#) — a Groovy-based desktop framework inspired by Grails
- [GroovyPlugin](#) — A Groovy plugin for JSPWiki
- [OCM Groovy](#) — A Object Content Mapping implemented in Groovy leveraging the Java Content Repository (JCR)
- [Tellurium](#) — A test framework built on top of the Selenium test framework implemented in Groovy
- [Gradle](#) — The next generation build system
- [GPars](#) — A Groovy concurrency/parallelism library
- [CodeNarc](#) — Static analysis for defects, bad practices, inconsistencies and style issues in Groovy code.
- [GroovyServ](#) — GroovyServ makes Groovy's startup time much faster, by pre-invoking Groovy as a server.
- [GBench](#) — The benchmarking module for Groovy
- [GProf](#) — The profiling module for Groovy

## COM Scripting



### Under Construction

The **Scriptom** documentation is changing to reflect the new **Maven** build and artifacts. This is going to take some time. In the meantime, please take what you read with a grain of salt, and pardon the messiness. Rest assured that while there will be some restructuring of the documentation, the previous builds will still be available for anyone who needs them, ongoing, indefinitely.

## Introduction

**Scriptom** is an optional Groovy module originally developed by [Guillaume Laforge](#). It combines the elegant "syntactical sugar" of Groovy with the power of the [Jacob library \(Java COM Bridge\)](#). **Scriptom** lets you use **Active X** or **COM** Windows components from Groovy. The result is something that looks eerily similar to *VBScript* - only groovier.

You can use **Scriptom** to automate **Word** or **Excel** documents, control **Internet Explorer**, make your PC talk using the **Microsoft Speech API**, monitor processes with **WMI** (Windows Management Instrumentation), or browse the **Windows Registry** using **WShell** - and much more. **Scriptom** also provides an easy way to talk to custom **VB6** or **Microsoft.NET** libraries.

Of course, **Scriptom** can be used only on **Microsoft Windows**.

**Scriptom** is included as an option in the [Windows Installer](#), and **Scriptom** can be downloaded from this page (see below). The **Scriptom** codebase is stable and feature-complete. The **Jacob** project - **Scriptom**'s foundation - was

started in 1999 and is being used in countless production Java applications worldwide. **Scriptom** is now in wide use as well, and has proven to be stable and mature.

**Scriptom** gives you all the **COM**-scripting power of **Jacob**, only it is a lot easier. See [Getting Started with COM](#) for some tips to get you started.

## Requirements

The following are required to run **Scriptom**:

- **Groovy 1.5** or **1.6**
- **Java 1.5** or later
- **Windows** and **Java** for **x86** or **x64 (AMD64)** architectures. **Itanium (IA-64)** is not supported.

## Maven-Generated Documentation

Maven-generated documentation is available at <http://groovy.codehaus.org/modules/scriptom/1.6.0/index.html>. This contains useful information from the build, including JavaDoc for all projects.

## Installation

**Scriptom** is part of the [Windows Installer](#). If you are running Groovy scripts on **Windows**, you are all set.

### "Just the (Arti)facts, Ma'am."

If you are running Groovy outside the **Windows Installer**, and you aren't using **Maven** or **Grape**, you probably just need the pre-packaged JAR files and associated binaries. You've come to the right place.

Note that as of release **1.6.0**, the distribution archive is formatted differently. The new format matches the folder structure needed by the **Windows Installer**. Source code, build scripts, and documentation are no longer included in the distribution.

- ★ [Scriptom 1.6.0](#) (4/8/2009) - No new features or bug fixes. Distribution for **Windows Installer**. Includes only compiled artifacts. Supports **Groovy 1.5** and up.
- [scriptom-1.5.4.zip](#) Build 12 (12/26/2008) - Full project including source. Supports **Groovy 1.5** and up.

## Adding Scriptom to a Maven Project

You can add **Scriptom** to a **Maven** project. Because there is JNI involved, it isn't quite as simple as adding a dependency, but it is doable. **Need to come up with an example of creating an assembly with GMaven.**

1. Add the **Scriptom** dependency to your **Maven** project.

```
<dependency>
  <groupId>org.codehaus.groovy.modules.scriptom</groupId>
  <artifactId>scriptom</artifactId>
  <version>1.6.0</version>
</dependency>
```

2. Add the **Jacob** JAR dependency. This JAR must be loaded only once, so if you are working in a server application like **Tomcat** or **GlassFish**, you need to ensure that **Jacob**'s JAR is only loaded by the system classloader. **Need more explicit instructions on this...**

```
<dependency>
  <groupId>net.sf.jacob-project</groupId>
  <artifactId>jacob</artifactId>
  <version>1.14.3</version>
  <type>jar</type>
</dependency>
```

3. **Jacob** requires a DLL. It's a JNI project, after all. But **Jacob** actually supports two versions of the DLL it needs, one for 32-bit x86 systems, and one for the AMD x64 architecture (works with 64-bit Intel chips too). The easiest way to get this to work is to put both DLLs somewhere on the system path. **Jacob** will automatically pick the one it needs. **Need to detail the other ways to specify the Jacob DLL.**

```
<dependency>
  <groupId>net.sf.jacob-project</groupId>
  <artifactId>jacob</artifactId>
  <version>1.14.3</version>
  <type>dll</type>
  <classifier>x64</classifier>
</dependency>
<dependency>
  <groupId>net.sf.jacob-project</groupId>
  <artifactId>jacob</artifactId>
  <version>1.14.3</version>
  <type>dll</type>
  <classifier>x86</classifier>
</dependency>
```

**Need to detail the TLB projects, constants, etc.**

**The Old Way to Install (deprecated)...**

**Download** the project archive and extract the files.

- 👍 [scriptom-1.5.4.zip](#) Build 12 (12/26/2008) - **Scriptom** JAR file, **Scriptom-1.5.4bX-32.dll**, **Scriptom-1.5.4bX-64.dll**, Java source code, ANT build script, examples, tests, utilities, and documentation.

Install the jar file and DLL file(s) into your project, and optionally install an update from Microsoft:

1. Add the **Scriptom** jar file (**scriptom-1.5.4bX-XX.jar**) into your Java classpath. It contains both **Scriptom** and **Jacob** class files, so you must not include **jacob.jar**.

⊖ **Scriptom** contains a full (current) version of the **Jacob** library. If you are using **Jacob** directly elsewhere in your project, remove all references to it and use the version built in to **Scriptom**.

2. Copy both **Scriptom-1.5.4bX-XX.dll** files to somewhere on your **java.library.path**. (usually somewhere on the system 'PATH').

 The **Scriptom DLL** naming convention allows multiple versions of **Scriptom** to run on the same machine. However, you can only run a single version of **Scriptom JAR** in any given project.

3. To avoid the dreaded **java.lang.UnsatisfiedLinkError**, download and install one of the following updates from Microsoft: [Microsoft Visual C++ 2005 SP1 Redistributable Package \(x86\)](#) or [Microsoft Visual C++ 2005 SP1 Redistributable Package \(x64\)](#). **Scriptom** doesn't support the **IA-64 (Itanium)** architecture at this time, mainly due to lack of interest. If you are wondering about the different processor architectures, check out the [x86-64 wiki](#). It is usually necessary to install these updates on **Windows Server 2003** and **Windows 2000**, and we've found that it may also be necessary for **Windows XP** and even **Vista**.

 A project can only use one version of **Scriptom** at a time. If you have installed **Groovy** using the **Groovy Windows Installer**, you must remove any versions of **Scriptom JAR** files or replace them with the latest version. A version of **Scriptom** is installed as part of the **Groovy Windows Installer** installation.

**Scriptom 1.5** is not supported for **Groovy 1.0** and earlier (**Scriptom 1.2** is still available).

## Building from Source

The project source is managed by **Subversion**. The projects are already set up to work with **Eclipse**, but it isn't hard to get them working with other IDEs, and you can get by with Notepad. The project trunk is located at <http://svn.codehaus.org/groovy/modules/scriptom/trunk>. This is a **Maven** build. Because the tests are dependent on **Windows** technologies, there are some rather strange software requirements:

- **Windows XP** or later.
- **Visual C# Express 2008** or **Visual Studio 2008**
- **Java 1.5.x** or later. **Java 1.6.x** recommended.
- **Maven 2.0.9** or better.
- A **Subversion** client. I recommend you get one or both of:
  - **Tortoise**
  - **SlikSVN**

To build, use the following command line:

```
C:\work\groovy\modules\scriptom\trunk>mvn  
clean install
```

The latest build requires **Groovy 1.5** or better.

**Scriptom** includes source files and compiled DLLs (Windows libraries compiled from C++) from **Jacob** version **1.14**.

You don't need to download sources and binaries from the **Jacob** project on **SourceForge.net**, though you can do so if you need to build the entire project yourself.

To build the project on Windows, run **project/build/make.bat**. Since **Scriptom** is a Windows-specific module, there isn't a shell script for building on other operating systems.

The build process requires **Java 1.5** or higher (**Java 1.6** recommended) and **ANT 1.6.5** or better.

## Changes from Scriptom 1.2

**Scriptom 1.5** is a substantial upgrade to previous versions of **Scriptom**, and *is not backward compatible*. We hope you will agree that it is worth a little code rework to get all these great new features! **Scriptom 1.2** is the version that is documented in *Groovy in Action*.

- Provides simplified helper methods for dealing with COM apartment threading models.
- **ActiveXProxy** name changed to **ActiveXObject** to match WSH convention.
- **VariantProxy** no longer exists (the functionality was refactored into **ActiveXObject**).
- Variant values are converted to and from the equivalent Java type; no more calling **.value** on everything!
- Supports COM event callbacks using closures (this is still a work in progress, but usable as is).
- Supports indexed properties, including multiple indexes.
- Supports typed, multi-dimensional SafeArrays.
- Supports pass-byref (in/out) values, both in method calls and event handler callbacks.
- Supports COM enumerable types (so you can use **.each**, for example).
- Supports missing arguments (for optional parameters) in method/property calls and events.
- Includes constant definitions for several popular Windows applications, derived from these type libraries (see JavaDoc):
  - Microsoft Office 2003 (works with other versions as well)
    - Word 2003
    - Excel 2003
    - Outlook 2003
    - Access 2003
    - PowerPoint 2003
  - Microsoft Internet Explorer 6 (works with other versions as well)
  - Microsoft Scripting (i.e., FileSystemObject)
  - Microsoft Speech API
  - Windows Scripting Host (WSH)
  - WbemScripting (WMI - Windows Management Instrumentation)
- Supports the latest features from **Jacob 1.14**, including new support for the **Decimal** data type and 64-bit integers.

## Migrating from Previous Versions of Scriptom

**Scriptom 1.5** is not backward compatible with previous versions of **Scriptom**. To get your scripts running again, do this:

- Change all references to **ActiveXProxy** into **ActiveXObject**.
- Wrap any code that references an **ActiveXObject** in **Scriptom.inApartment { ... }**, which replaces the way Scriptom previously handled COM threading.
- Remove all references to **.value** property references.
- Remove any statically typed references to **VariantProxy**.
- Support for COM events is greatly improved in this version - and also greatly changed. Refer to the **Events** section below for more information.

## Quick Start

Let's say you want an application that talks. Pure Java implementations aside (this is, after all, a Groovy/COM tutorial), and ignoring the fact that the default voice on pre-Vista machines sounds like [Dr. Zoidberg](#) with a sinus infection, you could use the **Microsoft Speech API (SAPI)** to get the job done.

You start by creating an **ActiveXObject** with the prog-id for **SpVoice**. Now you can call any of the methods it supports. By default, **SpVoice** will block until it is done speaking, but we're also going to have it speak

asynchronously and wait until it is done.



```
import org.codehaus.groovy.scriptom.*
import static
org.codehaus.groovy.scriptom.tlb.sapi.Speech
VoiceSpeakFlags.*
import static
org.codehaus.groovy.scriptom.tlb.sapi.Speech
RunState.*

//Definitive proof that you CAN talk and
chew gum at the same time.
Scriptom.inApartment
{
    def voice = new
ActiveXObject('SAPI.SpVoice')

    //This runs synchronously.
    voice.speak "Hello, GROOVY world!"

    //This runs asynchronously.
    voice.speak "GROOVY and SCRIPT um make com
automation simple, fun, and groovy, man!",
SVSFlagsAsync
    while(voice.Status.RunningState !=
SRSEDone)
    {
        println 'Chew gum...'
        sleep 1000
    }
}
println 'Speaker is done.'
```

If you have scripted COM before, you are probably used to using "magic numbers" throughout your code in place of COM constants. In this code sample, we're using fully-qualified constants instead.

**Scriptom** includes fully JavaDoc'd constant and interface definitions from a number of commonly used type-libraries, and you can even create your own. The source code for generating COM type library definitions for **Groovy** is written in **Groovy** 😊, and it's included in the project. It may not seem like a big deal to replace a couple of numbers, but it will be a lot easier in 10 years to find relevant information on **SpeechVoiceSpeakFlags.SVSFlagsASync** than on the number **1** (Google returns a little more than 9 billion hits for the number '1', and about 1,000 for 'SpeechVoiceSpeakFlags.SVSFlagsAsync,' including hits on this paragraph). And besides, the code reads better.

Speaking of interfaces, it turns out that **SpVoice** supports several. You can test an **ActiveXObject** to see if it supports a given interface using **.supportsInterface**, and you can cast an **ActiveXObject** to a given interface using **.toInterface**.

This next example displays the COM interfaces that **SpVoice** supports (within the **SAPI** library only):

```
import org.codehaus.groovy.scriptom.*;
import
org.codehaus.groovy.scriptom.tlb.sapi.Speech
Lib;

Scriptom.inApartment
{
    def voice = new
ActiveXObject('SAPI.SpVoice')
    SpeechLib.interfaces.each {name, iid ->
if(voice.supportsInterface(iid)) println
"SpeechLib.$name - $iid"}
}
```

## Programmer's Guide

<a href="#">The Least You Need to Know about COM</a>	how not to crash the JVM or lock your process
<a href="#">COM Data Types in Scriptom</a>	supported types and conversions in Java, VB6, and VB.NET
<a href="#">COM Methods and Properties in Scriptom</a>	calling COM methods and properties from Groovy

<a href="#">Passing Values by Reference (in-out)</a>	how to let methods change the value of passed-in arguments
<a href="#">COM Events</a>	support for COM events
<a href="#">All About Arrays</a>	UNDER CONSTRUCTION

## Examples

Here is a simple example that uses the Microsoft **ScriptControl** to evaluate a JScript expression. This is a very indirect way to add 2 and 2.

```
import org.codehaus.groovy.scriptom.*

Scriptom.inApartment
{
    def scriptControl = new
ActiveXObject("ScriptControl")
    scriptControl.Language = "JScript"
    println scriptControl.Eval('2.0 + 2.0;')
}
```

There are many, many potential uses for **Scriptom** - far to many to try to maintain as part of this documentation. So we've included a whole slew of meaty examples in the project archive for you to play with. We'd like to encourage you to go look at those examples, run them, and modify them. And if you come up with an especially interesting example, let us know about it. We may even include it in a future release!

Some additional examples included with **Scriptom**:

- Automated writing to and reading from **Excel** spreadsheets. Includes COM events example.
- Navigation in **Internet Explorer**. Includes COM events example.
- Gathering information about processes and Windows NT services using **WMI**.
- Parsing a \*.msg file with **Microsoft Outlook**.

Consuming **Visual Basic 6** (VB6) and **Visual Basic.NET** COM-enabled DLLs.

## Articles

Articles about **COM scripting** in general, and **Scriptom** in particular.

<a href="#">Getting Started with COM</a>	January 11, 2008	Tips for exploring <b>COM</b> libraries, finding documentation, etc.
--	------------------	--

<a href="#">Bridging the Gap Between Java and .NET with Groovy and Scriptom</a>	December 3, 2007	Implement <b>FIPS 140-1</b> compliant <b>S HA-1</b> in <b>Java</b> with <b>Groovy</b> , <b>Scriptom</b> , and <b>Visual Basic.NET</b> .
<a href="#">Juixe TechKnow</a>	August 8, 2006	A short <b>Microsoft Outlook</b> example.
<a href="#">Siebel</a>	March 17th, 2008	Siebel "business component" programming

## Post Scriptom

All known (unresolved) issues and feature requests are listed in the [Scriptom Jira](#) database.

Changes to each build are summarized in the [Change Log](#).

Recent builds of **Scriptom** can be found [here](#). Older versions are archived:

<a href="#">Scriptom-1.5.zip</a>	Build 10. Bundled with the <b>Groovy 1.5 Windows Installer</b> . Requires <b>Groovy 1.5</b> and <b>Java 5</b> or better.
<a href="#">Scriptom 1.2</a>	(also <b>Scriptom 1.1</b> and <b>Scriptom 1.0</b> ) Original version, and the one that is documented in <b>Groovy in Action</b> . Requires <b>Groovy 1.0</b> or better and <b>Java 1.4</b> or better.

## Progids

**Scriptom** uses late binding. Visual Basic and VBA uses early binding by default, but they also support late binding (with **CreateObject**). If you are translating working Visual Basic code and if your **Scriptom** code fails at the point where you've got your call

```
ActiveXObject oSDO = new  
ActiveXObject ("SageDataObject140.SDOEngine")
```

(or whatever your object is) with an error message

**Caught:**

**org.codehaus.groovy.scriptom.ActiveXObject\$C  
reationException:**

**Could not create ActiveX object:**

**'SageDataObject140.SDOEngine'; Can't  
get object clsid from progid**

then the chances are the progid "Sage...." that you are using is the wrong one.

So you need to go and look in the registry to find what it might be. One way to do this is with Microsoft's new Powershell cmd utility which you can download from the ms web site. Install this and run the command line

```
dir REGISTRY::HKEY_CLASSES_ROOT\CLSID  
-include PROGID -recurse | foreach  
{$_ .GetValue( " " )}
```

It will produce a ream of different progids which you can sort and search for one that looks a likely candidate. In my case it was SDOEngine.14

Alternatively if you have used the scriptom utility ExtractTlbInfo.groovy to generate name maps for the com object you could read the source code for the (in my case SageDateObject140.java) class and you might find some code and/or comment like this:

```

/**
 * A {@code Map} of CoClass names to
 prog-ids for this type library.<p>
 *
 * Note that some objects that support
 events do not publish a prog-id.
 * This is a known limitation of this
 library that we hope to resolve in
 * a future release.<p>
 *
 * Supported prog-ids:
 * <ul>
 *   <li><b>SDOEngine</b> =
 SDOEngine.14</li>
 * </ul>
 */
public final static Map progIds;
static
{
    TreeMap v = new TreeMap();
    v.put("SDOEngine", "SDOEngine.14");
    progIds =
Collections.synchronizedMap(Collections.unmo
difiableMap(v));
}

```

Additionally, you can use the *Object Browser* included with Visual Basic (I recommend the one with VBA) to figure out possible progids and then search the registry. This is a good way to associate interfaces (and their methods) with a particular progid. If there are *VBScript* or *JScript* examples available, they will include the correct progids, since these languages are always late-bound.

If all else fails, hit the Groovy mailing lists. There are lots of people out there with COM experience who can point

you in the right direction.

## All About Arrays

This article is under construction. You can find a good example of using an **org.codehaus.groovy.scriptom.SafeArray** in the **ExcelReport.groovy** example. This example demonstrates reading the value of a multi-cell **Range** as a 2-dimensional array (much faster than accessing the cells individually).

## Change Log

Old **Scriptom** builds can be found [here](#).

Version	Date	Release	Jacob	Changes
---------	------	---------	-------	---------

---

1.6.0	3/28/2009		1.14.3	<p>Compiled against <b>Groovy 1.6.0</b>. Project has been refactored to a <b>Maven 2</b> build to make it easier to maintain and easier to include in "enterprise" projects. Type libraries have been broken out into separate JARs. The core <b>Scriptom</b> JAR is once again very lightweight (about 50KB). <b>Jacob</b> classes are no longer bundled into <b>Scriptom</b>, making it easier to install into J2EE apps. <b>Jacob</b> DLLs now use their default names, making it easier to use <b>Jacob</b> with Java in parallel with <b>Scriptom</b> in Groovy. There is an assembly project that contains all the artifacts necessary to install into a non-Maven (non-Ivy) build - for use by the <i>Windows Installer</i>. This build contains no new features and no bug fixes.</p>
1.5.4 b12	12/26/2008	<b>Groovy 1.6.0 Windows Installer</b>	1.14 final	<p>Compiled against <b>Groovy 1.5.6</b>. Uses <b>Jacob 1.14 final</b>. Fixes a bug caused by a minor change in the <b>Groovy</b> APIs from <b>Groovy 1.5</b> to <b>Groovy 1.6</b>.</p>

1.5.4 b11	2/5/2008		1.14 final	Compiled against <b>Gr oovy 1.5.4</b> . Uses <b>Ja cob 1.14 final</b> . Exposes static methods on <b>Scripto m</b> object to allow <b>Ja va</b> code (using <b>Jaco b</b> directly) to manage <b>COM</b> threa ding in coordination with <b>Scriptom</b> .
<b>1.5.0 b10 Production</b>	12/10/2007	<b>Groovy 1.5 Windows Installer, Groovy 1.5.1 Windows Installer</b>	1.14M5 (modified)	<b>Really</b> fixed the bug introduced in <b>Build 8</b> that broke <b>Scripto m</b> for <b>Java 5</b> .
1.5.0 b9	12/8/2007		1.14M5	Fixed a bug introduced in <b>Build 8</b> that broke <b>Scripto m</b> for <b>Java 5</b> .

1.5.0 b8	12/7/2007		1.14M5	<p>The release version of <b>Groovy</b> will be <b>1.5</b>. <b>Scriptom's</b> major and minor version numbers are being changed to match the supported version of <b>Groovy</b>. Added version info to manifest. <b>Jar</b> file is marked as "sealed."</p> <p>Recompiled to <b>Groovy 1.1 rc3</b> and <b>Jacob 1.14 M5</b>. <b>Scriptom</b> no longer uses the <b>jacob.dll</b> because of the potential for naming conflicts with other versions on the same machine.</p> <p>The 32-bit and 64-bit <b>dlls</b> have been renamed so that they can coexist on the same machine - with other versions as well. <b>Scriptom</b> will automatically find the correct version of the <b>DLL</b> for the <b>JVM</b> you are running.</p>
2.0 b7	11/26/2007		1.14M5	<p>Refactored <b>EventArguments</b> into standalone class, from an inner class of <b>ActiveXObject</b> (this change should not normally affect existing code). Fix for bugs due to <b>Big Decimal-to-Decimal</b> conversion overflows. Improved support for unsigned integer types. Uses <b>Jacob 1.14 M5</b>.</p>

2.0 b6 (Beta)	11/15/2007		1.14M3	Refactored <b>org.codehaus.groovy.scriptom.tlb.office2003</b> packages to <b>org.codehaus.groovy.scriptom.tlb.office</b> . Added type-library constants packages from <b>Access 2003</b> and <b>PowerPoint 2003</b> .
2.0 b5 (Beta)	11/14/2007		1.14M3	Moved from <b>Jacob 1.14 M2</b> to <b>Jacob 1.14 M3</b> . Uses new <b>VT_I8</b> (64-bit signed integer) support. More unit testing.
2.0 b4 (Beta)	-	Groovy 1.1 rc2 Windows Installer	1.14M2	Fix for a <b>NullPointerException</b> that occurs only when debug tracing is turned on.

## COM Data Types in Scriptom

The following table summarizes the equivalent types between **Scriptom** and COM (VB6/VBA/.NET).

Groovy/Java	VB6/VBA	VB.NET	Comments
null	Empty/Nothing	Empty/Nothing	
<b>VariantNull</b>	Null	???	<b>VariantNull</b> is not the same as Java <b>null</b> . Use <b>Scriptom.NULL</b> .
boolean	Boolean	Boolean	
byte	Byte	Byte/SByte	8-bit signed or unsigned integer. 0 to 255 in the <b>COM</b> library. -128 to 127 in <b>Java</b> . Conversion happens automatically.
short	Integer	Short	16-bit signed integer. -32,768 to 32,767.

(int)	-	UShort	16-bit unsigned integer. 0 to 65,535.
int	Long	Integer	32-bit signed integer. -2,147,483,648 to 2,147,483,647.
(long)	-	UInteger	32-bit unsigned integer. 0 to 4,294,967,295.
long	-	Long	64-bit signed integer. -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. 64-bit integers are not supported by COM in Windows Millenium or Windows 2000 and earlier.
(BigInteger)	-	ULong	64-bit unsigned integer. 0 to 18,446,744,073,709,551,615. 64-bit integers are not supported by COM in Windows Millenium or Windows 2000 and earlier.
float	Single	Single	Low-precision floating point. -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
double	Double	Double	High-precision floating point. The range of a <b>Double</b> is -1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.

<b>BigDecimal</b>	-	Decimal	High-precision fixed/floating point (96 bits plus scaling), but without the large mantissa values supported by <b>Double</b> . <b>Scriptom</b> supports the <b>Currency</b> data type by converting to and from <b>Decimal</b> . The range of a <b>Decimal</b> is +/-79,228,162,514,264,337,593,543,950,335 for zero-scaled numbers (that is, numbers with no decimal places). For numbers with 28 decimal places, the range is +/-7.9228162514264337593543950335. The smallest possible non-zero number is 0.00000000000000000000000000000001. <b>BigDecimal</b> values with greater precision than <b>Decimal</b> supports are automatically rounded.
<b>BigDecimal</b>	Currency	-	Numbers from -922,337,203,685,477.5808 to 922,337,203,685,477.5807, with a fixed decimal point. <b>Scriptom</b> handles <b>Currency</b> values internally as type <b>Decimal</b> .
<b>Date</b>	Date	Date	
<b>String</b>	String	String	
<b>SafeArray</b>	Array	Array	<b>Scriptom</b> supports multi-dimensional arrays for COM.
<b>ActiveXObject</b>	A COM-callable object	A COM-callable object	

Note that implicit type conversions are supported, so you could - for instance - pass a **float** to a method that expects a **double**. **COM** unsigned integer types are converted to the next largest Java signed equivalent integer type so that large positive values do not cause overflows.

**⚠ CAUTION:** The default floating-point type in Groovy is **BigDecimal**, which gets converted to a **COM Decimal**. Although **Decimal** will work in the majority of cases, type conversions involving **Decimal** are inefficient compared to other numeric types. It is a good idea to explicitly coerce **Decimal** values to type **Double** or type **Integer**, whichever makes the most sense. Also note that **Decimal** does not handle the types of large mantissas that type **Double** can, but it supports more precision.

## COM Events

**Scriptom** lets you subscribe to COM events on an **ActiveXObject** using the **.events** metaproperty. The event handler is a **Closure**. The arguments in the **Closure** are passed through an instance of **EventArguments**. In this example, we're subscribing to the **Change** event of an Excel *Worksheet*. The **Change** event passes one argument - an Excel *Range*. In the context of this code, we're assuming that the *Range* object contains just one cell. In the general case, that would be a bad assumption. Once the event handler is defined this way, every change to a cell in the worksheet is going to print a line to standard out with the column, row, and new value.

```
worksheet.events.Change =
{args ->
    def range = args[0]
    println "\tEVENT Change
({range.Column},{range.Row}) =
{range.Value}"
}
```

## Byref Arguments

As with other COM method types, events support passing values by reference. This is particularly important in light of the fact that events don't return a value. If you want to be able to send data back to whoever raised the event, you have to pass values back to the sender using byref arguments. Fortunately, this is easy to do using the **EventArguments** instance.

Take, for example, the following class written in **Visual Basic 2005**. When you call **RaisePassBooleanByref**, it will raise the **OnPassBooleanByref** event, by reference. This gives us an opportunity to change the value of the boolean in the event handler!

```
Option Explicit On
Option Strict On
Option Compare Binary
```

```
<Microsoft.VisualBasic.ComClass()> Public
Class TestEvents
    Public Event OnPassBooleanByref(ByRef
Value As Boolean)

    Public Function
RaisePassBooleanByref(ByVal Value As
Boolean) As Boolean
        RaiseEvent OnPassBooleanByref(Value)
        Return Value
    End Function
End Class
```

In the following **Groovy** code, note how the return value changes after we define the event handler, which simply inverts the value of the first argument.

```
assert activeX.RaisePassBooleanByref(false)
== false
activeX.events.OnPassBooleanByref = {args ->
args[0] = !args[0]}
assert activeX.RaisePassBooleanByref(false)
== true
```

Note that you may only change a value if it has been passed by reference. Otherwise you get an exception.

#### Event Handlers for Office and Internet Explorer

Back in the first example, I defined an event handler for an Excel *Worksheet*. There is another piece of information

you need to make that example actually work. For many **COM** objects (and all true **ActiveX** objects), the underlying **COM** event handler is defined in a standard way that's easy to find and work with. However, for some **COM** objects, like those in **Microsoft Office** and **Internet Explorer**, there isn't enough information available for **Scriptom** to find the associated event interface. When this happens, you must manually define the prog-id of the object before you define any event handler. To help you out as much as possible, these are available as constants in the **Scriptom** library for the **Office** suite and for **Internet Explorer**. Here is how you define a prog-id for an **ActiveXObject**.

```
import
org.codehaus.groovy.scriptom.tlb.office.excel.Excel;

worksheet.events.useProgId
Excel.progIds.Worksheet
//Equivalent to worksheet.events.useProgId
'Excel.Sheet'
```

Additionally, there are some objects in **Office** and other applications where the event interface exists, but it cannot be discovered, and the object does not have an associated prog-id. The Excel *Workbook* object is one example of this. We're hoping to address this limitation in a future version of **Scriptom**.

## COM Methods and Properties in Scriptom

### Method Syntax

The syntax for calling a method on an **ActiveXObject** is (mostly) the same as calling a method on any other Groovy object. The only caveat is that optional parameters may be omitted, or you can specify **Scriptom.MISSING** to indicate that an optional parameter is not defined.

```
def voice = new
ActiveXObject('SAPI.SpVoice');
voice.Speak('It's all GROOVY, man!');
```

### Property Syntax

In Groovy, which follows the JavaBean model, properties don't have indexes. There are lots of COM properties without indexes, and they work the same way. In the code example below, the **voice.Status** property returns an **ActiveXObject**, and the **RunningState** property returns an integer flag.

```

def voice = new
ActiveXObject('SAPI.SpVoice')

voice.Speak 'GROOVY and SCRIPT um make com
automation simple, fun, and groovy, man!',
SpeechVoiceSpeakFlags.SVSFlagsAsync
while(voice.Status.RunningState !=
SpeechRunState.SRSEDone)
{
    ...
}

```

COM also supports parameterized (or 'indexed') properties. Indexed COM properties look like an array, list, or a map. Properties may be read from or written to, but they do not represent standalone objects (that's **not** an array you are working with). In some cases, properties have more than one index. The following practical example sets the value of the first column in an Excel spreadsheet to a date value (alternately, a Java **Date** could have been used).

```

worksheet.Cells.Item[row+1,1] =
"$row/1/2007"

```

Yep, that's a property!

#### Maybe We Made It too Flexible?

Methods and a property getters (both non-indexed and indexed) are pretty much the same thing, and in most cases, they are interchangeable. So the following two lines of code do the same thing:

```

println worksheet.Cells.Item[row+1,1]
println worksheet.Cells.Item(row+1,1)

```

Indexed properties don't work if all the parameters are optional. This doesn't normally happen in the real world (why would you do that?), but it is possible. To support this unlikely scenario, there is an alternate syntax for accessing

properties as methods:

- To get a property value, simply call it as a method. Alternately (to match *setter* syntax), you can prefix the property name with `_get`.

To set a property value, prefix the property name with `_set` or `_put`. The last argument passed to the method is the value.

```
//Getters, all equivalent.  
println worksheet.Cells.Item[row+1,1]  
println worksheet.Cells.Item(row+1,1)  
println worksheet.Cells._getItem(row+1,1)  
  
//Setters, all equivalent.  
worksheet.Cells.Item[row+1,1] = new Date()  
worksheet.Cells._setItem(row+1,1, new  
Date())  
worksheet.Cells._putItem(row+1,1, new  
Date())
```

The method syntax is actually a little faster than the bracketed property syntax, but we recommend that you use the syntax that makes your intentions most evident.

### Passing Values by Reference (in-out)

**Java** and **Groovy** are both "pass-by-value" languages. That is, if you pass a value to a method and the method modifies it, you don't see a change in the calling scope.

**COM**, on the other hand, supports both "pass-by-value" (or "byval") and "pass-by-reference" (or "byref"). "Byval" is sometime referred to as an "in" parameter, and "byref" is sometimes referred to as an "in/out" parameter, reflecting the direction the data is flowing. When you call a method that accepts a "pass-by-reference" parameter, the method can modify the parameter value, and this will be reflected in the calling scope.

Byref argument passing was always a favorite of **C/C++** programmers, who often use the result of a method to pass exception information. It was the default way to pass values in versions of **Visual Basic** through **6**. With the advent of modern try/catch exception handling, modifying values inside a method is generally considered bad programming practice. **Visual Basic** (and **COM**) still supports byref, but byval is now defined as the default setting. You should not have to deal with byref parameters often, especially when dealing with APIs that are designed for scripting. That doesn't mean it will never happen, though.

It shouldn't surprise you that **Scriptom** supports passing parameters by-reference, just in case you need it.

There are actually two ways that values are passed by-reference. For [COM Events](#), which do not support returning

values, the only way to pass back information is through a byref parameter. That is all explained in the article, and it is actually pretty transparent. You don't have to do anything special for it to work. Your event handler can just change one of the arguments passed to it, and the caller sees the changed value.

The other way that values are passed by reference is through method calls. This one is a little more complicated, but only a little. **Scriptom** takes care of most of the scary details for you.

### How Does it Work?

The basic concept is that you place the value in a wrapper object, pass the wrapper to the method, and finally get the new value back from the wrapper when the method returns. **Scriptom's** wrapper object is **VariantByref** ([org.codehaus.groovy.scriptom.VariantByref](http://org.codehaus.groovy.scriptom.VariantByref)).

One cool feature is that **Scriptom** allows you to just pass a **String** or an **Integer** or whatever type the method requires, if you don't care about getting back the changed value. Most of the time, even when a parameter is defined as byref, you don't really care about seeing the changed value. So you can ignore all of this.

Okay, so let's assume that you do care. The changed value matters to you. Otherwise you would have stopped reading in the last paragraph, right? Consider the following **VB6** object. It simply takes a byref **String** and modifies it.

#### Option Explicit

```
Public Sub StringByrefDemo(ByRef A As  
String)  
    A = A & " VB6 was here!"  
End Sub
```

The following example illustrates how to call the **VB DLL** so that the modified **String** is *not* lost.

```

Scriptom.inApartment
{
    def testObj = new
ActiveXObject("MyDemo.PassByrefDemo");

    VariantByref svalue = new
VariantByref("This is Groovy!");
    testObj.StringByrefDemo(svalue);
    assert svalue.value == "This is Groovy!
VB6 was here!";
}

```

That is really all there is to it. Again, you shouldn't have to use this feature very often. But when the API is designed around byref parameters, this can be a lifesaver!

### Scriptom Archive

<a href="#">Scriptom-1.5.zip</a>	Build 10. Bundled with the <b>Groovy 1.5 Windows Installer</b> . Requires <b>Groovy 1.5</b> and <b>Java 5</b> or better.
<a href="#">Scriptom 1.2</a>	(also <b>Scriptom 1.1</b> and <b>Scriptom 1.0</b> ) Original version, and the one that is documented in <b>Groovy in Action</b> . Requires <b>Groovy 1.0</b> or better and <b>Java 1.4</b> or better.

### Copy of COM Scripting

#### *Introduction*

**Scriptom** ([scriptom-1.5.zip](#)), an optional Groovy module originally developed by [Guillaume Laforge](#), combines the elegant "syntactical sugar" of Groovy with the power of the [Jacob library \(Java COM Bridge\)](#). The result is something that looks an awful lot like **Windows Scripting Host** (WSH) scripting with VBScript - only it's for **Java**. Once installed into Groovy, **Scriptom** allows you to script **ActiveX** or **COM** Windows components from Groovy.

What this means is that you can use Groovy to automate **Word** or **Excel** documents, control **Internet Explorer**, make your PC talk using the **Microsoft Speech API**, monitor processes with **WMI** (Windows Management Instrumentation), or browse the **Windows Registry** using **WShell** (among many, many other things). It is also a convenient way to talk to custom **VB6** or COM-enabled **Microsoft.NET** libraries (compare this to the administrative headaches of getting **Java** and **.NET** to talk using **Web Services** - it's a no brainer). All this without ever leaving the comfortable **Java** universe.

Of course, **Scriptom** can be used only on **Microsoft Windows**.

**Scriptom** is bundled into the [Groovy Windows-Installer](#), and the most current version can be downloaded from this page (see below). The codebase is considered stable and feature-complete. The **Jacob** project - **Scriptom's** foundation - was started in 1999 and is being used in thousands of production applications worldwide. **Scriptom** is only a few years old, but it is already stable and mature because it is built on an existing, best-of-breed platform.

**Scriptom** gives you all the **COM**-scripting power of **Jacob**, only it is a lot easier.

The **Scriptom** team

- [Guillaume Laforge](#)
- Dierk Koenig
- Marc Guillemot
- [Jason Smith](#)

### Installation

**Scriptom 1.5** requires **Groovy 1.5**. Because of dependencies on **Jacob 1.14**, **Scriptom** also requires **Java 1.5** or higher.

**Download** the project archive(s) and extract the files.

-  [scriptom-1.5.zip](#) Build 10 (12/10/2007) - **Scriptom** JAR file, **Scriptom-1.5.0bX-32.dll**, **Scriptom-1.5.0bX-64.dll**, Java source code, ANT build script, examples, tests, and utilities.
-  [scriptom-1.5-javadoc.zip](#) - **JavaDoc** for **Scriptom** and included type-library information.

Install the jar file and DLL file(s) into your project, and optionally install an update from Microsoft:

1. Add the **Scriptom** jar file (**scriptom-1.5.0bX-XX.jar**) into your Java classpath. It contains both **Scriptom** and **Jacob** class files, so you must not include **jacob.jar**.

 **Scriptom** contains a full (current) version of the **Jacob** library. If you are using **Jacob** directly elsewhere in your project, remove all references to it and use the version built in to **Scriptom**.

2. Copy both **Scriptom-1.5.0bX-XX.dll** files to somewhere on your **java.library.path**. (usually somewhere on the system 'PATH').

 The **Scriptom DLL** naming convention allows multiple versions of **Scriptom** to run on the same machine. However, you can only run a single version of **Scriptom JAR** in any given project.

3. To avoid the dreaded **java.lang.UnsatisfiedLinkError**, download and install one of the following updates from Microsoft: [Microsoft Visual C++ 2005 SP1 Redistributable Package \(x86\)](#) or [Microsoft Visual C++ 2005 SP1 Redistributable Package \(x64\)](#). **Scriptom** doesn't support the **IA-64 (Itanium)** architecture at this time, mainly due to lack of interest. If you are wondering about the different processor architectures, check out the [x86-64 wiki](#). It is usually necessary to install these updates on **Windows Server 2003** and **Windows 2000**, and we've found that it may also be necessary for **Windows XP** and even **Vista**.

 A project can only use one version of **Scriptom** at a time. If you have installed **Groovy** using the **Groovy Windows Installer**, you must remove any versions of **Scriptom JAR** files or replace them with the latest version. A version of **Scriptom** is installed as part of the **Groovy Windows Installer** installation.

**Scriptom 1.5** is not supported for **Groovy 1.0** and earlier (**Scriptom 1.2** is still available). Please contact us if

this presents a serious problem for you.

### ***Building from Source***

The project archive contains all the **Scriptom** source files, Java source files from **Jacob**, and an ANT build script. To save space, the Groovy JAR files have been omitted from the archive. For a successful compilation, Groovy JAR files must be added to the **project/lib** folder.

The latest build requires **Groovy 1.5**.

**Scriptom** includes source files from **Jacob** version **1.14**. You don't need to download it from the **Jacob** project on **SourceForge.net**.

To build the project, run **project/build/make.bat**.

The build process requires **Java 1.5** or higher (**Java 1.6** recommended) and **ANT 1.6.5** or better.

### ***Changes from Scriptom 1.x***

**Scriptom 1.5** is a substantial upgrade to previous versions of **Scriptom**, and *is not backward compatible*. We hope you will agree that it is worth a little code rework to get all these great new features!

- Provides simplified helper methods for dealing with COM apartment threading models.
- **ActiveXProxy** name changed to **ActiveXObject** to match WSH convention.
- **VariantProxy** no longer exists (the functionality was refactored into **ActiveXObject**).
- Variant values are converted to and from the equivalent Java type; no more calling **.value** on everything!
- Supports COM event callbacks using closures (this is still a work in progress, but usable as is).
- Supports indexed properties, including multiple indexes.
- Supports typed, multi-dimensional SafeArrays.
- Supports pass-byref (in/out) values, both in method calls and event handler callbacks.
- Supports COM enumerable types (so you can use **.each**, for example).
- Supports missing arguments (for optional parameters) in method/property calls and events.
- Includes constant definitions for several popular Windows applications, derived from these type libraries (see JavaDoc):
  - Microsoft Office 2003 (works with other versions as well)
    - Word 2003
    - Excel 2003
    - Outlook 2003
    - Access 2003
    - PowerPoint 2003
  - Microsoft Internet Explorer 6 (works with other versions as well)
  - Microsoft Scripting (i.e., FileSystemObject)
  - Microsoft Speech API
  - Windows Scripting Host (WSH)
  - WbemScripting (WMI - Windows Management Instrumentation)
- Supports the latest features from **Jacob 1.14**, including new support for the **Decimal** data type and 64-bit integers.

### ***Migrating from Previous Versions of Scriptom***

**Scriptom 1.5** is not backward compatible with previous versions of **Scriptom**. To get your scripts running again, do this:

- Change all references to **ActiveXProxy** into **ActiveXObject**.
- Wrap any code that references an **ActiveXObject** in **Scriptom.withMTA { ... }**, which replaces the way

Scriptom previously handled COM threading.

- Remove all references to **.value** property references.
- Remove any statically typed references to **VariantProxy**.
- Support for COM events is greatly improved in this version - and also greatly changed. Refer to the **Events** section below for more information.
- Verify that your project is using the correct version of **jacob.dll**.

### **Quick Start**

Let's say you want an application that talks. Pure Java implementations aside (this is, after all, a Groovy/COM tutorial), and ignoring the fact that the default voice on pre-Vista machines sounds like [Dr. Zoidberg](#) with a sinus infection, you could use the **Microsoft Speech API (SAPI)** to get the job done.

You start by creating an **ActiveXObject** with the prog-id for **SpVoice**. Now you can call any of the methods it supports. By default, **SpVoice** will block until it is done speaking, but we're going to have it speak asynchronously and wait until it is done.

```

import org.codehaus.groovy.scriptom.*
import
org.codehaus.groovy.scriptom.tlb.sapi.Speech
VoiceSpeakFlags;
import
org.codehaus.groovy.scriptom.tlb.sapi.Speech
RunState;

//Definitive proof that you CAN talk and
chew gum at the same time.
Scriptom.inApartment
{
    def voice = new
ActiveXObject('SAPI.SpVoice')
    voice.Speak 'GROOVY and SCRIPT um make com
automation simple, fun, and groovy, man!',
SpeechVoiceSpeakFlags.SVSFlagsAsync
    while(voice.Status.RunningState !=
SpeechRunState.SRSEDone)
    {
        println 'Chew gum...'
        sleep 1000
    }
}
println 'Speaker is done.'

```

If you have scripted COM before, you are probably used to using "magic numbers" throughout your code in place of COM constants. In this code sample, we're using fully-qualified constants instead.

**Scriptom** includes fully JavaDoc'd constant and interface definitions from a number of commonly used type-libraries, and you can even create your own. The source code for generating COM type library definitions for **Groovy** is written in **Groovy** 😊, and it's included in the project. It may not seem like a big deal to replace a couple of

numbers, but it will be a lot easier in 10 years to find relevant information on **SpeechVoiceSpeakFlags.SVSFlagsASync** than on the number **1** (Google returns a little more than 9 billion hits for the number '1', and about 1,000 for 'SpeechVoiceSpeakFlags.SVSFlagsAsync,' including hits on this paragraph). And besides, the code reads better.

Speaking of interfaces, it turns out that **SpVoice** supports several. You can test an **ActiveXObject** to see if it supports a given interface using **.supportsInterface**, and you can cast an **ActiveXObject** to a given interface using **.toInterface**.

This next example displays the COM interfaces that **SpVoice** supports (within the **SAPI** library only):

```
import org.codehaus.groovy.scriptom.*;
import
org.codehaus.groovy.scriptom.tlb.sapi.Speech
Lib;

Scriptom.inApartment
{
    def voice = new
ActiveXObject( 'SAPI.SpVoice' )
    SpeechLib.interfaces.each {name, iid ->
if(voice.supportsInterface(iid)) println
"SpeechLib.$name - $iid"}
}
```

#### *Programmer's Guide*

<a href="#">The Least You Need to Know about COM</a>	how not to crash the JVM or lock your process
<a href="#">COM Data Types in Scriptom</a>	supported types and conversions in Java, VB6, and VB.NET
<a href="#">COM Methods and Properties in Scriptom</a>	calling COM methods and properties from Groovy
<a href="#">Passing Values by Reference (in-out)</a>	how to let methods change the value of passed-in arguments
<a href="#">COM Events</a>	support for COM events
<a href="#">All About Arrays</a>	UNDER CONSTRUCTION

#### *Examples*

Here is a simple example that uses the Microsoft **ScriptControl** to evaluate a JScript expression. This is a very indirect way to add 2 and 2.

```
import org.codehaus.groovy.scriptom.*

Scriptom.inApartment
{
    def scriptControl = new
    ActiveXObject("ScriptControl")
    scriptControl.Language = "JScript"
    println scriptControl.Eval('2.0 + 2.0;')
}
```

There are many, many potential uses for **Scriptom** - far to many to try to maintain as part of this documentation. So we've included a whole slew of meaty examples in the project archive for you to play with. We'd like to encourage you to go look at those examples, run them, and modify them. And if you come up with an especially interesting example, let us know about it. We may even include it in a future release!

Some additional examples included with **Scriptom**:

- Automated writing to and reading from **Excel** spreadsheets. Includes COM events example.
- Navigation in **Internet Explorer**. Includes COM events example.
- Gathering information about processes and Windows NT services using **WMI**.
- Parsing a \*.msg file with **Microsoft Outlook**.

Consuming **Visual Basic 6** (VB6) and **Visual Basic.NET** COM-enabled DLLs.

#### Articles

## Articles about COM scripting in general, and Scriptom in particular.

<a href="#">Getting Started with COM</a>	January 11, 2008	Tips for exploring <b>COM</b> libraries, finding documentation, etc.
<a href="#">Bridging the Gap Between Java and .NET with Groovy and Scriptom</a>	December 3, 2007	Implement <b>FIPS 140-1</b> compliant <b>S HA-1</b> in <b>Java</b> with <b>Groovy</b> , <b>Scriptom</b> , and <b>Visual Basic.NET</b> .
<a href="#">Juixe TechKnow</a>	August 8, 2006	A short <b>Microsoft Outlook</b> example.

<a href="#">Siebel</a>	March 17th, 2008	Siebel "business component" programming
------------------------	------------------	---

### **Post Scriptom**

All known (unresolved) issues and feature requests are listed in the [Scriptom Jira](#) database.

Changes to each build are summarized in the [Change Log](#).

Recent builds of **Scriptom** can be found [here](#). Older versions are archived:

<a href="#">Scriptom-1.5.zip</a>	Build 10. Bundled with the <b>Groovy 1.5 Windows Installer</b> . Requires <b>Groovy 1.5</b> and <b>Java 5</b> or better.
<a href="#">Scriptom 1.2</a>	(also <b>Scriptom 1.1</b> and <b>Scriptom 1.0</b> ) Original version, and the one that is documented in <b>Groovy in Action</b> . Requires <b>Groovy 1.0</b> or better and <b>Java 1.4</b> or better.

### **Scriptom 1.2**

 This is the page of the old version of Scriptom. As the new version, [Scriptom 2.0](#), brings with it a couple breaking changes, you may still want to use this version. However, we highly encourage you to switch to the [newer version](#) which solves a certain number of bugs and provide a much better integration to COM / ActiveX components.

### **Introduction**

**Scriptom** is an optional Groovy module developed by [Guillaume Laforge](#) leveraging the [Jacob library \(JAvA COM Bridge\)](#). Once installed in your Groovy installation, it allows you to script any ActiveX or COM Windows component from within your Groovy script. Of course, this module can be used on Windows only.

**Scriptom** is especially interesting if you are developing Groovy shell scripts under Windows. You can combine both Groovy code and any Java library with the platform-specific features available to Windows Scripting Host or OLE COM automation from Office.

### **Installation**

#### **Zip bundle**

The easiest way for installing Scriptom is to unzip the [Zip bundle](#) in your `%GROOVY_HOME%` directory. The distribution contains the `jacob.jar` and `jacob.dll`, and the `scriptom.jar`. The DLL needs to be in the bin directory, or in your `java.library.path` to be loaded by `jacob.jar`.

#### **Building from sources**

If you are brave enough and prefer using the very latest fresh version from CVS Head, you can build **Scriptom** from sources. Checkout `modules/scriptom`, and use Maven to do the installation automatically. If your `%GROOVY_HOME%` points at the `target/install` directory of your groovy-core source tree, just type:

```
maven
```

Otherwise, if you have installed Groovy in a different directory, you have two possibilities, either you change the property `groovy.install.staging.dest` to your `%GROOVY_HOME%` directory in the `project.properties` file, and run maven, or you can type:

```
maven  
-Dgroovy.install.staging.dest=%GROOVY_HOME%
```

#### *Usage*

Let's say we want to script Internet Explorer. First, we're going to import the ActiveX proxy class. Then, we're going to create a GroovyObjectSupport wrapper around the ActiveXComponent class of Jacob. And now, we're ready to use properties or methods from the component:

```
import  
org.codehaus.groovy.scriptom.ActiveXProxy  
  
// instantiate Internet Explorer  
def explorer = new  
ActiveXProxy("InternetExplorer.Application")  
  
// set its properties  
explorer.Visible = true  
explorer.AddressBar = true  
  
// navigate to a site by calling the  
Navigate() method  
explorer.Navigate("http://glaforge.free.fr/w  
eblog")
```

Note however that `explorer.Visible` returns a proxy, if you want to get the real value of that property, you will have to

use the expression `explorer.Visible.value` or `explorer.Visible.getValue()`.

#### Limitations

For the moment, **Scriptom** is in a beta stage, so you may encounter some bugs or limitations with certain ActiveX or COM component, so don't hesitate to post bugs either in JIRA or on the mailing lists. There may be some issues with the mappings of certain objects returned by the component and the Java/Groovy counterpart.

An important limitation for the first release is that it is not yet possible to subscribe to events generated by the components you are scripting. In the next releases, I hope I will be able to let you define your own event handlers with closures, with something like:

```
import
org.codehaus.groovy.scriptom.ActiveXProxy

def explorer = new
ActiveXProxy("InternetExplorer.Application")
explorer.events.OnQuit = { println "Quit" }
explorer.events.listen()
```

But for the moment, event callbacks are not supported.

There is an experimental implementation currently in CVS Head, it does not work with the groovy command, but it does work when launching a script from a Java program with the GroovyShell object. There is perhaps a problem with Classworlds or Jacob, and the different classloaders. If anyone has a clue, I'm game!

#### *Samples*

If you checkout the **Scriptom** sources, you will find a few samples in the **src/script** directory. I will show you some samples in the following sub-sections.

#### Scripting Internet Explorer

```
import
org.codehaus.groovy.scriptom.ActiveXProxy

// instantiate Internet Explorer
def explorer = new
ActiveXProxy("InternetExplorer.Application")

// set its properties
explorer.Visible = true
explorer.AddressBar = true

// navigate to a site
explorer.Navigate("http://glaforge.free.fr/w
eblog")
Thread.sleep(1000)
explorer.StatusText = "Guillaume Laforge's
weblog"
Thread.sleep(2000)

// quit Internet Explorer
explorer.Quit()
```

Scripting Excel

```
import
org.codehaus.groovy.scriptom.ActiveXProxy

// create a proxy for Excel
def xls = new
ActiveXProxy("Excel.Application")
```

```
xls.Visible = true

Thread.sleep(1000)

// get the workbooks object
def workbooks = xls.Workbooks
// add a new workbook
def workbook = workbooks.Add()

// select the active sheet
def sheet = workbook.ActiveSheet

// get a handle on two cells
a1 = sheet.Range('A1')
a2 = sheet.Range('A2')

// sets a value for A1
a1.Value = 123.456
// defines a formula in A2
a2.Formula = '=A1*2'

println "a1: ${a1.Value.value}"
println "a2: ${a2.Value.getValue()}"

// close the workbook without asking for
saving the file
```

```
workbook.Close(false, null, false)
// quits excel
xls.Quit()
```

Warning: on my machine (WinXP Home), there is still an Excel.exe process running. I have no clue why Excel is still running.

Mixing VBScript or JScript with Groovy

```
import
org.codehaus.groovy.scriptom.ActiveXProxy

// invoke some VBScript from Groovy and get
the results!
def sc = new ActiveXProxy("ScriptControl")
sc.Language = "VBScript"
println sc.Eval("1 + 1").value
```

Scripting the Windows Shell object

```
import
org.codehaus.groovy.scriptom.ActiveXProxy

// showing the current directory
def cmd = new
ActiveXProxy("Scripting.FileSystemObject")
println cmd.GetAbsolutePathName(".").value

sh = new ActiveXProxy("Shell.Application")

// minimizing all opened windows
sh.MinimizeAll()
```

```
// opens an Explorer at the current location
sh.Explore(cmd.GetAbsolutePathName(".").value)
)
```

```
// choosing a folder from a native windows
directory chooser
def folder = sh.BrowseForFolder(0, "Choose a
folder", 0)
println folder.Items().Item().Path.value
```

```
def wshell = new
ActiveXProxy("WScript.Shell")
// create a popup
wshell.popup("Groovy popup")
```

```
// show some key from the registry
def key =
"HKEY_CURRENT_USER\\Software\\Microsoft\\Win
dows\\CurrentVersion\\Internet
Settings\\User Agent"
println wshell.RegRead(key).value
```

```
def net = new
```

```
ActiveXProxy("WScript.Network")
// prints the computer name
println net.ComputerName.value
```

Scripting Windows Media Player

```
import
org.codehaus.groovy.scriptom.ActiveXProxy
import java.io.File

// create a proxy for the Shell object
def sh = new
ActiveXProxy("Shell.Application")

// use a Windows standard folder chooser
def folder = sh.BrowseForFolder(0, "Choose a
folder with wav files", 0)

// get the folder chosen
def folderName =
folder.Items().Item().Path.value
println "Playing Wav files from:
${folderName}"

// create a Windows Media Player (from its
Class ID)
def player = new
ActiveXProxy("clsid:{6BF52A52-394A-11D3-B153
-00C04F79FAA6}")

// for each file in the folder
```

```
new File(folderName).eachFile{ file ->
    if (file.name.endsWith("wav")) {
        println file
        player.URL = file.absolutePath
        // play the wav for one second
        control = player.controls.play()
        Thread.sleep(1000)
    }
}
```

```
// close the player  
player.close()
```

When event callbacks are supported, you will be able to subscribe to the `player.statusChange` event, so that you can play the wav entirely, before loading a new sample (instead of listening only to the first second of each sample).

#### Converting a Word document into HTML

This program takes a Word document as first parameter, and generate an HTML file with the same name, but with the `.html` extension.

```
import  
org.codehaus.groovy.scriptom.ActiveXProxy  
import java.io.File  
  
def word = new  
ActiveXProxy("Word.Application")  
  
word.Documents.Open(new  
File(args[0]).canonicalPath)  
word.ActiveDocument.SaveAs(new File(args[0]  
- ".doc" + ".html").canonicalPath, 8)  
word.Quit()
```

#### Printing the contents of your Outlook Inbox

```

import
org.codehaus.groovy.scriptom.ActiveXProxy

def outlook = new
ActiveXProxy("Outlook.Application")
def namespace = outlook.GetNamespace("MAPI")
// There is only "MAPI"

// 6 == Inbox; other values in Outlook's VBA
documentation
def inbox = namespace.GetDefaultFolder(6)
def mails = inbox.Items

println "Elements in your Inbox: " +
mails.Count.value

for (i in 1..mails.Count.value) {
  def mail = mails.Item(i)
  println i + ": " + mail.Subject.value + "
(" + mail.Size.value + " bytes)"
}

```

## Scriptom Articles

Articles about **COM scripting** in general, and **Scriptom** in particular.

<a href="#">Getting Started with COM</a>	January 11, 2008	Tips for exploring <b>COM</b> libraries, finding documentation, etc.
<a href="#">Bridging the Gap Between Java and .NET with Groovy and Scriptom</a>	December 3, 2007	Implement <b>FIPS 140-1</b> compliant <b>S HA-1</b> in <b>Java</b> with <b>Groovy</b> , <b>Scriptom</b> , and <b>Visual Basic.NET</b> .

<a href="#">Juixe TechKnow</a>	August 8, 2006	A short <b>Microsoft Outlook</b> example.
<a href="#">Siebel</a>	March 17th, 2008	Siebel "business component" programming

## Bridging the Gap Between Java and .NET with Groovy and Scriptom

Ever wonder what all the fighting is about?

On the one hand, we have **Java**, with its basis in object-oriented theory, automatic memory management, dependency on a runtime, and a robust library. And on the other, we have **Microsoft.NET**, with its basis in object-oriented theory, automatic memory management, dependency on a runtime, and a robust library.

You'd think with all these two kids have in common, they would get along better. No such luck.

Up until now, getting **Java** to talk to **Microsoft.NET** has been only slightly more fun than being on the receiving end of a root canal. One book I picked up recently was all about interoperability using **Web Services**. *Web Services!* Not that there is anything wrong with **Web Services** in general, but I why do I need a web server, **WSDL**, and the overhead of two-way **XML** serialization if I just want a list of the processes on my local system (something **Java** can not do)?

Maybe you've tried something like passing data to a spawned process using standard-input and standard-output. It's limited, but it works. You still need something like **SOAP** or **XStream** if you want to work with objects, and at least you don't have the overhead of a full blown web server. But it's still not simple. Or maybe you've tried using sockets. Same thing, but without incurring the overhead of starting and stopping a process every time you need to do something that is windows-ey. Still, that's not simple.

You are probably thinking, "There must be a better way." And that would make sense, because you took the time to read the title of this article, correct? I am here to tell you - *there is a better way*. In fact, I am here to tell you, it's *easy*.

### Microsoft's Component Object Model (COM)

Microsoft spent decades developing this technology called **COM**. It stands for **Component Object Model**. "Wait a second," you say. "I think I remember reading about that somewhere. Isn't **COM** dead? Doesn't **.NET** make it obsolete?" You might think so, but look how long **.NET** has been around, and **Microsoft Office** is still compiled from plain old **ANSI C++**. Consider also that the **.NET** libraries for **Microsoft Office** are thin wrappers around the **COM** libraries that **Office** exposes. **COM** is embedded deeply in **Windows**, it is mature, stable, and it's not going anywhere.

**Microsoft.NET** has excellent support for **COM** interoperability. You can create and consume managed **COM** libraries using **VB.NET**, **C#**, or **C++/CLI**. It's almost trivial to do so in **VB.NET**, for example. You just set a few properties, build the project, and you've got a **COM-callable DLL**.

So if **COM** is there, it's mature, it's stable over decades, and it works for what you are doing, *why not just use it?*

### Groovy and Scriptom

**Groovy**, for those who aren't already familiar with it, is a very cool dynamic language that is designed specifically to run on the **Java JVM**. The syntax is based on **Java**, and everything gets compiled to **Java** bytecode. **Java** program

mers have an easy learning curve, and **Groovy** can be integrated seamlessly (and quickly) into existing **Java** projects. **Groovy** is a lot more expressive than **Java**, so you can typically get things done in about half the lines of code. And once you get used to it, **Groovy** is easier to read than **Java**.

**Scriptom** is an optional module for **Groovy** that lets you use **COM** objects without having to know anything (or at least very little) about the inner workings of **COM**. **Scriptom** takes advantage of **Groovy's** dynamic (late-bound) nature to map **COM** objects into **Groovy** objects at run-time. There are no type-libraries to deal with and no wrappers to maintain. Just code it and run it. It really is a lot like writing code using **VBScript** (except that **Groovy** is far more advanced as a language). It is almost eerie at times to see **Java** comingled with what appears to be something straight out of **WSH (Windows Script Host)**. Thus, the name - "*Script-om*." It's very script-like.

### *Let's Do Something! FIPS 140-1 on Java*

It's high time for an example. It is said that the best way to learn is by doing, so let's put together a small application. And what's more, let's make it practical.

I was working on a project recently that needed a [FIPS 140-1](#) compliant cryptographic hash/digest algorithm. I needed to protect user passwords, and my client has some pretty strict rules about which cryptographic algorithms may and may not be used. **Java** implements several hash/digest algorithms, but none of them are compliant with **FIPS 140-1**. What this means is that I had to look outside of **Java** for a solution.

After Googling for several hours, I realized that it is pretty hard to find anything that is certified as **FIPS 140-1** compliant. There are some big libraries, mostly written in **C++** with **Java JNI** wrappers. One or two are open source, but most cost money (and they don't list a price - you *know* what that means). All of them were far too complex for my needs. I don't want to have to maintain yet-another-set-of-libraries for just one measly function point.

So, cutting to the chase, it turns out that a number of the algorithms available in the **MS-CAPI (Microsoft Crypto API)** are certified **FIPS 140-1** compliant! That's right, what I needed was built right in to the base installation of Windows. All I needed to do was figure out how to get from the Windows SDK API level into **Java**. After a little more research, I discovered that Microsoft has wrapped parts of the **MS-CAPI** into the **.NET Framework**. One of these wraps the **FIPS 140-1** compliant implementation of [SHA-1](#). Perfect!

So all I had to do was:

- create a **.NET DLL** with **COM**-accessible objects, and
- access it using **Groovy** and **Scriptom**.

Piece of cake!

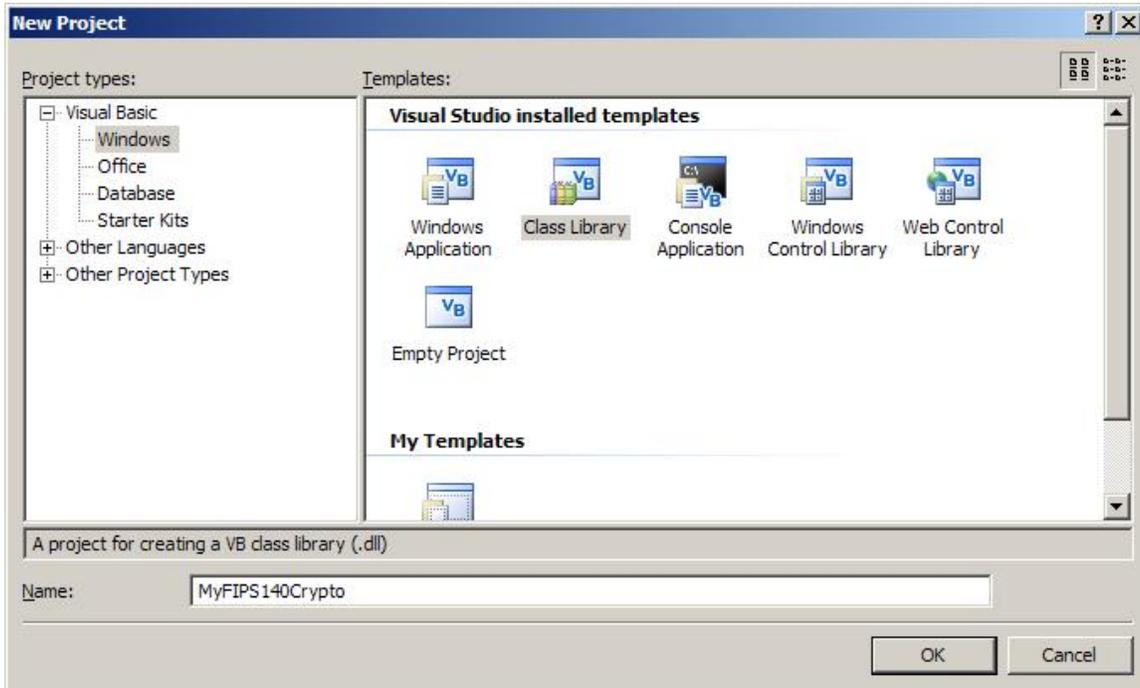


What I am showing you here isn't sufficient for securely hashing a password. You'd need to add a unique seed value, such as the user's name or ID number, to prevent simple birthday attacks. There are some other things you would need to do to strengthen the password to prevent easy brute-force attacks. Consult your local security guru before using any cryptography. If you don't understand what you are doing, you are probably doing it wrong (and fooling yourself about how secure you are in the process). Poorly implemented cryptography is often as bad, or worse, than none at all.

First things first. Let's create the **DLL**. We'll be wrapping Microsoft's **SHA1CryptoServiceProvider** (pretty trivial to code), and we'll also be walking through all the steps to make the **DLL COM-accessible**. I'm using **Visual Basic** in this example, but with a little translation this should work with any of the mainstream **.NET** languages.

**Step 1. Create a New Project**

Open **Visual Studio** and create a new **VB** class library project called **MyFips140Crypto**.



**Step 2. Add the Code**

Rename **Class1** to **SHA** and add the following code:

```
Option Explicit On  
Option Compare Text  
Option Strict On
```

```
Imports System.Security.Cryptography  
Imports System.Text
```

```
<Microsoft.VisualBasic.ComClass()> Public  
Class SHA
```

```
<System.Runtime.InteropServices.ComVisible(F
```

```

else)> _
    Private Provider As
SHA1CryptoServiceProvider

    ' A creatable COM class must have a
Public Sub New()
    ' with no parameters, otherwise, the
class will not be
    ' registered in the COM registry and
cannot be created
    ' via CreateObject.
Public Sub New()
    MyBase.New()

    Provider = New
SHA1CryptoServiceProvider()
    Provider.Initialize()
End Sub

''' <summary>
'''     Generates the SHA hash of an input
string.
''' </summary>
''' <param name="Input">
'''     A string of characters;
'''     case is significant, and unicode
is accepted.
''' </param>
''' <returns>
'''     A 20-byte hash representing the
160 bit result.

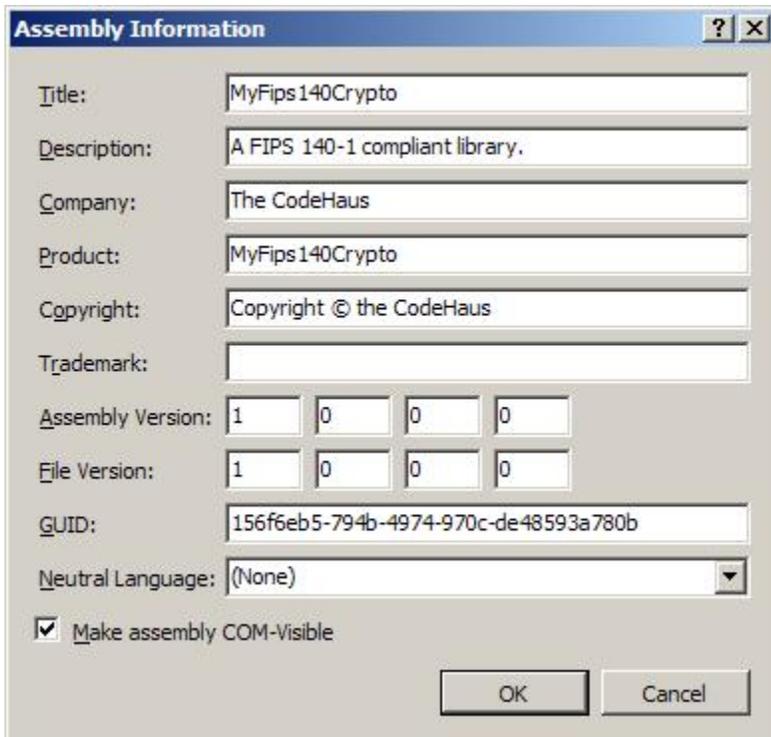
```

```
''' </returns>
''' <remarks>
'''     This uses the MS CAPI algorithm
for SHA-1, so it
'''     is FIPS 140-1 compliant.
''' </remarks>
Public Function GenerateHash(ByVal Input
As String) As Byte()
    Return
Provider.ComputeHash(Encoding.Unicode.GetByt
```

```
es (Input) )  
    End Function  
End Class
```

### Step 3. Enable COM

To enable **COM** support, open the project properties and click the **Assembly Info** button. Fill in the dialog any way you want, but make sure that **Make assembly COM-Visible** is checked.



The screenshot shows the 'Assembly Information' dialog box with the following fields and values:

- Title: MyFips140Crypto
- Description: A FIPS 140-1 compliant library.
- Company: The CodeHaus
- Product: MyFips140Crypto
- Copyright: Copyright © the CodeHaus
- Trademark: (empty)
- Assembly Version: 1.0.0.0
- File Version: 1.0.0.0
- GUID: 156f6eb5-794b-4974-970c-de48593a780b
- Neutral Language: (None)
- Make assembly COM-Visible

Buttons: OK, Cancel

### Step 4. Give It a Strong Name

What is a "strong name," you ask? Well, it is some sort of Microsoft security cryptographic digital signature thingy. Why do you care? In most cases, you won't. The important thing to remember is that you won't be able to successfully register the library later without this.

Lucky for us, Microsoft has made this step relatively easy. From the project properties page, you can have **Visual Studio** create a new strong name key file for you. You can optionally protect the key file with a password.



Once the strong name key file is associated with your project, you generally don't have to worry about it again.

#### Step 5. Build It

Build the project.

#### Step 6. Register It

Open **Visual Studio 2005 Command Prompt**. It is one of the tools that comes standard with **Visual Studio**. Navigate to the folder where your compiled DLL is going to reside, and type in the following command:

```
C:\work\MyFips140Crypto\MyFips140Crypto\bin\
Release>regasm MyFips140Crypto.dll /codebase
```

❌ The **/codebase** parameter must be specified. If you forget, the assembly will appear to be registered correctly, but you will get a "Could not co-create object" exception when you try to run this from **Groovy**.

If you have been lucky enough to get everything right so far, you'll get a confirmation message telling you that the assembly was registered successfully. Congratulations!

⚠️ If you are using **Visual Studio Express**, you may not have a **Visual Studio 2005 Command Prompt**. The **regasm.exe** utility is available on your machine, and you can access it by searching for the file and adjusting your path environment variable, or by installing the full **Microsoft.NET SDK**.

❌ If you add new **COM**-visible classes to your library, you will have to re-register the **DLL** in order for **Groovy** to be able to see them.

- ➖ The **regasm.exe** utility requires **Administrator** privileges to execute. This can affect installations, and is particularly relevant on **Windows Vista**, since the command prompt must be started with the correct privileges.

### *Groovy!*

Okay, that was tough (not really). There are a couple of not-so-obvious steps involved in creating even a simple **COM-enabled .NET** library. Fortunately, once you know all the steps, you just follow the same recipe. Once you have set up a **.NET** project this way, it is easy to add additional methods and classes.

You might expect that for the **Groovy** portion of this project, there is a similar type of overhead incurred. You'd be wrong though. With **Groovy**, it is more like scripting. I wasn't lying earlier when I said this was easy. Here is all the code you need to compute a **FIPS 140-1** compliant **SHA-1** hash of an arbitrary string:

```
import org.codehaus.groovy.scriptom.*;

Scriptom.inApartment
{
    ActiveXObject sha = new
ActiveXObject('MyFips140Crypto.SHA')
    SafeArray hash = sha.GenerateHash('Your
plain text goes here.')
    println hash.toArray() //Converts COM
SafeArray to Java byte[]
}
```

There you have it. A **FIPS 140-1** approved **SHA-1** implementation in about 20 lines of code. That was easy.

**i** **COM** objects always run in an "apartment." Different "apartments" correspond to different threading models, and the **COM** threading models don't quite correspond to **Java's** threading model. To help you with initialization and teardown of **COM** apartments within **Java**, **Scriptom** provides the **.inApartment** method. It is considered best practice to wrap any **COM** objects in an **.inApartment** closure. Threading in **Scriptom** can also be managed by calling the methods on the **com.jacob.com.ComThread** object.

### *Integrating with Java*

This part is for you if you are new to **Groovy**. If you have been using **Groovy** for a while, you're done. Go forth and codify already.

At first glance, it might not be obvious how easy it is to integrate this into your existing **Java** project. It's easy. It's really, really easy. **Groovy** objects are compiled to **Java** byte-code. In other words, **Groovy** objects are **Java** objects. So for the most part, **Java** can call methods on **Groovy** objects just like calling methods on another **Java** object. That's what they mean when they say: "**Groovy is Java!**"

So let's wrap our script into a **Groovy** object so that it can be easily called from **Java**. Create a file called **Fips140SHA1.groovy** in **/org/yourcompany/yourproject**:

```

package org.yourcompany.yourproject;

import org.codehaus.groovy.scriptom.*;

public class Fips140Sha1
{
    public byte[] generateHash(String input)
    {
        return Scriptom.inApartment
        {
            ActiveXObject sha = new
ActiveXObject('MyFips140Crypto.SHA');
            SafeArray hash =
sha.GenerateHash(input);
            return hash.toArray(); //Converts COM
SafeArray to Java byte[]
        }
    }
}

```

Look familiar? It's kind of hard to tell the difference between **Groovy** syntax and **Java** syntax sometimes, isn't it? Compile your project with **Groovyc** (either the command-line executable or the ANT task) and you are done.

#### *Wrap Up*

We've only just scratched the surface of what it is possible to do with **Scriptom** and **.NET**, not to mention all the things you can do with the hundreds of existing **COM** libraries that are available in **Windows**. No one is suggesting that you give up **WORA**, but for those *MeSsy* jobs that **Java** just can't do, this is one approach that is quick, painless, and pretty easy to maintain over time. If you like what you see, or if you want to provide some constructive criticism, drop by the **Groovy users** mailing list and let us know what you think.

[Jason Smith](#) is currently a member of the **Scriptom** development team.

**Getting Started with COM**

Code written with **Scriptom** looks great and is easy to read. Sure. But it may not be as easy to write as it is to read. Where is the look-ahead help? Where is the integration with your favorite IDE? How do you get started writing code against **COM** objects, many of which have little or no online documentation? In this rather short article, I talk about why **Scriptom** does not have IDE integration, along with some techniques and tools you can use to make **COM** programming with **Scriptom** much easier.

### ***No IDE Integration for Scriptom***

Let's start with IDE integration. To make it short, there isn't any, and it isn't planned. The trouble is, **Scriptom** uses "late bound" **COM** calls, exclusively. Every **COM** object in **Scriptom** is just an **ActiveXObject** to **Groovy**. Even at runtime, there is no reliable way to determine exactly what kind of an object you are working with under the covers. **Scriptom** just passes the method name, along with the arguments, to **Windows** - and trusts that everything will go well.

For those of us who are used to static typing, this may seem like a dangerous practice. I think you will find, though, that it works surprisingly well. It also simplifies things considerably. Have you seen the number of **COM** wrappers that **Microsoft.NET** has to support (for some, multiple versions of the same library)? **Scriptom** doesn't require a **COM** wrapper generator (though it does include constant definitions for some **COM** libraries). You can just write your code and run it. And it just works.

**Scriptom** uses the same calling mechanism that is used by Microsoft's typeless scripting languages, like **VBScript** and **JScript**. It is also the calling convention that is used from **Visual Basic 6** or **VBA** when you are working with an **Object**. That is why they call it **Scriptom**. It uses the calling convention that is normally used by scripting languages.

### ***An Easier Way than Trial and Error***

So if there is no IDE integration for **Scriptom**, how do you figure out the **COM** thing? Here are some ideas, based on my own experience working with **COM**. These aren't quite the same as having full IDE integration, like **Java** in **Eclipse**, but in my experience they do help you get the job done.

- *Use **VB6** or **VBA** to browse the object models and write sample code.* **Visual Basic 6** and **VBA** have a nearly identical set of tools for browsing objects, setting references to **COM** libraries, and for look-ahead help. Most **COM** libraries were designed to be compatible with **Visual Basic**, and those will usually work well with **Scriptom**. **VBA** is available as the macro language in all the **Microsoft Office** products, and in many other products as well.
  - *Alternately, use **Visual Studio.NET** to browse the **COM** object model.* I don't recommend this over **VB6/VBA** because the support for **COM** is a little more obscure in the **Microsoft.NET** products. However, in a pinch, it works, and the **Visual Studio Express** products are free.
  - **Visual Basic 6** is part of **Visual Studio 6**, which was desupported by Microsoft in 2008. **Visual Studio 6** still works if you have it, but if you don't, you can't get it. Microsoft has removed it and all the associated documentation from their website.
- *Buy a book.* There are many books available that cover **COM APIs** for specific products. For example, a used copy of **Excel 2002 VBA** on Amazon would run you less the \$15, including shipping. When you are working with the really complex applications, like **Excel** and **Word**, I have always found it beneficial to have a second opinion.
- *Use the web to find examples.* Some people call it "Programming by Google." If you want to do something, you can bet that some other bloke has already done it and published it somewhere on the web.
- *Read the manual.* Use Microsoft's search tool at <http://msdn.microsoft.com> to find the official documentation. Some of it is very good, some it will leave you with questions, but all of it is official.
- *Don't write code, write a macro.* For **Microsoft Office** applications that support macros, the macro language is **VBA**. This means that when you create a macro, **Word** or **Excel** will actually write the code for you. It is usually trivial to port **VBA** to **Groovy/Scriptom**. This is an excellent way to figure out how to do obscure

tasks in **Office**.

There. That should get you started off on the right foot. Enjoy!

This article was consolidated from ideas developed in an email thread in the **Groovy users** list. Thanks to Thomas Elmiger and A. Tres Finocchiaro.

## Using Scriptom to Automate Microsoft Excel

### *Why Scriptom?*

There must be a dozen different libraries out there that support reading and writing **Excel** documents. Some, like [POI](#) or [OpenXLS](#), let you manipulate **Excel** files directly from Java. These libraries don't use **Microsoft Excel**, so they work well in mixed OS environments. However, they don't support some key features of **Excel** (there are significant limitations), and their APIs don't correspond to **Excel's COM API**, so it can be difficult at times to figure out how use them to do even simple things.

Other libraries, like [COM4J](#), [JCOM](#), and [JACOB](#) let you use **Excel** automation to read and create **Excel** files, either directly or through wrapper classes generated from **COM** type libraries. This gives you full access to the complete featureset of **Excel**. It also limits you to **Windows**. It can be sometimes be difficult to translate from Microsoft's Visual Basic-oriented documentation to the corresponding library/wrapper in Java. This can be particularly challenging if you aren't already up to speed on **COM** and the **Excel API** - which is probably true for most Java programmers. **Excel** automation is also a lot slower than direct file manipulation (process startup overhead, process-to-process communication).

**Scriptom** is a **COM** automation library for **Groovy**, so it falls squarely into the second group. In fact, **Scriptom** is an extension of [JACOB](#). **JACOB** does the heavy lifting, and **Scriptom** makes it groovy.

*So why would you use **Scriptom** over one of the alternatives?*

Scriptom is intended to replicate the feel of COM scripting - as much as possible - but with Groovy instead of VBScript. It looks like COM scripting, and it feels like COM scripting. So when you are, for instance, translating a VBA macro to its equivalent Groovy, there is very little translation involved. You can't quite do a copy-paste, but it's close. And when you are programming against the Microsoft documentation for the Excel API, you don't have to ask a lot of questions. The code simply works the way it's documented.

Additionally, Scriptom provides special support for scripting Excel. The Scriptom library contains definitions for all the constants in the Excel type library. Scriptom also includes **ExcelHelper** ([org.codehaus.groovy.scriptom.util.Excel.ExcelHelper](#) - whew!), which gives you an easy way to work with Excel documents in a multiprocessing (server) environment.

Are you feeling groovy yet?

Scriptom is suitable for use in low-to-medium volume tasks where Excel is available. For example, it is useful in batch jobs, where you aren't too concerned about execution time. And it can be used to produce reports in Excel format for a web site (throw in a pie chart in a few lines of code), as long as you don't try to deliver more than one report every couple of seconds.

If you feel the need for speed and you can live with its limitations, consider POI. In fact, many Java-based frameworks and libraries that provide Excel support out of the box (Cocoon, ColdFusion, etc.) use POI under the covers. However, if your goals are *easy-to-write*, *easy-to-maintain*, and *full-featured*, and the **Windows/Excel** thing isn't a showstopper, your best bet may be **Scriptom**.

## ***Let's Get Started!***

LINK TO DOCUMENT WITH INSTALLATION INSTRUCTIONS

Enough with the introductions. Let's get started with an example that shows just how easy it is to work with Excel using Scriptom.

<http://poi.apache.org/hssf/index.html>

<http://www.rgagnon.com/javadetails/java-0516.html> \\\- various ways to read and write Excel documents

minimalistic framework, and syntax that corresponds directly to the Microsoft documentation

## **The Least You Need to Know about COM**

The [wiki on COM](#) explains, among many other things, that **COM (Component Object Model)** is an older Microsoft technology that encompasses **OLE, OLE Automation, ActiveX, COM+** and **DCOM**. **Microsoft.NET** is not based on **COM**, but for the most part it supports it seamlessly. **Scriptom** lets you easily integrate these technologies into your **Java/Groovy** project.

Despite the fact that **Visual Basic** versions up through 6 were abstractions on top of **COM**, it's not at all simple. I won't go into the details here (the [wiki](#) does a better job than I could at explaining it all). The important point to remember is that **COM** is fundamentally a **C/C++** based technology that was developed using pre-Java ideas of object orientation. To be specific, **C/C++** apps are generally responsible for cleaning up allocated memory, and the **COM** threading models differ somewhat from **Java**.

Because **C++/COM** and **Java** aren't quite compatible, it is possible to get **Scriptom** to randomly crash your JVM (core-dump), use up all available memory, peg the CPU at 100% for hours on end, or prevent your Java process from exiting. If you use it correctly, though, you'll find that **Scriptom** is rock-solid stable.

This article is going to explain, in very simple terms, some of the differences between **COM** and **Java**, and how to deal with the issues they present in the simplest way possible.

## **Apartment Threads for Absolute Beginners**

Sorry about this, but you have to manage **COM** thread apartment contexts. The good news is that this is not nearly as onerous as it sounds. You can do this using **Jacob** directly (giving you a lot of flexibility). Of course, this article is about the simple, safe ways to use **Scriptom**, so we'll leave the advanced **Jacob** topics for another.

**Scriptom** provides an even simpler and safer way that handles 95% of the cases you are likely to run into. It also tends to produce the most stable applications, making it ideal for use on the server side. All you have to do is wrap your code in a **Groovy** closure, like this:

## **Scriptom.inApartment**

```
{  
    ...  
}
```

When you wrap your Scriptom code using **.inApartment**, the thread apartment is initialized if needed, and all **COM** resources are released when you are done. You can do this over and over again in a thread, and you can also nest calls to **.inApartment**, allowing you to use it safely with other code (this addresses an issue with **Jacob**). You'll see this pattern time and time again in our examples and test code.

The main side-effect of this is that you have to define all your **ActiveXObjects** within the apartment scope (the closure). In case that doesn't work for your purposes, you can freely use **Jacob** to manage your **COM** thread contexts. However, you can't safely mix and match the **Scriptom** thread management with **Jacob** calls in the same thread.

### **It's NOT a Memory Leak!**

It should come as no surprise that **COM** strings, numbers, and objects are not interchangeable with **Java** strings, numbers, and objects (though **Scriptom** does a pretty good job of making them look as if they are). Every time we exchange data with a **COM** library, the **Java** data-type needs to be converted to and from a form that is suitable for **COM**. That form just happens to be Microsoft's (pseudo) universal data type: the **Variant**. Without going too deeply into the murky details, a **Variant** can store various forms of numbers, strings, and objects. You don't normally need to worry about this because **Scriptom** handles all the conversion details for you.

What you need to know about **Variants** is that they can't be completely reclaimed by standard **Java** garbage collection. There is a little "residue" that **Jacob** has to hold on to even after you are finished using the **Variant**. This is necessary to prevent random JVM crashes (it's a **C/C++/JNI** thing). Over a large number of calls, this residue can fill all the memory on your server.

*Don't panic!* It isn't a memory leak. When you exit the **.inApartment** closure, the **COM** context is released and all the memory is reclaimed.

There is one more thing you should be aware of. The clean-up process gets much slower if you make a very large number (millions) of **COM** calls before cleaning up. So for a long running thread, you are better off breaking up the apartment contexts rather than having one context for the whole thread.

In short:

- Use an **.inApartment** closure around your **COM** code (more specifically, any code that uses **ActiveXObject**).
- Break **.inApartment** closures into reasonably-sized chunks.

## **Gant**

Gant has moved: it has become a project in its own right and has its own space -- <http://gant.codehaus.org>.

## **GFreeMarker**

## **Contribution Overview**

GFreeMarker is an integration of the FreeMarker template engine for Groovy. It allows adding plugins written in Groovy for content rendering, making FreeMarker+Groovy just as easy as [Smarty](#). This way of doing avoids security issues related to allowing direct Groovy scripting in templates. With GFreeMarker, **you** create plugins, and make it available to customers.

### ***Team Members***

Cedric Champeau <cedric dot champeau at lingway dot com>

### **Download**

#### ***Distributions***

[GFreeMarker 0.2](#)

[GFreeMarker 0.2 \(javadoc\)](#)

[GFreeMarker 0.2 \(sources\)](#)

#### ***Installing***

Pending.

#### ***Pre-requisites***

None.

### **Documentation**

#### ***How does it work ?***

Imagine your customer requires an URL Encoding text transform, and that FreeMarker does not offer this transformation. Then, you would just need to :

- Create a plugin named urlencoder that implements the IGroovyFreeMarkerPlugin interface
- Copy this plugin into the Groovy FreeMarker template engine plugins directory

That's all ! Now your customer can access it quite directly in the template. For example :

```
<@groovy plugin="urlencoder">this is an  
expression that will be converted to its URL  
Encoding form</@groovy>
```

#### ***Writing a plugin***

Here's a sample plugin which converts a string through URL encoder :

```
import
org.codehaus.groovy.gfreemarker.IGroovyFreeM
arkerPlugin

class urlencoder implements
IGroovyFreeMarkerPlugin {
    String transform(Map params, String
content) {
        URLEncoder.encode(content);
    }
}
```

#### *Using the template engine*

Here's a sample code which shows how easy it is to embed FreeMarker as a template engine for Groovy.

```
import
org.codehaus.groovy.gfreemarker.FreeMarkerTe
mplateEngine

def tpl = '''
Hello, ${user.name}
<@groovy plugin="urlencoder" mode=user>this
is a test ${user.name}</@groovy>'''
def engine = new
FreeMarkerTemplateEngine("plugins")
def binding = ["user" : ["name":"cedric"]]
println
engine.createTemplate(tpl).make(binding)
```

This code assumes a directory called *plugins* is in the working directory. The template engine will dynamically load the "urlencoder" class.

## Developers

### *Source Control*

<http://svn.codehaus.org/groovy-contrib/gfreemarker/trunk/>

### *Building*

GFreeMarker uses Maven2 as its build tool. Follow the instructions at <http://maven.apache.org> or install an IDE which has Maven 2 support (I personally use [IntelliJ IDEA 7](#)).

### *Contributing*

Feel free to contribute, as I don't have much time to spend on this project.

## Community

### *Mailing List(s)*

<http://groovy.codehaus.org/Mailing+Lists>

### *Issue tracker*

N/A

## Google Data Support

The GData module makes using the Google Data APIs easier from within Groovy.

Google supports a general data access protocol [GData](#). They supply a [Java library](#) which wraps the protocol and provides a higher level API. Groovy programs can make use of this library "as is". However, it's not very Groovy! The GData module provides a couple of Categories which make it a lot easier to manipulate the Google data.

Here's an example of reading events from Google Calendar and then adding an event to the calendar:

```
import com.google.gdata.client.*
import com.google.gdata.client.calendar.*
import com.google.gdata.data.*
import com.google.gdata.data.extensions.*
import com.google.gdata.util.*

import groovy.google.gdata.GDataCategory
import
org.codehaus.groovy.runtime.TimeCategory
```

```

def myId = System.properties.id
def myPassword = System.properties.pass
def feedUrl =
"http://www.google.com/calendar/feeds/$myId/
private/full"

use (TimeCategory, GDataCategory) {
    def myService = new
CalendarService("codehausGroovy-groovyExamp1
eApp-1")

    myService.userCredentials = [myId,
myPassword]

    //
    // List existing entries
    //

    //
    // Get at most 20 events in the period
starting 1 week ago and ending 4 weeks in
the future
    //
    myService.getFeed(feedUrl, 1.week.ago,
4.weeks.from.today, 20).entries.each {entry
->
        entry.times.each {time ->
            println "${entry.title.text}
From: ${time.startTime.toUiString()} To:
${(time.endTime.toUiString())}"

```

```

        }
    }

    //
    //  Get at most 20 events in the period
starting 1 year ago lasting 2 years
    //
    myService.getFeed(feedUrl, 1.year.ago,
2.years, 20).entries.each {entry ->
        entry.times.each {time ->
            println "${entry.title.text}
From: ${time.startTime.toUiString()} To:
${(time.endTime.toUiString())}"
        }
    }

    //
    //  Add an entry
    //

    // Use standard groovy magic to set the
properties after construction
    def me = new Person(name: "John Wilson",
email: "tugwilson@gmail.com", uri:
"http://eek.ook.org")

    //
    //  Need special magic in the
GDataCategory to do this
    //

```

```
// title and content are treated as
plain text. If you want XHTML or XML then
pass a closure or a
    // Buildable object and it will run it
in a builder context
    //
    // Note that we can't use title and
content in the Category as they are already
properties of the class.
    // Later I'll create a custom MetaClass
for EventEntry which will let us use these
names. Until then we'll mangle them
    //
    // author can be a single Person or a
list of Person
    //
    // time can be a single When or a list
of them
    //
    def newEntry = new EventEntry(title1:
"This is a test event", content1: "this is
some content", author: me,
                                     time: new
When(start: 1.hour.from.now, end:
2.hours.from.now))
```

```
    myService.insert(feedUrl, newEntry)
}
```

## Gram

Gram is a simple xdoclet-like tool for processing doclet tags or Java 5 annotations in source code or bytecode and auto-generating files, data or resources.

Gram = Groovy + [JAM](#). JAM does all the hard work of abstracting away the details between annotations and doclet tags and handling Java 1.4 and 5 compliance. Groovy takes care of the scripting, code generation & templating. Gram is the little tidy bit of code in between.

The sources can be found here : [Gram](#)

## Using Gram

You can use the Gram class as a main() and run it from your IDE if you wish. There is a GramTask as well for using it inside Ant.

Often since JAM depends on Sun's doclet stuff, you can have issues running the GramTask inside Maven and sometimes Ant. So I tend to run the Gram command line tool from inside an Ant build. e.g.

```
<java classname="org.codehaus.gram.Gram"
fork="true">
    <classpath refid="tool.classpath"/>

    <!-- the directory where the source
code lives -->
    <arg value="src/java"/>

    <!-- the groovy script to run to
generate stuff -->
    <arg
value="src/script/MyGram.groovy"/>
</java>
```

## Example script

Here's a simple example which just lists all the hibernate persistent classes in your source code



## Jar Dependencies

Gram depends on:

- The Groovy 'all' jar (groovy-all-\*.jar)
- JAM from the [Annogen](#) project
- and potentially: xml-apis.1.02b.jar and Sun's tools.jar

## Articles

You might find some more documentation in the form of blog posts by Andres Almiray:

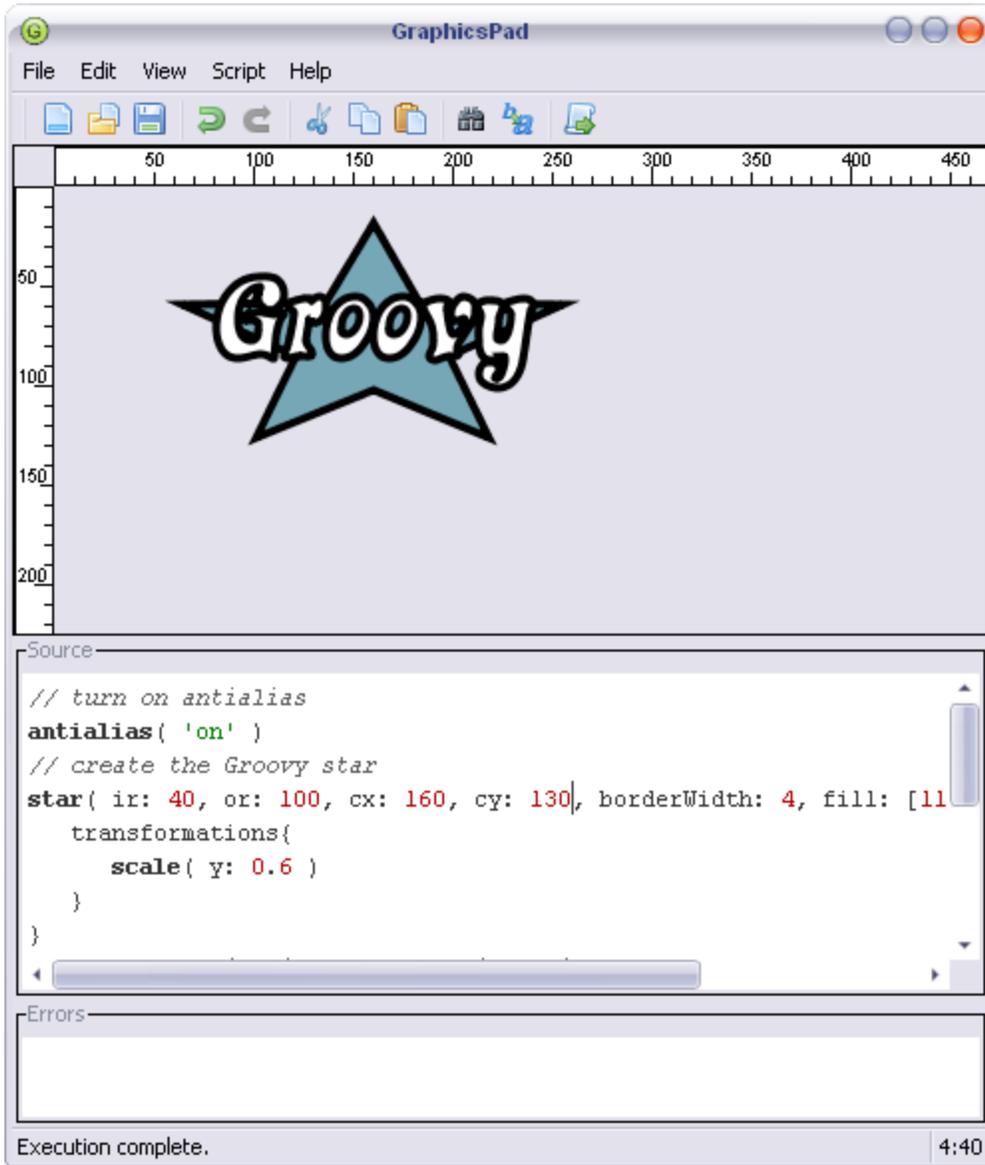
- [Getting started with Gram](#)
- [Going further with Gram](#)

## GraphicsBuilder

GraphicsBuilder is a Groovy builder for Java 2D

### Module Overview

GraphicsBuilder provides an easy way to create Java2D graphics in a similar manner as [JavaFX Script](#) but in a groovier way. Here is the Groovy logo (kind of) drawn with GraphicsBuilder using the built-in GraphicsPad application



This is the code used to draw the logo

```
// turn on antialias
antialias( 'on' )
// create the Groovy star
star( ir: 40, or: 100, cx: 160, cy: 120,
borderWidth: 4,
      fill: [118,167,183] as Color ){
  transformations{
    scale( y: 0.6 )
  }
}
// load the swingtime font, get it? Swing
def fontFile = new File("SWINM____.TTF")
font(
Font.createFont(Font.TRUETYPE_FONT,fontFile)
.deriveFont(50.0f) )
// draw the text
text( text: 'Groovy', borderWidth: 4, fill:
'white' ){
  transformations{
    translate( x:80, y:30 )
    scale( y: 1.2 )
  }
}
}
```

**Download**

*Distributions*

[graphicsbuilder-0.6.1 \(with dependencies\)](#)

[graphicsbuilder-0.6](#)

[graphicsbuilder-0.6-dependencies](#)

*Installing*

Drop all jar files into \$GROOVY\_HOME/lib  
Drop the scripts into \$GROOVY\_HOME/bin

Since Groovy 1.1-beta-3 the Windows NSIS Installer includes GraphicsBuilder as an additional module to install (<http://groovy.codehaus.org/Windows+NSIS-Installer>)

### **Pre-requisites**

GraphicsBuilder 0.6.0 requires Groovy 1.5.0+ and Java 1.6+  
Additional extensions have the following requisites

<b>Extension</b>	<b>Java Version</b>	<b>Extra libraries</b>
graphicsbuilder-ext-swingx	1.5+	swingx-0.9.2
graphicsbuilder-ext-svg	1.5+	batik-1.7 (the whole package)
graphicsbuilder-ext-swf	1.5+	transformSWF
graphicsbuilder-ext-jhlibs	1.5+	Filters.jar
graphicsbuilder-ext-jdk6	1.6+	
graphicsbuilder-ext-animation	1.6+	Timing Framework
graphicsbuilder-ext-substance	1.6+	Substance + Substance extras 5.0

### **GraphicsBuilder 0.6.1 requires Groovy 1.6-beta-2**

### **Documentation**

#### Tutorials

- [Alphabetical Index](#)
- [GraphicsBuilder - Animation](#)
  - [GraphicsBuilder - Animation - Bind](#)
- [GraphicsBuilder - GDK](#)
- [GraphicsBuilder - Groodles](#)
  - [GraphicsBuilder - Groodle1 - Screencast](#)
  - [GraphicsBuilder - Groodle2 - Screencast](#)
  - [GraphicsBuilder - Groodle3 - Screencast](#)
- [GraphicsBuilder - Groups](#)
- [GraphicsBuilder - Misc](#)
  - [GraphicsBuilder - Misc - Antialias](#)
  - [GraphicsBuilder - Misc - Clip](#)
  - [GraphicsBuilder - Misc - Color](#)
  - [GraphicsBuilder - Misc - Draw](#)
  - [GraphicsBuilder - Misc - Font](#)
  - [GraphicsBuilder - Misc - Image](#)
  - [GraphicsBuilder - Misc - RenderingHint](#)
- [GraphicsBuilder - Outlines](#)
  - [GraphicsBuilder - Outlines - CubicCurve](#)

- [GraphicsBuilder - Outlines - Line](#)
- [GraphicsBuilder - Outlines - Polyline](#)
- [GraphicsBuilder - Outlines - QuadCurve](#)
- [GraphicsBuilder - Paints](#)
  - [GraphicsBuilder - Paints - BorderPaint](#)
  - [GraphicsBuilder - Paints - ColorPaint](#)
  - [GraphicsBuilder - Paints - GradientPaint](#)
  - [GraphicsBuilder - Paints - LinearGradient](#)
  - [GraphicsBuilder - Paints - MultiPaint](#)
  - [GraphicsBuilder - Paints - Paint](#)
  - [GraphicsBuilder - Paints - RadialGradient](#)
  - [GraphicsBuilder - Paints - TexturePaint](#)
- [GraphicsBuilder - Shapes](#)
  - [GraphicsBuilder - Area - Add](#)
  - [GraphicsBuilder - Area - Intersect](#)
  - [GraphicsBuilder - Area - Shape](#)
  - [GraphicsBuilder - Area - Subtract](#)
  - [GraphicsBuilder - Area - Xor](#)
  - [GraphicsBuilder - Shapes - Arc](#)
  - [GraphicsBuilder - Shapes - Arrow](#)
  - [GraphicsBuilder - Shapes - Circle](#)
  - [GraphicsBuilder - Shapes - Cross](#)
  - [GraphicsBuilder - Shapes - Donut](#)
  - [GraphicsBuilder - Shapes - Ellipse](#)
  - [GraphicsBuilder - Shapes - Morph](#)
  - [GraphicsBuilder - Shapes - Path](#)
  - [GraphicsBuilder - Shapes - Pin](#)
  - [GraphicsBuilder - Shapes - Polygon](#)
  - [GraphicsBuilder - Shapes - Rays](#)
  - [GraphicsBuilder - Shapes - Rect](#)
  - [GraphicsBuilder - Shapes - RegularPolygon](#)
  - [GraphicsBuilder - Shapes - RoundRect](#)
  - [GraphicsBuilder - Shapes - Star](#)
  - [GraphicsBuilder - Shapes - Text](#)
  - [GraphicsBuilder - Shapes - Triangle](#)
  - [GraphicsBuilder - Shapes - Xpath](#)
- [GraphicsBuilder - Strokes](#)
  - [GraphicsBuilder - Strokes - BasicStroke](#)
  - [GraphicsBuilder - Strokes - CompositeStroke](#)
  - [GraphicsBuilder - Strokes - CompoundStroke](#)
  - [GraphicsBuilder - Strokes - Shapetroke](#)
  - [GraphicsBuilder - Strokes - Stroke](#)
  - [GraphicsBuilder - Strokes - TextStroke](#)
  - [GraphicsBuilder - Strokes - WobbleStroke](#)
  - [GraphicsBuilder - Strokes - ZigzagStroke](#)
- [Graphicsbuilder - SVG](#)
- [GraphicsBuilder - Swing](#)
- [GraphicsBuilder - Transformations](#)
- [GraphicsBuilder - Tutorials](#)

**Update:** all graphics operation support binding in the same way as [SwingBuilder](#) does.

**Developers**

Andres Almiray

### **Source Control**

<http://svn.codehaus.org/groovy-contrib/graphicsbuilder>

### **Building**

GraphicsBuilder uses Maven2 as its build tool, which means that if you want to build your own version from source you'll need to have it installed. Follow the instructions at <http://maven.apache.org>. Make sure to use maven 2.0.7 or higher.

Have fun!

### **Contributing**

Please contact the team members by e-mail or use the mailing lists =)

### **Community**

#### **Mailing List(s)**

<http://groovy.codehaus.org/Mailing+Lists>

#### **Issue tracker**

<http://jira.codehaus.org/secure/BrowseProject.jspp?id=10242>

### **Alphabetical Index**

[#A](#), [#B](#), [#C](#), [#D](#), [#E](#), [#F](#), [#G](#), [#H](#), [#I](#), [#J](#), [#K](#), [#L](#), [#M](#), [#N](#), [#O](#), [#P](#), [#Q](#), [#R](#), [#S](#), [#T](#), [#U](#), [#V](#), [#W](#), [#X](#), [Y](#), [#Z](#)

<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
add			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
opacity	o		

passThrough	pt	false	
<b>ambientLight</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>animate</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>antialias</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
key		antialiasing	
value		antialias on	true == [on,yes] false == [off,no]
<b>arc</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
close			
composite	c		
extent		90	
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	

start		0	
width	w	10	
x		0	
y		0	
<b>arrow</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
depth			
fill	f		
height	h	6	
opacity	o		
passThrough	pt	false	
rise			
width	w	10	
x		0	
y		0	
<b>average</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>

clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			
<b>background</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
color			
<b>balloon</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
anglePosition	ap		[start, end]
arc		5	
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	20	
opacity	o		
passThrough	pt	false	
tabDisplacement	td	0.5	
tabHeight	th		

tabLocation	tl	bottom	[bottom, left, right, top]
tabWidth	tw	5	
width	w	20	
x		0	
y		0	
<b>basicStroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asStroke			
cap			
color			
dash			
dashphase		0	
join			
miterlimit			
opacity	o		
width	w	1	
<b>bind</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>blur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
premultiplyAlpha			

useAlpha			
<b>borderPaint</b>			
Property	Alias	Default Value	Notes
<b>boxBlur</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
hRadius	hr		
iterations			
premultiplyAlpha			
radius	r		
vRadius	vr		
<b>brushedMetal</b>			
Property	Alias	Default Value	Notes
amount			
clip			
color			
enabled		true	
monochrome			
radius	r		
shine			
<b>butterflyShape</b>			
Property	Alias	Default Value	Notes
asImage	i	false	

asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>caustics</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
bgColor			
brightness			
clip			
dispersion			
enabled		true	
samples			
scale			
time			
turbulence			

<b>cellular</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
angle			
angleCoefficient			
clip			
colormap			
distancePower			
enabled		true	
f1			
f2			
f3			
f4			
gradientCoefficient			
gridType			
randomness			
scale			
stretch			
turbulence			
<b>check</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
background			
clip			

enabled		true	
foreground			
fuziness			
xScale			
yScale			
<b>chromaKey</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
bTolerance			
clip			
color			
enabled		true	
hTolerance			
sTolerance			
<b>chrome</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
bumpFunction			
bumpHeight			
bumpShape			
bumpSoftness			
bumpSource			
clip			
colorSource			
diffuseColor			

enabled		true	
environmentMap			
exposure			
material			
viewDistance			
<b>circle</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
opacity	o		
passThrough	pt	false	
<b>circleDistort</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
centreX	cx		
centreY	cy		
clip			
edgeAction			
enabled		true	

height			
interpolation			
radius	r		
spreadAngle			
<b>clip</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
shape			
<b>close</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>color</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
red	r		
green	g		
blue	b		
alpha	a		
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>colorPaint</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asPaint			
color			
red	r		
green	g		
blue	b		
alpha	a		

<b>colorScheme</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
colorTint			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
mixColor		Color[r=0,g=0,b=0]	
mixValue		0	
<b>composite</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
composite			
enabled		true	
transform			
<b>compositeStroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asStroke			
stroke1			
stroke2			
<b>compoundStroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asStroke			
operation			
stroke1			

stroke2			
<b>contour</b>			
Property	Alias	Default Value	Notes
clip			
contourColor			
enabled		true	
levels			
offset			
scale			
<b>contrast</b>			
Property	Alias	Default Value	Notes
brightness			
clip			
contrast			
enabled		true	
<b>convolve</b>			
Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
kernel			
premultiplyAlpha			
useAlpha			
<b>cross</b>			

Property	Alias	Default Value	Notes
angle			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
cx		5	
cy		5	
fill	f		
opacity	o		
passThrough	pt	false	
radius	r	5	
roundness			
width	w		
<b>crystallize</b>			
Property	Alias	Default Value	Notes
clip			
edgeColor			
edgeThickness			
enabled		true	
fadeEdges			
<b>cs</b>			alias for <b>colorScheme</b>

Property	Alias	Default Value	Notes
<b>cubicCurve</b>			
Property	Alias	Default Value	Notes
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
ctrlx1			
ctrlx2			
ctry1			
ctry2			
opacity	o		
passThrough	pt	false	
x1			
x2			
y1			
y2			
<b>curl</b>			
Property	Alias	Default Value	Notes
angle			
clip			
edgeAction			

enabled		true	
height	h		
interpolation			
radius	r		
transition			
width	w		
<b>curveTo</b>			
Property	Alias	Default Value	Notes
x1			
x2			
x3			
y1			
y2			
y3			
<b>despeckle</b>			
Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			
<b>detectEdges</b>			
Property	Alias	Default Value	Notes
clip			

enabled		true	
hEdgeMatrix			
vEdgeMatrix			
<b>diffuse</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
interpolation			
scale			
<b>diffusion</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
colorDither			
enabled		true	
levels			
matrix			
serpentine			
<b>dilate</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blackFunction			
clip			
colormap			
enabled		true	

iterations			
newColor			
threshold			
<b>displace</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
clip			
displacementMap			
edgeAction			
enabled		true	
interpolation			
<b>dissolve</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
density			
enabled		true	
softness			
<b>distantLight</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>dither</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
colorDither			
enabled		true	

levels			
matrix			
<b>dolphinShape</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>donut</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		

borderWidth	bw		
composite	c		
cx		20	
cy		20	
fill	f		
ir		10	
opacity	o		
or		20	
passThrough	pt	false	
sides			
<b>draw</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
keepTrans			
opacity	o		
passThrough	pt	false	
shape			
<b>dropShadow</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>

clip			
color	c		
enabled		true	
opacity	o		
shadowOnly		false	
size			
<b>ellipse</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
cx		10	
cy		5	
fill	f		
opacity	o		
passThrough	pt	false	
radiusx	rx	10	
radiusy	ry	5	
<b>emboss</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			

enabled		true	
<b>embossEdges</b>			
Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			
<b>equalize</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
<b>erode</b>			
Property	Alias	Default Value	Notes
blackFunction			
clip			
colormap			
enabled		true	
iterations			
newColor			
threshold			
<b>erodeAlpha</b>			
Property	Alias	Default Value	Notes
clip			

enabled		true	
radius	r		
softness			
threshold			
<b>exposure</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
exposure			
<b>fade</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
dimensions			
enabled		true	
fadeStart			
fadeWidth			
invert			
sides			
<b>fastBlur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
radius	r	3	
<b>fastMotionBlur</b>			

Property	Alias	Default Value	Notes
angle			
centreX	cx		
centreY	cy		
clip			
distance			
enabled		true	
rotation			
zoom			
<b>fbm</b>			alias for <b>fractalBrownian Motion</b>

Property	Alias	Default Value	Notes
amount			
angle			
basis			
basisType			
bias			
clip			
colormap			
enabled		true	
gain			
h			
lacunarity			
octaves			

operation			
scale			
stretch			
<b>feedback</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
centreX	cx		
centreY	cy		
clip			
distance			
enabled		true	
endAlpha			
iterations			
rotation			
startAlpha			
zoom			
<b>fieldWarp</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
clip			
edgeAction			
enabled		true	
inLines			
interpolation			

outLines			
power			
strength			
<b>fill</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
fillColor			
<b>filters</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>fishShape</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	

y		0	
<b>flare</b>			
Property	Alias	Default Value	Notes
baseAmount			
centre			
clip			
color			
dimensions			
enabled		true	
radius	r		
rayAmount			
ringAmount			
ringWidth			
<b>flip</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
operation			
<b>flush3D</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
<b>font</b>			
Property	Alias	Default Value	Notes

font		Font[family=Dialog,style=plain,size=12]	
<b>footShape</b>			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>fourColorFill</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
colorNE			
colorNW			
colorSE			
colorSW			
dimensions			

enabled		true	
<b>fractalBrownianMotion</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
angle			
basis			
basisType			
bias			
clip			
colormap			
enabled		true	
gain			
h			
lacunarity			
octaves			
operation			
scale			
stretch			
<b>fractalFlame</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
colorScheme1	cs1		
colorScheme2	cs2		
enabled		true	

ifs		Singularity	org.jvnet.substance.watermarkpack.flamefractal.Singularity
iterations	i	100000	
<b>gain</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
bias			
clip			
enabled		true	
gain			
<b>gamma</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
bGamma			
clip			
enabled		true	
gGamma			
gamma			
rGamma			
<b>gaussianBlur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
radius	r		

useAlpha			
<b>glint</b>			
Property	Alias	Default Value	Notes
amount			
blur			
clip			
colormap			
enabled		true	
glintOnly			
length			
threshold			
<b>glow</b>			
Property	Alias	Default Value	Notes
amount			
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
radius	r		
useAlpha			
<b>gradientColormap</b>			
Property	Alias	Default Value	Notes
<b>gradientPaint</b>			
Property	Alias	Default Value	Notes

asPaint			
color1	c1	Color[r=0,g=0,b=0]	
color2	c2	Color[r=255,g=255,b=255]	
cycle		false	
fit		true	
stretch		false	
x1		0	
x2		100	
y1		0	
y2		0	
<b>gradientWipe</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
density			
enabled		true	
invert			
mask			
softness			
<b>grayOut</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
<b>grayscale</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>

clip			
enabled		true	
<b>grayscaleColormap</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>group</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
opacity	o		
passThrough	pt	false	
<b>halftone</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
density			
enabled		true	
invert			
mask			
softness			
<b>hline</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>

x			
<b>hsbAdjust</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
bFactor			
clip			
enabled		true	
hFactor			
sFactor			
<b>iceCreamShape</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>image</b>			

Property	Alias	Default Value	Notes
asImage	i	false	
autoDrag		false	
classpath			
composite	c		
file			
image			
opacity	o		
passThrough	pt	false	
url			
x		0	
y		0	
<b>inFieldWarpLine</b>			
Property	Alias	Default Value	Notes
<b>interpolate</b>			
Property	Alias	Default Value	Notes
clip			
destination			
enabled		true	
interpolation			
<b>intersect</b>			
Property	Alias	Default Value	Notes
asImage	i	false	
asShape	s		

autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
opacity	o		
passThrough	pt	false	
<b>invert</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
<b>invertAlpha</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
<b>javaLnF</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
<b>kaleidoscope</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
angle2			
centreX	cx		

centreY	cy		
clip			
edgeAction			
enabled		true	
interpolation			
radius	r		
sides			
<b>knot</b>			
<b>lensBlur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
bloom			
bloomThreshold			
clip			
enabled		true	
radius	r		
sides			
<b>levels</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
highLevel			
highOutputLevel			
lowLevel			
lowOutputLevel			

<b>life</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blackFunction			
clip			
colormap			
enabled		true	
iterations			
newColor			
<b>lightRays</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
centreX	cx		
centreY	cy		
clip			
colormap			
distance			
enabled		true	
raysOnly			
rotation			
strength			
threshold			
zoom			
<b>lights</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>

bumpFunction			
bumpHeight			
bumpShape			
bumpSoftness			
bumpSource			
clip			
colorSource			
diffuseColor			
enabled		true	
environmentMap			
material			
viewDistance			
<b>line</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
opacity	o		
passThrough	pt	false	
x1		0	
x2		0	

y1		0	
y2		10	
<b>lineTo</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
x			
y			
<b>linearColormap</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>linearGradient</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asPaint			
linkTo			
x1		0	
x2		100	
y1		0	
y2		0	
<b>lookup</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
colormap			
enabled		true	
<b>mapCoordinates</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			

edgeAction			
enabled		true	
interpolation			
xMapFunction			
yMapFunction			
<b>marble</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
interpolation			
xScale			
yScale			
<b>marbleTexture</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
clip			
colormap			
enabled		true	
scale			
stretch			
turbulence			
turbulenceFactor			
<b>maskColor</b>			

Property	Alias	Default Value	Notes
clip			
enabled		true	
mask			
<b>material</b>			
Property	Alias	Default Value	Notes
<b>matrix</b>			
Property	Alias	Default Value	Notes
enabled		true	
interpolation			
m00		1	
m01		0	
m02		0	
m10		0	
m11		1	
m12		0	
<b>maximum</b>			
Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			
<b>median</b>			

Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			
<b>meshWarp</b>			

Property	Alias	Default Value	Notes
clip			
destGrid			
edgeAction			
enabled		true	
frames			
interpolation			
morphImage			
sourceGrid			
time			
<b>minimum</b>			

Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			

<b>mirror</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
centreY	cy		
clip			
enabled		true	
gap			
opacity	o		
<b>morph</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
end	e		
fill	f		
morph	m	0	
opacity	o		
passThrough	pt	false	
start	a		
<b>mosaic</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blockSize			

clip			
edgeAction			
enabled		true	
interpolation			
<b>motionBlur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
clip			
distance			
enabled		true	
premultiplyAlpha			
rotation			
wrapEdges			
zoom			
<b>moveTo</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
x		0	
y		0	
<b>multiPaint</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>noise</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
clip			

density			
distribution			
enabled		true	
monochrome			
<b>noiseTexture</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
angle			
bias			
clip			
colormap			
enabled		true	
function			
gain			
operation			
scale			
stretch			
turbulence			
<b>oil</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
interpolation			

levels			
range			
<b>outFieldWarpLine</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>outline</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blackFunction			
clip			
colormap			
enabled		true	
iterations			
newColor			
<b>paint</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>path</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
close			
composite	c		
fill	f		

opacity	o		
passThrough	pt	false	
winding			
<b>perspective</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
x0		0	
x1		0	
x2		0	
x3		0	
y0		0	
y1		0	
y2		0	
y3		0	
<b>pin</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		

cx		5	
cy		5	
fill	f		
height	h		
opacity	o		
passThrough	pt	false	
radius		5	
<b>pinch</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
angle			
centreX	cx		
centreY	cy		
clip			
edgeAction			
enabled		true	
interpolation			
radius	r		
<b>plasma</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
colormap			
enabled		true	
scaling			

seed			
turbulence			
useColormap			
useImageColors			
<b>pointLight</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>pointlize</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeColor			
edgeThickness			
enabled		true	
fadeEdges			
fuzziness			
<b>polar</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
interpolation			
type			
<b>polygon</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	

asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
opacity	o		
passThrough	pt	false	
points			
<b>polyline</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
opacity	o		
passThrough	pt	false	
points			
<b>posterize</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	

numLevels			
<b>premultiply</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
<b>props</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>quadCurve</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
ctrlx			
ctry			
opacity	o		
passThrough	pt	false	
x1			
x2			
y1			
y2			
<b>quadTo</b>			

Property	Alias	Default Value	Notes
x1			
x2			
y1			
y2			
<b>quantize</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
<b>quilt</b>			
Property	Alias	Default Value	Notes
a			
b			
c			
clip			
colormap			
d			
enabled		true	
iterations			
k			
<b>raceCarShape</b>			
Property	Alias	Default Value	Notes
asImage	i	false	
asShape	s		

autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>radialGradient</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
absolute		false	
asPaint			
cx			
cy			
cycle		nocycle	
fx			
fy			
linkTo			
radius	r		
<b>rays</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>

angle		0	
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
cx		5	
cy		5	
extent		0.5	
fill	f		
opacity	o		
passThrough	pt	false	
radius	r	5	
rays		2	
rounded		false	
<b>rect</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
arcHeight	ah		
arcWidth	aw		
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		

borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>reflection</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blurEnabled			
blurRadius			
clip			
enabled		true	
length			
opacity			
reflectionOnly		false	
<b>regularPolygon</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle		0	
asImage	i	false	
asShape	s		
autoDrag		false	

borderColor	bc		
borderWidth	bw		
composite	c		
cx		5	
cy		5	
fill	f		
opacity	o		
passThrough	pt	false	
radius	r	5	
sides		3	
<b>renderingHint</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
key	k		
value	v		
<b>rescaleColors</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
scale			
<b>rgba</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
red	r		
green	g		
blue	b		

alpha	a		
<b>rgbAdjust</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
bFactor			
clip			
enabled		true	
gFactor			
rFactor			
<b>rhinoShape</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>ripple</b>			

Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
interpolation			
waveType			
xAmplitude			
xWavelength			
yAmplitude			
yWavelength			
<b>rotate</b>			
Property	Alias	Default Value	Notes
angle		0	
x			
y			
<b>roundRect</b>			
Property	Alias	Default Value	Notes
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
bottomLeftHeight	blh		
bottomLeftWidth	blw		

bottomRightHeight	brh		
bottomRightWidth	brw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
topLeftHeight	tlh		
topLeftWidth	tlw		
topRightHeight	trh		
topRightWidth	trw		
width	w	10	
x		0	
y		0	
<b>saturation</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
clip			
enabled		true	
<b>scale</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
enabled		true	
interpolation			
x		1	

y		1	
<b>scratch</b>			
Property	Alias	Default Value	Notes
angle			
angleVariation			
clip			
color			
density			
enabled		true	
length			
seed			
width	w		
<b>sgaussianBlur</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
radius	r	3	
<b>shade</b>			
Property	Alias	Default Value	Notes
bumpFunction			
bumpHeight			
bumpSource			
clip			
enabled		true	

environmentMap			
<b>shadow</b>			
Property	Alias	Default Value	Notes
addMargins			
angle			
clip			
distance			
enabled		true	
opacity			
radius			
shadowColor			
shadowOnly			
<b>shape</b>			
Property	Alias	Default Value	Notes
<b>shapeBurst</b>			
Property	Alias	Default Value	Notes
clip			
colormap			
enabled		true	
factor			
invert			
merge			
type			
useAlpha			

<b>shapeStroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
advance		10	
asStroke			
shapes			
<b>shapeTo</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
connect		false	
shape			
<b>sharpen</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
edgeAction			
enabled		true	
premultiplyAlpha			
useAlpha			
<b>shatter</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
centreX	cx		
centreY	cy		
clip			
enabled		true	
endAlpha			
iterations			

rotation			
startAlpha			
tile			
transition			
zoom			
<b>shine</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle			
bevel			
brightness			
clip			
distance			
enabled		true	
radius	r		
shadowOnly			
shineColor			
softness			
<b>skeletonize</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blackFunction			
clip			
colormap			
enabled		true	
iterations			

newColor			
<b>skew</b>			
Property	Alias	Default Value	Notes
enabled		true	
interpolation			
x		0	
y		0	
<b>smartBlur</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
hRadius	hr		
threshold			
vRadius	vr		
<b>smear</b>			
Property	Alias	Default Value	Notes
angle			
background			
clip			
colormap			
density			
distance			
enabled		true	
mix			

shape			
<b>smooth</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
<b>solarize</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
<b>sparkle</b>			
Property	Alias	Default Value	Notes
amount			
clip			
color			
enabled		true	
radius	r		
randomness			
rays			
<b>spectrumColormap</b>			
Property	Alias	Default Value	Notes
<b>sphereDistort</b>			
Property	Alias	Default Value	Notes
centreX	cx		
centreY	cy		

clip			
edgeAction			
enabled		true	
interpolation			
radius	r		
refractionIndex			
<b>splineColormap</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>spotLight</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>stackBlur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
iterations		3	
radius	r	3	
<b>stamp</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
black			
clip			
color1	c1		
color2	c2		
enabled		true	
radius	r		

softness			
threshold			
white			
<b>star</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
angle		0	
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
count		5	
cx		5	
cy		5	
fill	f		
ir		3	
opacity	o		
or		8	
passThrough	pt	false	
<b>stegosaurusShape</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		

autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h	10	
opacity	o		
passThrough	pt	false	
width	w	10	
x		0	
y		0	
<b>stop</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
offset	s		
color	c		
opacity	o		
<b>stroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
stroke			
<b>subtract</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	

borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
opacity	o		
passThrough	pt	false	
<b>swim</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amount			
angle			
clip			
edgeAction			
enabled		true	
interpolation			
scale			
stretch			
time			
turbulence			
<b>text</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		

borderWidth	bw		
composite	c		
fill	f		
halign	ha	left	
opacity	o		
passThrough	pt	false	
spacing		0	
text	t	Groovy	
valign	va	bottom	
x		0	
y		0	
<b>textStroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
font		Font[family=Dialog,style=plain,size=12]	
repeat		false	
stretch		false	
text		Groovy	
<b>texturePaint</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
absolute		false	
asPaint			
classpath			
file			

height	h		
image			
url			
width	w		
x		0	
y		0	
<b>threshold</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
black			

clip			
color1	c1		
color2	c2		
enabled		true	
lowerThreshold			
upperThreshold			
white			

<b>transform</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
enabled		true	
interpolation			
<b>transformations</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>translate</b>			

Property	Alias	Default Value	Notes
enabled		true	
interpolation			
x		0	
y		0	
<b>transparency</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
opacity	o		
<b>triangle</b>			
Property	Alias	Default Value	Notes
angle			
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		
composite	c		
fill	f		
height	h		
opacity	o		
passThrough	pt	false	
rightAngleAt			

rotateAtCenter			
width	w	10	
x		0	
y		0	
<b>twirl</b>			
Property	Alias	Default Value	Notes
angle			
centreX	cx	0.5	
centreY	cy	0.5	
clip			
edgeAction			
enabled		true	
interpolation			
radius	r		
<b>unpremultiply</b>			
Property	Alias	Default Value	Notes
clip			
enabled		true	
<b>unsharp</b>			
Property	Alias	Default Value	Notes
clip			
edgeAction			
enabled		true	
premultiplyAlpha			

radius	r		
useAlpha			
<b>variableBlur</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
blurMask			
clip			
enabled		true	
hRadius	hr		
iterations			
premultiplyAlpha			
radius	r		
vRadius	vr		
<b>viewBox</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>vline</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
y			
<b>warpGrid</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
<b>water</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amplitude			
centreX	cx		
centreY	cy		

clip			
edgeAction			
enabled		true	
interpolation			
phase			
radius	r		
wavelength			
<b>weave</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
clip			
enabled		true	
roundThreads			
shadeCrossings			
useImageColors			
xGap			
xWidth			
yGap			
yWidth			
<b>wobbleStroke</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
amplitude		2	
asStroke			
detail		2	
<b>wood</b>			

Property	Alias	Default Value	Notes
angle			
clip			
colormap			
enabled		true	
fibres			
gain			
rings			
scale			
stretch			
turbulence			
<b>xarcTo</b>			

Property	Alias	Default Value	Notes
angle			
largeArc			
rx			
ry			
sweep			
x			
y			
<b>xclose</b>			

Property	Alias	Default Value	Notes
<b>xcurveTo</b>			
Property	Alias	Default Value	Notes

x1			
x2			
x3			
y1			
y2			
y3			
<b>xhline</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
x			
<b>xlineTo</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
x			
y			
<b>xmoveTo</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
x		0	
y		0	
<b>xor</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage	i	false	
asShape	s		
autoDrag		false	
borderColor	bc		
borderWidth	bw		

composite	c		
fill	f		
opacity	o		
passThrough	pt	false	
<b>xpath</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
asImage		false	
asShape			
autoDrag		false	
borderColor			
borderWidth			
close			
composite			
fill			
opacity			
passThrough		false	
winding			
<b>xquadTo</b>			
<b>Property</b>	<b>Alias</b>	<b>Default Value</b>	<b>Notes</b>
x1			
x2			
y1			
y2			
<b>xshapeTo</b>			

Property	Alias	Default Value	Notes
connect		false	
shape			
<b>xvline</b>			
Property	Alias	Default Value	Notes
y			
<b>zigzagStroke</b>			
Property	Alias	Default Value	Notes
amplitude		10	
asStroke			
stroke			
wavelength		10	

## GraphicsBuilder - Animation

[animate](#)

[bind](#)

### GraphicsBuilder - Animation - Bind

Bind currently follows the same rules as [SwingBuilder.bind](#).

## GraphicsBuilder - GDK

GraphicsBuilder provides new methods for existing Java2D classes.

java.awt.Color

Color <b>derive</b> ( Map )	creates a new color based on this color. Map must contain any of [red,green,blue,alpha]
-----------------------------	--

java.awt.BasicStroke

BasicStroke <b>derive</b> ( Map )	creates a new stroke based on this stroke. Map must contain any of [width,cap,join,miterlimit,dash,dashphase]
-----------------------------------	--

java.awt.Shape

Area <b>plus</b> ( Shape )	applies Area.add	enables + operator
Area <b>minus</b> ( Shape )	applies Area.subtract	enables - operator
Area <b>and</b> ( Shape )	applies Area.intersect	enables & operator
Area <b>xor</b> ( Shape )	applies Area.xor	enables ^ operator

**Note:** [shapes](#) also have these 4 operators enabled

## GraphicsBuilder - Groodles

Groodles are Groovy's version of [Joshua Marinacci's doodles](#)

- Groodle #1 - MouseEnter/MouseExit triggered animations [\[v1|v2\]](#) [\[code\]](#) [\[screencast\]](#)
- Groodle #2 - Custom shape and opacity triggers [\[v1\]](#) [\[code\]](#) [\[screencast\]](#)
- Groodle #3 - Drag&Drop [\[v1\]](#) [\[code\]](#) [\[screencast\]](#)

## GraphicsBuilder - Groodle1 - Screencast

Here is the code corresponding to this screencast:

```

import groovy.swing.SwingBuilder
import groovy.swing.j2d.GraphicsBuilder
import groovy.swing.j2d.GraphicsPanel

def gb = new GraphicsBuilder()

def go = gb.group {
    def foreground = color(r:1,g:1)
    def background = color('black')
    (0..9).each { i ->
        (0..19).each { j ->
            def cell = "c_{i}_{j}"
            rect( x:20*j, y:20*i, w: 20, h: 20,
bc: background,
                f: animate(
[foreground,background],
                                startValue:

```

```
background,
                                duration: 5000,
start: false,
                                id: "$cell" ),
mouseEntered: {e ->
                gb."$cell".restart()
              }
            )
        }
    }
}
```

```
def swing = SwingBuilder.build {
    frame( title: "Groodles", size:
[410,230],
          locationRelativeTo: null, show:
true ){
        panel( new GraphicsPanel(),
```

```
graphicsOperation: go )
    }
}
```

GraphicsBuilder - Groodle2 - Screencast

```
import groovy.swing.SwingBuilder
import groovy.swing.j2d.GraphicsBuilder
import groovy.swing.j2d.GraphicsPanel

def gb = new GraphicsBuilder()

def go = gb.group {
    antialias on
    balloon( x:20, y:20, w:100, bw:2, bc:
        'black', arc: 10,
            tabWidth:animate(0..20, duration:300,
start:no,id:'at',startDelay:1000),
            height:
animate(0..100,duration:1000,start:no,id:'ah
'),
            arc:    animate(0..10,
duration:1000,start:no,id:'aa'),

opacity:animate(0f..1f,duration:1000,start:n
o,id:'o') ){
    // let's use a multipaint to fill the
shape
    multiPaint {
        // base color
```

```

        colorPaint( 'blue' )
        // highlights
        linearGradient( x2: 0, y2:50 ){
            stop( s: 0, c:
color('white').derive(a:0.75) )
            stop( s: 0.66, c:
color('white').derive(a:0) )
        }
    }
}

// this group is the ballon popup
group( o: 0, id: 'popup', f: 'blue', bc: no
){
    balloon( x:140, y: 40, w: 80, h: 60,
tabWidth: 16, arc: 10,
            f: 'white', bc: 'black', bw: 2 )

    font( size: 20, style: 'bold' )
    text( x: 158, y: 50, text: 'Click' )
    text( x: 164, y: 70, text: 'me!' )
}

def normal = color('red')
def highlight = color('white')
def pressed = color('green')

// animation trigger
star( cx: 180, cy: 130, f: normal, or: 15,
ir: 8, id: 'star',
      mousePressed: {e ->

```

```
        star.fill = pressed
        at.restart()
        ah.restart()
        aa.restart()
        o.restart()
    },
    mouseReleased: {e-> star.fill =
highlight},
    mouseEntered: {e->
        star.fill = highlight
        popup.opacity = 1
    },
    mouseExited: {e->
        star.fill = normal
        popup.opacity = 0
    })
}

def swing = SwingBuilder.build {
    frame( title: "Groodle #2", size:
[240,200],
        locationRelativeTo: null, show:
true ){
        panel( new GraphicsPanel(),
```

```
graphicsOperation: go )
    }
}
```

## GraphicsBuilder - Groovy3 - Screencast

⚠ graphicsbuilder 0.6-SNAPSHOT is required to run this example

```
import groovy.swing.SwingBuilder
import groovy.swing.j2d.*

def gr = new GraphicsRenderer()
def gb = gr.gb

def go = gb.group {
def width = 150
def height = 150
def cx = width/2 + 10
def cy = height/2 + 10

antialias on
group( name: "frogface" ){
    circle( cx: cx, cy: cy, r: width/2, bc: no
){
    multiPaint {
        colorPaint( color('green') )
        radialGradient( cx: cx, cy: cy, r:
width/2 ) {
            stop( s: 0, c: color(r: 6, g:
160, b: 76, a: 127) )
            stop( s: 1, c: color(a: 204) )
```

```

        }
        radialGradient( cx: cx, cy: cy,
                        fx: 55, fy: 35, r:
width/1.4 ){
            stop( s: 0, c:
color('white').derive(a:0.5) )
            stop( s: 0.5, c:
color('white').derive(a:0) )
        }
    }
}

// left eye
circle( cx: 40, cy: 50, r: 20, bc:
'black', f: 'white' )
    circle( cx: 43, cy: 53, r: 12, bc:
'none', f: 'black' )

// right eye
circle( cx: 130, cy: 50, r: 20, bc:
'black', f: 'white' )
    circle( cx: 127, cy: 53, r: 12, bc:
'none', f: 'black' )

// nostrils
circle( cx: 75, cy: 80, r: 4, bc: 'black',
f: 'black' )
    circle( cx: 95, cy: 80, r: 4, bc: 'black',
f: 'black' )

// mouth

```

```
    ellipse( cx: 85, cy: 120, rx: 30, ry: 10,
bc: 'black', f: 'black' ){
    transformations { rotate( x: 85, y: 120,
angle: -10 ) }
    }

    transformations {
    translate( x: 250, y: 60 )
    }
}

// eye patch
add( bc: 'black', f: 'black', id:
'eyepatch', autoDrag: yes ) {
    rect( x: 10, y: 55, w: 150, h: 12 )
    circle( cx: 40, cy: 80, r: 20 )
    filters { dropShadow( name: 'shadow',
enabled: no ) }
}

// goatee
triangle( x: 60, y: 210, w: 40, angle: 180,
f: 'brown', id: 'goatee',
    autoDrag: yes ){
    filters { dropShadow( name: 'shadow',
enabled: no ) }
}

// moustache
add( f: 'brown', id: 'moustache', autoDrag:
yes ){
```

```

    ellipse( cx: 65, cy: 140, rx: 25, ry: 8
  ){
    transformations { rotate( x: 65, y:
140, angle: -12 ) }
    }
    ellipse( cx: 105, cy: 140, rx: 25, ry: 8
  ){
    transformations { rotate( x: 105, y:
140, angle: 12 ) }
    }
    filters { dropShadow( name: 'shadow',
enabled: no ) }
  }

// hat
add( f: 'orange', id: 'hat', autoDrag: yes
  ){
  triangle( x: 70, y: 230, w: 140, h: 40 )
  rect( x: 100, y: 180, w: 80, h: 40 )
  filters {
    shapeBurst( merge: true, type: 'up' ){
      linearColormap( color1:
color('orange').darker(), color2: 'orange' )
    }
    lights()
    dropShadow( name: 'shadow', enabled:
no )
  }
}

swingView {

```

```
    button( 'Reset', x: 5, y: 5,
actionPerformed: {e ->

['eyepatch', 'goatee', 'moustache', 'hat'].each
{
    gb."$it".txs.clear()
    }
})
}

}

def startDrag = { e ->
    def shape = e.target
    shape.borderWidth = 3
    shape.borderColor = 'red'
    shape.filters['shadow'].enabled = true
}

def endDrag = { e ->
    def shape = e.target
    shape.borderWidth = 1
    shape.borderColor = 'black'
    shape.filters['shadow'].enabled = false
}

['eyepatch', 'goatee', 'moustache', 'hat'].each
{
    def shape = gb."$it"
    shape.mousePressed = startDrag
    shape.mouseReleased = endDrag
    shape.mouseExited = endDrag
```

```
}
```

```
def swing = SwingBuilder.build {  
    frame( title: 'Groodle #3', size:  
[500,320],  
            locationRelativeTo: null, show:  
true ){  
    panel( new GraphicsPanel(),
```

```
graphicsOperation: go )
  }
}
```

## GraphicsBuilder - Groups

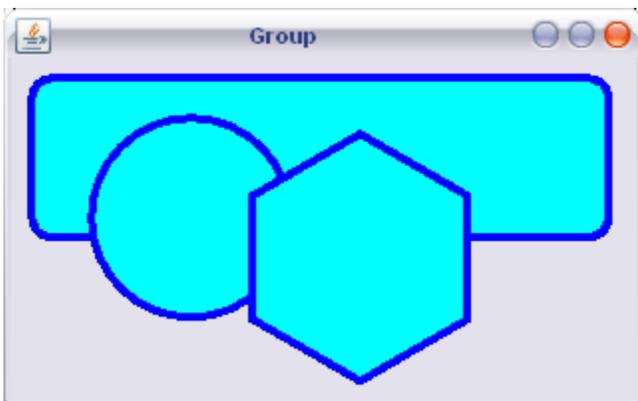
Groups are useful for arranging different operations in a single set. Groups may also define the following properties that can be overridden by any nested operation:

- `borderColor`: defines the color of the shape's outline
- `borderWidth`: defines the thickness of the shape's outline
- `fill`: defines the color, paint or gradient to fill the shape's content
- `opacity`: controls how much of the group is visible, value must be in the range [0..1], default is 1.

### Examples

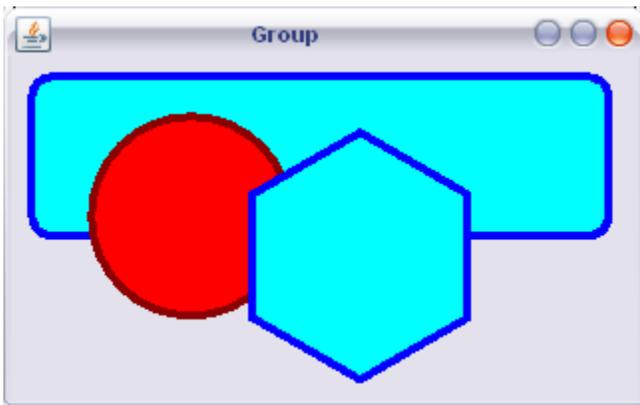
Borders and fill are passed to each children.

```
group( color: 'blue', strokeWidth: 4, fill:
'cyan' ){
  rect( x: 10, y: 10, width: 290, height:
80, arcWidth: 20, arcHeight: 20 )
  circle( cx: 90, cy: 80, radius: 50 )
  polygon(points: [175, 38, 229, 69, 229,
131, 175, 162, 121, 131, 121, 69])
}
```



Border and fill is overridden by one child.

```
group( color: 'blue', strokeWidth: 4, fill:
'cyan' ){
    rect( x: 10, y: 10, width: 290, height:
80, arcWidth: 20, arcHeight: 20 )
    circle( cx: 90, cy: 80, radius: 50,
borderColor: 'darkRed', fill: 'red' )
    polygon(points: [175, 38, 229, 69, 229,
131, 175, 162, 121, 131, 121, 69])
}
```



### GraphicsBuilder - Misc

[Antialias](#)

[Clip](#)

[Color](#)

[Draw](#)

[Font](#)

[Image](#)

[RenderingHint](#)

### GraphicsBuilder - Misc - Antialias

Enables or disables antialiasing. This is a shortcut for [renderingHint](#) as antialiasing is a very common option. The long way is

```
renderingHints( key: 'antialiasing', value:
'antialias on' )
```

It accepts the following values [false, true, 'off', 'on']

### GraphicsBuilder - Misc - Clip

Sets the current clip to an specified [shape](#). It accepts [transformations](#) too.

#### Properties

Property	Default	Notes
shape		a <a href="#">shape</a> or a java.awt.Shape

### GraphicsBuilder - Misc - Color

Creates a java.awt.Color based on the set of properties specified.

#### Properties

Property	Default	Notes
red	0	
green	0	
blue	0	
alpha	255	

You can wrap a color literal like 'white' or 'darkGreen' and get the corresponding java.awt.Color

### GraphicsBuilder - Misc - Draw

Renders [shapes](#) and [outlines](#). It is useful when reusing a previous saved shape/outline marked with `asShape=true` for example. It accepts [transformations](#), [paints](#) and all the properties of shapes/outlines, though it won't modify the original shape/outline. Draw will automatically translate the shape to the world origin (0,0).

#### Properties

Property	Default	Notes
shape		a <a href="#">shape</a> , an <a href="#">outline</a> or a java.awt.Shape

### GraphicsBuilder - Misc - Font

Creates a java.awt.Font based on the specified properties.

#### Properties

Property	Default	Notes
face		

style	<b>Font.PLAIN</b>	any combination of Font.[PLAIN,BOLD,ITALIC] or a   separated string of [plain,bold,italic]
size	<b>12</b>	

### GraphicsBuilder - Misc - Image

Draws an image. An Image operation can work as a wrapper for another image operation, in the same way that [paint](#) works. It also accepts [transformations](#).

#### Properties

Property	Default	Notes
x	<b>0</b>	
y	<b>0</b>	
image		must be a java.awt.Image, an image or shape operation
classpath		String
url		URL or String
file		File or String
opacity	<b>1</b>	controls how much of the image is visible, value must be in the range [0..1]
asImage	<b>false</b>	if true will skip drawing the image right away, useful for texture painting

You must specify at least any of [image,classpath,url,file]

#### Example

```
image( url:
' http://groovy.codehaus.org/download/attachm
ents/1866/groovydukemed.jpg ' )
```



## GraphicsBuilder - Misc - RenderingHint

Assigns hints to the Java2D pipeline useful for rendering.

### Properties

Property	Default	Notes
key		
value		

Key may have any value of: [ "antialiasing", "rendering", "dithering", "text antialiasing", "text lcd contrast", "fractionalmetrics", "interpolation", "alpha interpolation", "color rendering", "stroke control" ]

Value may have any value of: [ "antialias on", "antialias off", "antialias default", "render speed", "render quality", "render default", "dither disable", "dither enable", "dither default", "text antialias on", "text antialias off", "text antialias default", "text antialias gasp", "text antialias lcd hrgb", "text antialias lcd hbgr", "text antialias lcd vrgb", "text antialias lcd vbgr", "fractionalmetrics off", "fractionalmetrics on", "fractionalmetrics default", "interpolation nearest neighbor", "interpolation bilinear", "interpolation bicubic", "alpha interpolation speed", "alpha interpolation quality", "alpha interpolation default", "color render speed", "color render quality", "color render default", "stroke default", "stroke normalize", "stroke pure" ]

## GraphicsBuilder - Outlines

All outlines support the following additional properties

- `borderColor`: defines the color of the shape's outline. If false, the outline will not be drawn.
- `borderWidth`: defines the thickness of the shape's outline
- `opacity`: controls how much of the outline is visible, value must be in the range [0..1], default is 1.
- `asShape`: creates the shape but does not render to the screen, useful for mixing complex shapes

### Alphabetical List

[cubicCurve](#)

[line](#)

[polyline](#)

[quadCurve](#)

## GraphicsBuilder - Outlines - CubicCurve

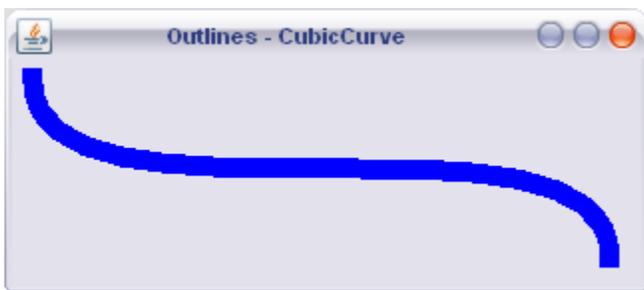
Draws a cubic parametric curve segment.

*Properties*

Property	Default	Notes
x1		
y1		
ctrlx1		
ctry1		
ctrlx2		
ctry2		
x2		
y2		

*Example*

```
cubicCurve( x1: 10, y1: 10, ctrlx1: 10,  
ctrly1: 100, ctrlx2: 300, ctrly2: 10, x2:  
300, y2: 100,  
borderColor: 'blue',  
borderWidth: 10 )
```



**GraphicsBuilder - Outlines - Line**

Draws a line, using the current color, between the points (x1, y1) and (x2, y2).

*Properties*

Property	Default	Notes
----------	---------	-------

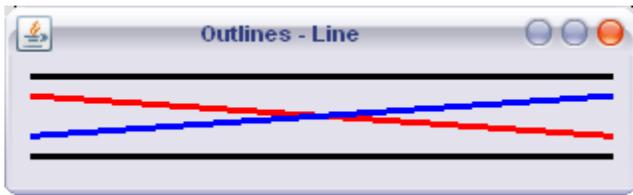
x1	0	
x2	0	
y1	0	
y2	10	

**Example**

```

line( x1: 10, y1: 10, x2: 300, y2: 10,
borderColor: 'black', borderWidth: 3 )
line( x1: 10, y1: 20, x2: 300, y2: 40,
borderColor: 'red', borderWidth: 3 )
line( x1: 10, y1: 40, x2: 300, y2: 20,
borderColor: 'blue', borderWidth: 3 )
line( x1: 10, y1: 50, x2: 300, y2: 50,
borderColor: 'black', borderWidth: 3 )

```



**GraphicsBuilder - Outlines - Polyline**

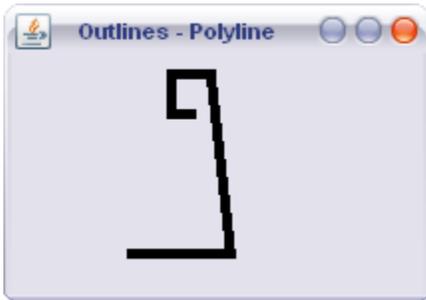
Draws a sequence of connected lines defined by arrays of x and y coordinates.

**Properties**

Property	Default	Notes
points		An even list of points

**Example**

```
polyline( points: [60, 100, 110, 100, 100, 10, 80, 10, 80, 30, 90, 30], borderColor: 'black', borderWidth: 5 )
```



GraphicsBuilder - Outlines - QuadCurve

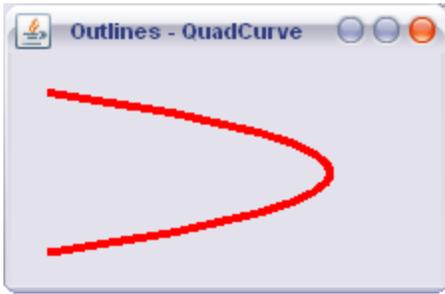
Draws a quadratic parametric curve segment.

*Properties*

Property	Default	Notes
x1		
y1		
ctrlx		
ctrly		
x2		
y2		

*Example*

```
quadCurve( x1: 20, y1: 20, ctrlx: 300, ctrly: 60, x2: 20, y2: 100, borderColor: 'red', borderWidth: 4 )
```



## GraphicsBuilder - Paints

Paints define how color patterns can be applied to a [Shape](#)'s contents either passing them as a parameter on `fill` or nesting them inside a [Shape](#) definition.

All paints support the following additional properties

- `asPaint`: creates the paint but does not apply it to the current context, useful for reusing paints in several shapes.

### Alphabetical list

[BorderPaint](#)

[ColorPaint](#)

[GradientPaint](#)

[LinearGradient](#)

[Paint](#)

[RadialGradient](#)

[TexturePaint](#)

### Extras

[MultiPaint](#)

## GraphicsBuilder - Paints - BorderPaint

Creates a paint suitable for a [shapes's](#) border only. Can not be nested in [multiPaint](#) or [paint](#).

## GraphicsBuilder - Paints - ColorPaint

Creates a color-based paint.

### Properties

Property	Default	Notes
color		
red	0	
green	0	
blue	0	
alpha	255	

## GraphicsBuilder - Paints - GradientPaint

Creates a linear gradient between 2 colors.

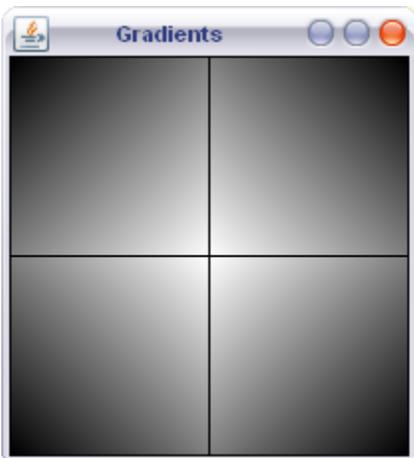
### Properties

Property	Default	Notes
x1	<b>0</b>	
y1	<b>0</b>	
x2	<b>100</b>	
y2	<b>0</b>	
color1	<b>'black'</b>	
color2	<b>'white'</b>	
cycle	<b>false</b>	
stretch	<b>false</b>	Scales the gradient to fit the Shape's bounds
fit	<b>true</b>	Scales the gradient retaining the aspect ratio

### Examples

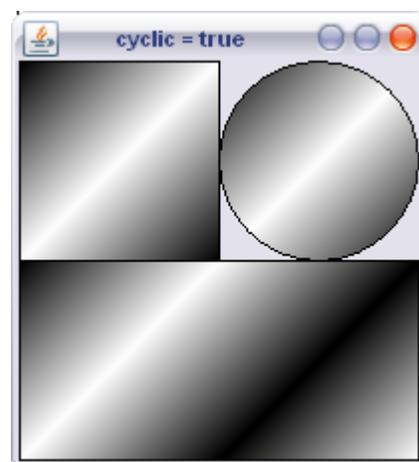
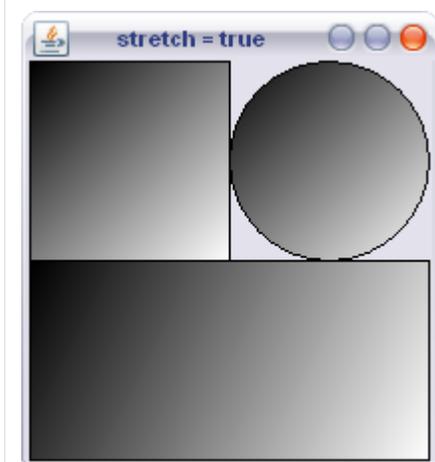
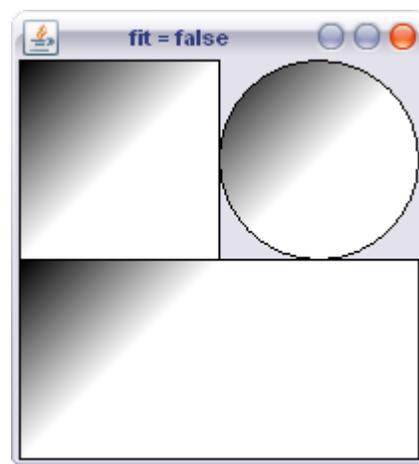
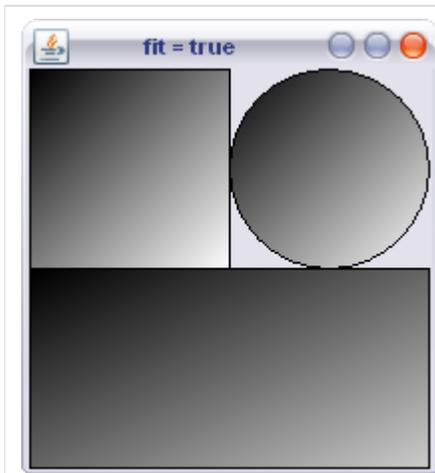
4 rectangles, each one with a diagonal gradient

```
rect( x: 0, y: 0, width: 100, height: 100 ){
  gradientPaint( x1: 0, y1: 0, x2: 50, y2:
50 )
}
rect( x: 100, y: 0, width: 100, height: 100
){
  gradientPaint( x1: 50, y1: 0, x2: 0, y2:
50 )
}
rect( x: 0, y: 100, width: 100, height: 100
){
  gradientPaint( x1: 0, y1: 50, x2: 50, y2:
0 )
}
rect( x: 100, y: 100, width: 100, height:
100 ){
  gradientPaint( x1: 50, y1: 50, x2: 0, y2:
0 )
}
```



Different settings for fit, stretch and cycle

```
gradientPaint( id: 'g1', x1: 0, y1: 0, x2:
50, y2: 50 )
rect( x: 0, y: 0, width: 100, height: 100,
fill: g1 )
circle( cx: 150, cy: 50, radius: 50, fill:
g1 )
rect( x: 0, y: 100, width: 200, height: 100,
fill: g1 )
```



### GraphicsBuilder - Paints - LinearGradient

Creates a linear gradient of many colors.

Requires graphicsbuilder-ext-jdk6 and Java6+.

### Properties

Property	Default	Notes
x1	0	
y1	0	
x2	100	
y2	0	
cycle	'nocycle'	value must be any of ['nocycle','reflect','repeat'] or any MultipleGradientPaint.CycleMethod
stretch	false	Scales the gradient to fit the Shape's bounds
fit	true	Scales the gradient retaining the aspect ratio
stops	[]	must define at least two stops
linkTo		a reference to a linearGradient or a <a href="#">radialGradient</a>

If `linkTo` is used, then this gradient will reuse the referenced one's stops, new stops may be added to this gradient though.

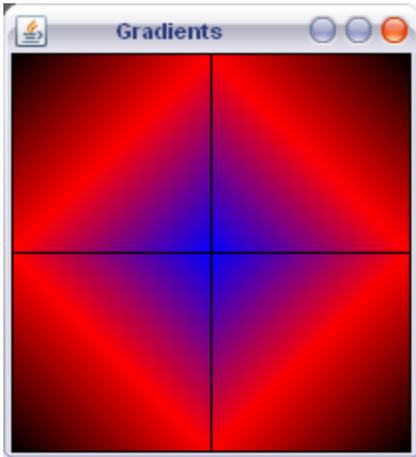
### Stops

Property	Default	Notes
offset		a float value in the range [0..1]
color		a Color or a color string
opacity		must be in the range [0..1]

### Example

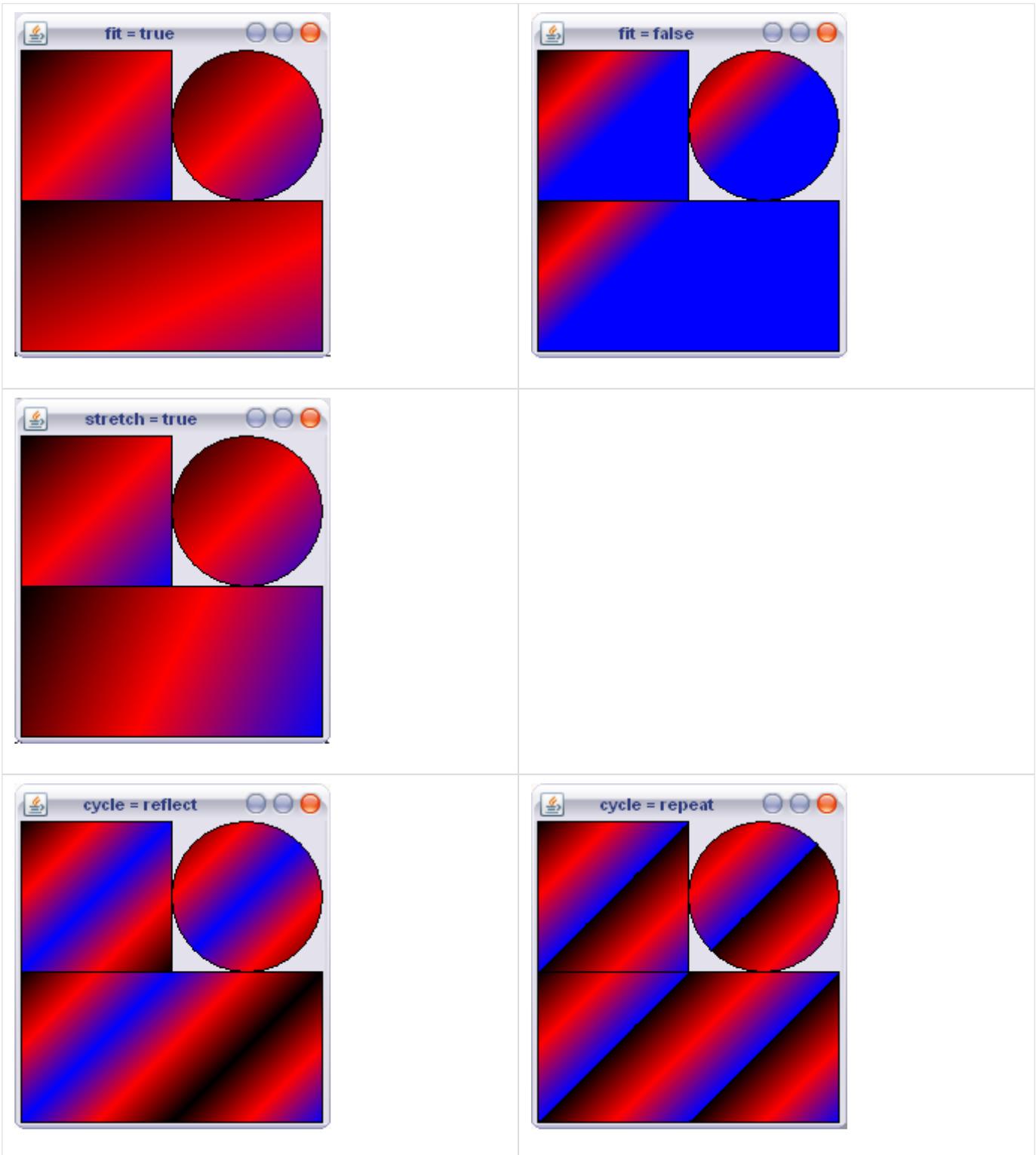
4 rectangles, each one with a diagonal gradient

```
def stops = {
  def offsets = [0,0.5,1]
  def colors = ['black','red','blue']
  (0..2).each{ stop( offset: offsets[it],
color: colors[it] ) }
}
rect( x: 0, y: 0, width: 100, height: 100 ){
  linearGradient( x1: 0, y1: 0, x2: 50, y2:
50 ){ stops() }
}
rect( x: 100, y: 0, width: 100, height: 100
){
  linearGradient( x1: 50, y1: 0, x2: 0, y2:
50 ){ stops() }
}
rect( x: 0, y: 100, width: 100, height: 100
){
  linearGradient( x1: 0, y1: 50, x2: 50,
y2: 0 ){ stops() }
}
rect( x: 100, y: 100, width: 100, height:
100 ){
  linearGradient( x1: 50, y1: 50, x2: 0,
y2: 0 ){ stops() }
}
```



Different settings for fit, stretch and cycle

```
def stops = {
  def offsets = [0,0.5,1]
  def colors = ['black','red','blue']
  (0..2).each{ stop( offset: offsets[it],
color: colors[it] ) }
}
inearGradient( x1: 0, y1: 0, x2: 50, y2: 50,
id: 'g1' ){ stops() }
rect( x: 0, y: 0, width: 100, height: 100,
fill: g1 )
circle( cx: 150, cy: 50, radius: 50, fill:
g1 )
rect( x: 0, y: 100, width: 200, height: 100,
fill: g1 )
```



### GraphicsBuilder - Paints - MultiPaint

MultiPaints allow several [paints](#) to be applied to the same [shape](#).

#### *Example*

The following example is taken from chapter 7 of [Filthy Rich Clients](#) where 3 gradients are applied to a circle in order to create the illusion of a sphere





```
        stop( offset: 0, color:
color('white').derive(alpha:0.4) )
        stop( offset: 0.5, color:
color('white').derive(alpha:0) )
```

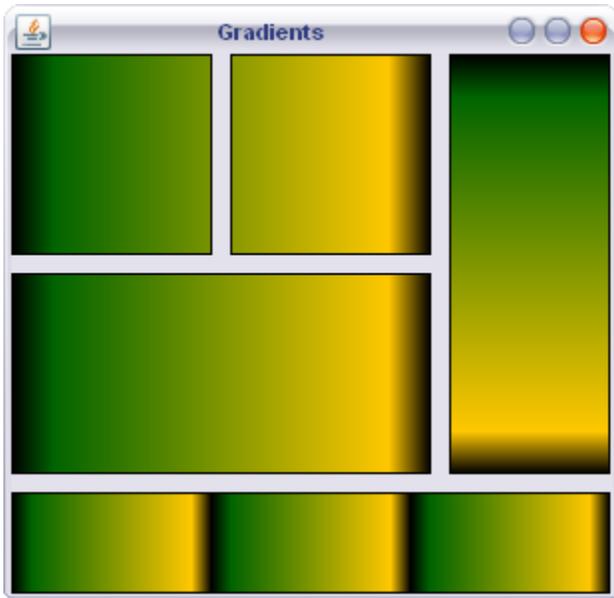


```
linearGradient( x1: 0, y1: 0, x2: 100, y2:
0, asPaint: true, id: 'g' ){
    stop( offset: 0, color: 'black' )
    stop( offset: 0.1, color: 'darkGreen' )
    stop( offset: 0.9, color: 'orange' )
    stop( offset: 1, color: 'black' )
}
rect( width: 100, height: 100, asShape:
true, id: 'r1' )
rect( x: 110, width: 100, height: 100,
asShape: true, id: 'r2' )
rect( y: 110, width: 210, height: 100,
asShape: true, id: 'r3' )

// 3 rectangles as a single shape
add( borderColor: 'black', fill: g ) {
    shape( r1 )
    shape( r2 )
    shape( r3 )
}

// vertical rectangle
rect( x: 220, width: 80, height: 210 ){
    paint( g, x2: 0, y2: 50 )
}

// horizontal rectangle
rect( y: 220, width: 300, height: 50 ){
    paint( g, cycle: 'repeat' )
}
```



### GraphicsBuilder - Paints - RadialGradient

Creates a radial gradient of many colors. It accepts [transformations](#) too.

Requires graphicsbuilder-ext-jdk6 and Java6+.

#### Properties

Property	Default	Notes
cx		
cy		
fx		will take the value of cx if not defined
fy		will take the value of cy if not defined
radius		
absolute	<b>false</b>	flag for moving the center relative to the Shape or not
cycle	<b>'nocycle'</b>	value must be any of ['nocycle', 'reflect', 'repeat'] or any MultipleGradientPaint.CycleMethod
stops	<b>[ ]</b>	must define at least two stops

linkTo		a reference to a radialGradient or a <a href="#">linearGradient</a>
--------	--	---

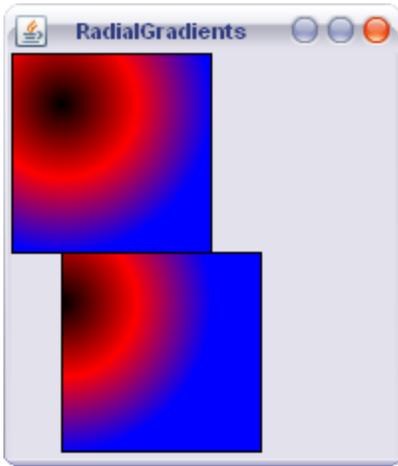
If `linkTo` is used, then this gradient will reuse the referenced one's stops, new stops may be added to this gradient though.

Stops

Property	Default	Notes
offset		a float value in the range [0..1]
color		a Color or a color string
opacity		must be in the range [0..1]

*Example*

```
def stops = {
  def offsets = [0,0.5,1]
  def colors = ['black','red','blue']
  (0..2).each{ stop( offset: offsets[it],
color: colors[it] ) }
}
rect( x: 0, y: 0, width: 100, height: 100 ){
  radialGradient( cx: 25, cy: 25, radius:
75 ){ stops() }
}
rect( x: 25, y: 100, width: 100, height: 100
){
  radialGradient( cx: 25, cy: 125, radius:
75, absolute: true ){ stops() }
}
```



### GraphicsBuilder - Paints - TexturePaint

Creates an image based paint. It is recommended that the image should have small dimensions as it will be tiled and copied to fit the Shape's bounds.

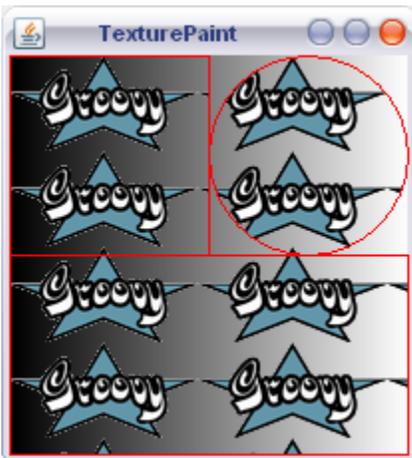
#### Properties

Property	Default	Notes
x	0	
y	0	
width		if specified will override the image's width
height		if specified will override the image's height
image		must be a java.awt.Image, an image or shape operation
classpath		String
url		URL or String
file		File or String

You must specify at least any of [image,classpath,url,file]

#### Example

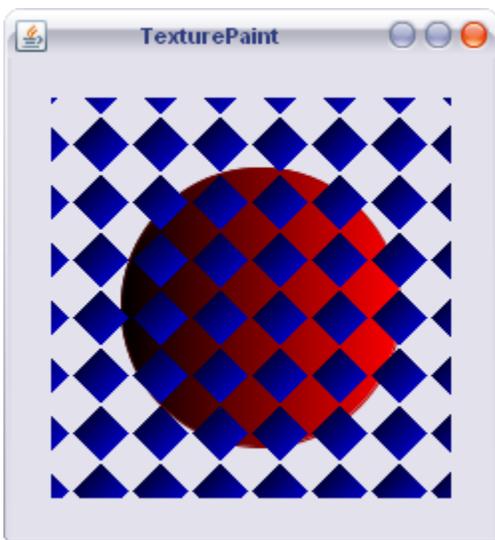
```
rect( width: 200, height: 200, borderColor:
false ){
    gradientPaint()
}
texturePaint( id: 'p', x: 0, y: 0, file:
'c:\\tmp\\groovy-logo.png' )
rect( x: 0, y: 0, width: 100, height: 100,
borderColor: 'red', fill: p, )
circle( cx: 150, cy: 50, radius: 50,
borderColor: 'red', fill: p )
rect( x: 0, y: 100, width: 200, height: 100,
borderColor: 'red', fill: p )
```



```

renderingHint( key: 'antialiasing', value:
'antialias on' )
rect( width: 20, height: 20, borderColor:
false, asShape: true, id: 'r' ){
    gradientPaint( y2: 50, color2: 'blue' )
    transformations { rotate(angle:45) }
}
circle( cx: 125, cy: 125, radius: 70,
asShape: true, id: 'c' )
draw( shape: c ){
    gradientPaint( color2: 'red' )
}
rect( x:20, y: 20, width: 200, height: 200
){
    texturePaint( image: r )
}

```



### GraphicsBuilder - Shapes

All shapes support the following additional properties

- `borderColor`: defines the color of the shape's outline. If false, the outline will not be drawn.
- `borderWidth`: defines the thickness of the shape's outline.

- fill: defines the color, [paint or gradient](#) to fill the shape's content.
- opacity: controls how much of the shape is visible, value must be in the range [0..1], default is 1.
- asShape: creates the shape but does not render to the screen, useful for mixing complex shapes.
- asImage: creates the shape but does not render to the screen, useful for drawing [images](#) or applying [textures](#)

#### Alphabetical list

[Arc](#)  
[Arrow](#)  
[Circle](#)  
[Cross](#)  
[Donut](#)  
[Ellipse](#)  
[Morph](#)  
[Path](#)  
[Pin](#)  
[Polygon](#)  
[Rays](#)  
[Rect](#)  
[RegularPolygon](#)  
[RoundRect](#)  
[Star](#)  
[Text](#)  
[Triangle](#)  
[Xpath](#)

#### Areas

GraphicsBuilder can perform 4 area operations. All area operations require at least 2 shapes, and support additional shapes in the same operation.

[Add](#)  
[Intersect](#)  
[Subtract](#)  
[Xor](#)  
[Shape](#)

#### Events

All shapes may receive event notifications if the appropriate event handler is declared. The object passed as parameter to each handler is a GraphicsInputEvent, which wraps the original swing event, it has the following properties

source	the component where the event occurred, typically a GraphicsPanel
event	the wrapped event
sourceShape	the shape where the event was triggered

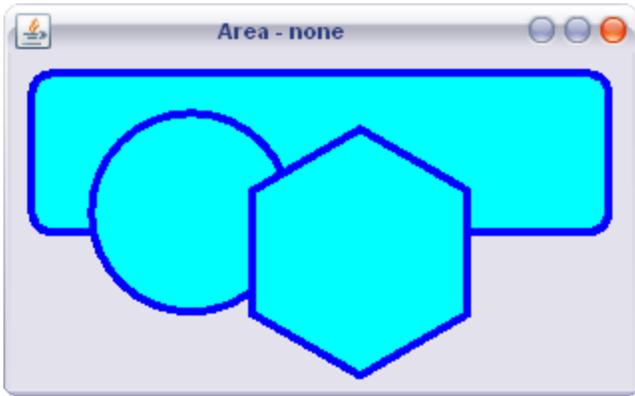
Event	Notes
-------	-------

mouseEntered	called when the mouse enters the bounds of a shape
mouseExited	called when the mouse exits the bounds of a shape
mouseClicked	called when the mouse is clicked within the bounds of a shape
mousePressed	called when the mouse is pressed within the bounds of a shape
mouseReleased	called when the mouse is released within the bounds of a shape
mouseMoved	called when the mouse is moved within the bounds of a shape
mouseDragged	called when the mouse is dragged within the bounds of a shape
mouseWheelMoved	called when the mouse's wheel is moved

### GraphicsBuilder - Area - Add

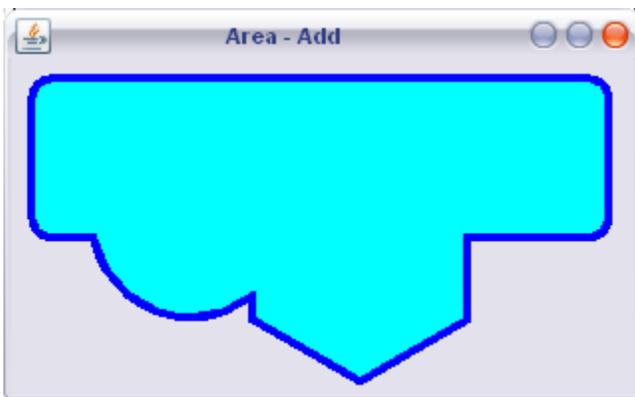
Before

```
group( borderColor: 'blue', borderWidth: 4,  
fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```



After

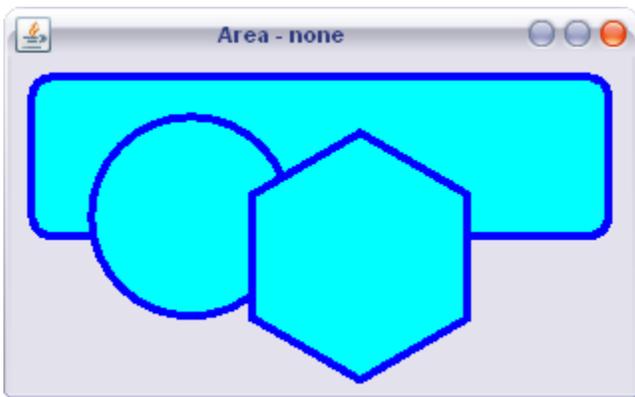
```
add( borderColor: 'blue', borderWidth: 4,  
fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```



GraphicsBuilder - Area - Intersect

Before

```
group( borderColor: 'blue', borderWidth: 4,  
fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```



After

```
intersect( borderColor: 'blue', borderWidth:  
4, fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```

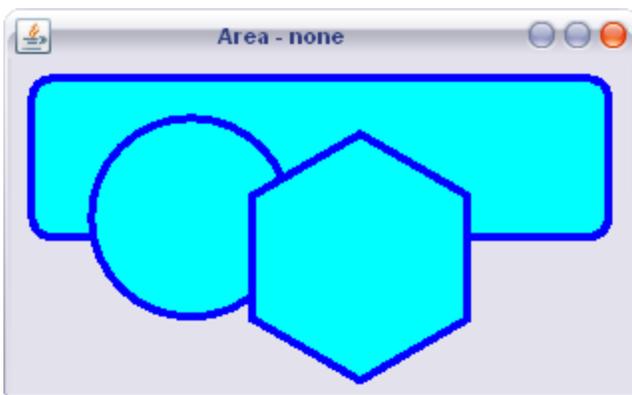


### GraphicsBuilder - Area - Shape

Area.shape is a placeholder for any Shape operation, providing reuse of previously defined shapes. The following example is the same as [Add](#) but reuses 2 previously defined shapes.

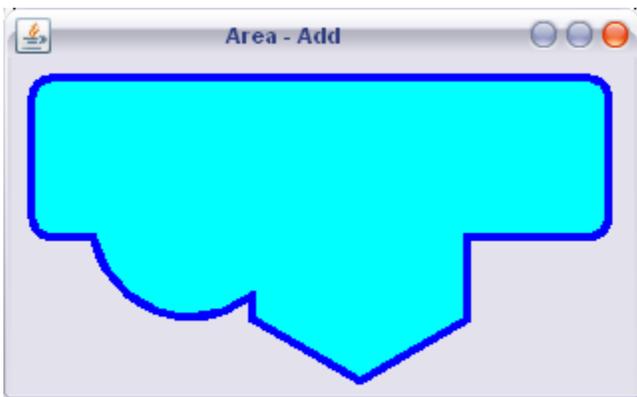
Before

```
group( borderColor: 'blue', borderWidth: 4,  
fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```



After

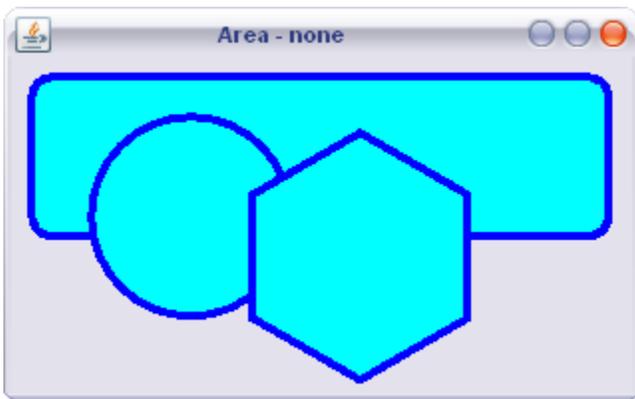
```
// note that the next two shapes have
asShape=true and have an id set
rect( x: 10, y: 10, width: 290, height: 80,
arcWidth: 20, arcHeight: 20, asShape: true,
id: 's1' )
circle( cx: 90, cy: 80, radius: 50, asShape:
true, id: 's2' )
add( borderColor: 'blue', borderWidth: 4,
fill: 'cyan' ){
    shape( s1 )
    shape( s2 )
    polygon(points: [175, 38, 229, 69, 229,
131, 175, 162, 121, 131, 121, 69])
}
```



GraphicsBuilder - Area - Subtract

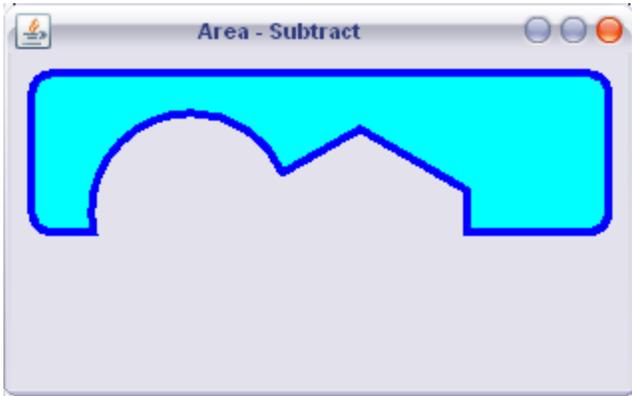
Before

```
group( borderColor: 'blue', borderWidth: 4,  
fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```



After

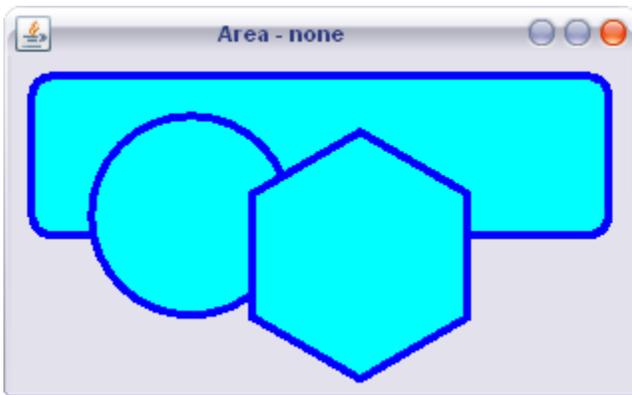
```
subtract( borderColor: 'blue', borderWidth:  
4, fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```



GraphicsBuilder - Area - Xor

Before

```
group( borderColor: 'blue', borderWidth: 4,  
fill: 'cyan' ){  
    rect( x: 10, y: 10, width: 290, height:  
80, arcWidth: 20, arcHeight: 20 )  
    circle( cx: 90, cy: 80, radius: 50 )  
    polygon(points: [175, 38, 229, 69, 229,  
131, 175, 162, 121, 131, 121, 69])  
}
```

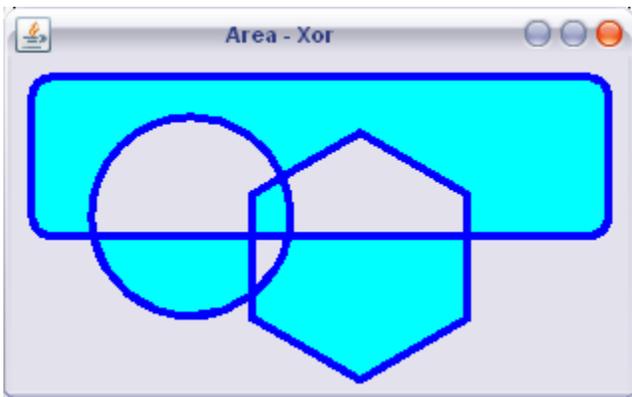


After

```

xor( borderColor: 'blue', borderWidth: 4,
fill: 'cyan' ){
    rect( x: 10, y: 10, width: 290, height:
80, arcWidth: 20, arcHeight: 20 )
    circle( cx: 90, cy: 80, radius: 50 )
    polygon(points: [175, 38, 229, 69, 229,
131, 175, 162, 121, 131, 121, 69])
}

```



### GraphicsBuilder - Shapes - Arc

Draws an arc defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE)

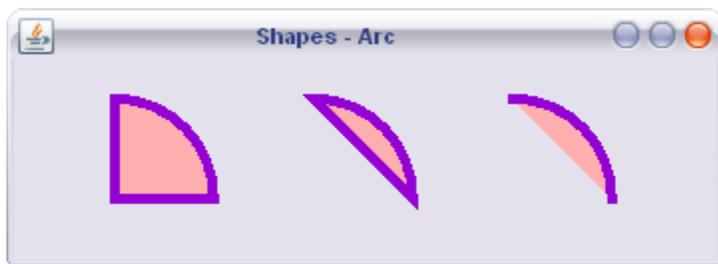
#### Properties

Property	Default	Notes
x	0	
y	0	
width	0	
height	0	
start	0	
extent	90	

close	<code>Arc2D.OPEN</code>	value must be any of <code>Arc2D.[OPEN,CHORD,CLOSE]</code> or <code>['open','chord','close']</code>
-------	-------------------------	---

**Example**

```
arc( x: 0, y: 20, width: 100, height: 100,
start: 0, extent: 90,
borderColor: 'darkViolet', borderWidth:
5, fill: 'pink', close: 'pie' )
arc( x: 100, y: 20, width: 100, height: 100,
start: 0, extent: 90,
borderColor: 'darkViolet', borderWidth:
5, fill: 'pink', close: 'chord' )
arc( x: 200, y: 20, width: 100, height: 100,
start: 0, extent: 90,
borderColor: 'darkViolet', borderWidth:
5, fill: 'pink', close: 'open' )
```



**GraphicsBuilder - Shapes - Arrow**

Draws an arrow anchored at [x,y].

**Properties**

Property	Default	Notes
x	0	
y	0	
width	10	

height	6	
depth	0.5	must be in the range [0..1]
rise	0.5	must be in the range [0..1]
angle	0	

### GraphicsBuilder - Shapes - Circle

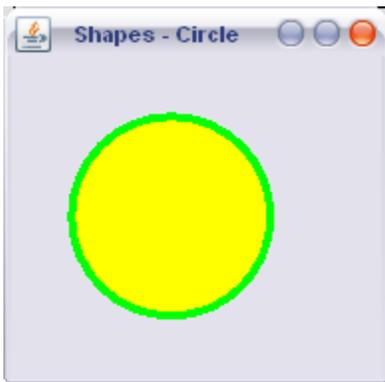
Draws a circle that is defined by a framing rectangle

#### Properties

Property	Default	Notes
cx	5	
cy	5	
radius	5	

#### Example

```
circle( cx: 80, cy: 80, radius: 50,
borderColor: 'green', borderWidth:4, fill:
'yellow' )
```



### GraphicsBuilder - Shapes - Cross

Draws a cross centered at [cx,cy].

#### Properties

Property	Default	Notes
cx	5	
cy	5	
radius	5	
width	$\text{radius} * 3 / 5$	
roundness	0	must be in the range [0..1]
angle	0	

### GraphicsBuilder - Shapes - Donut

Draws a donut-like shape centered at [cx,cy].

#### Properties

Property	Default	Notes
cx	20	
cy	20	
or	20	
ir	10	
sides		will use a regular polygon if set, a circle otherwise
angle	0	

### GraphicsBuilder - Shapes - Ellipse

Draws an ellipse that is defined by a framing rectangle.

#### Properties

Property	Default	Notes
cx	10	
cy	5	
radiusx	10	

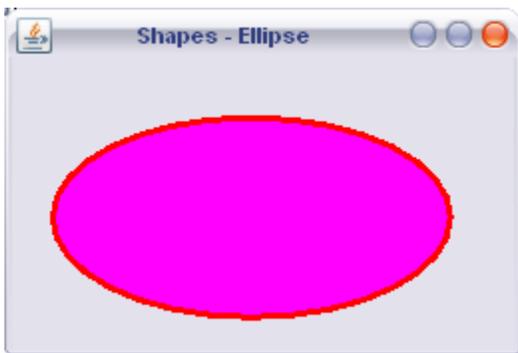
radiusy	5	
---------	---	--

**Example**

```

ellipse( cx: 120, cy: 80, radiusx: 100,
radiusy: 50, borderColor: 'red',
borderWidth: 3, fill: 'magenta' )

```



**GraphicsBuilder - Shapes - Morph**

Draws a shape which geometry is constructed from two other shapes: a start shape and an end shape. The morph property of a morphing shape defines the amount of transformation applied to the start shape to turn it into the end shape. Both shapes must have the same winding rule.

Requires graphicsbuilder-ext-swingx and swingx in classpath.

**Properties**

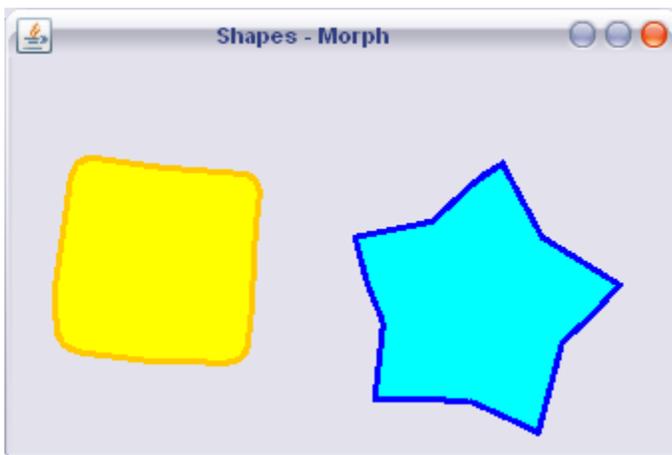
Properties	DefaultValue	Notes
start		may be a java.awt.Shape or any shape operation
end		may be a java.awt.Shape or any shape operation
morph	0	float value in the range (0..1)

**Example**

```

def rect = rect( x: 20, y: 50, width: 100,
height: 100, arcWidth: 20, arcHeight: 20,
asShape: true)
def star = star( cx: 100, cy: 100, ir: 50,
or: 80, count: 5, asShape: true )
morph( start: rect, end: star, morph: 0.1,
borderWidth: 3, borderColor: 'orange', fill:
'yellow' )
morph( start: rect, end: star, morph: 0.8,
borderWidth: 3, borderColor: 'blue', fill:
'cyan' ){
  transformations { translate(x: 140, y: 0)
}
}

```



### GraphicsBuilder - Shapes - Path

Draws an arbitrary geometric path. Paths are described by a series of pathOperations:

Operation	Description
moveTo[x,y]	adds a point to the path by moving to the specified coordinates (x,y)

<code>lineTo[x,y]</code>	adds a point to the path by drawing a straight line from the current coordinates to the new specified coordinates (x,y)
<code>curveTo[x1,y1,x2,y2,x3,y3]</code>	adds a curved segment, defined by three new points, to the path by drawing a Bézier curve that intersects both the current coordinates and the specified coordinates (x3,y3), using the specified points (x1,y1) and (x2,y2) as Bézier control points
<code>quadTo[x1,y1,x2,y2]</code>	adds a curved segment, defined by two new points, to the path by drawing a Quadratic curve that intersects both the current coordinates and the specified coordinates (x2,y2), using the specified point (x1,y1) as a quadratic parametric control point
<code>hline[x]</code>	adds a point to the path by drawing an horizontal line to the specified coordinates (x,current.y)
<code>vline[y]</code>	adds a point to the path by drawing a vertical line to the specified coordinates (current.x,y)
<code>shapeTo[shape,connect]</code>	appends the geometry of the specified Shape, shape operation or outline operation to the path, possibly connecting the new geometry to the existing path segments with a line segment
<code>close</code>	closes the current subpath by drawing a straight line back to the coordinates of the last <code>moveTo</code>

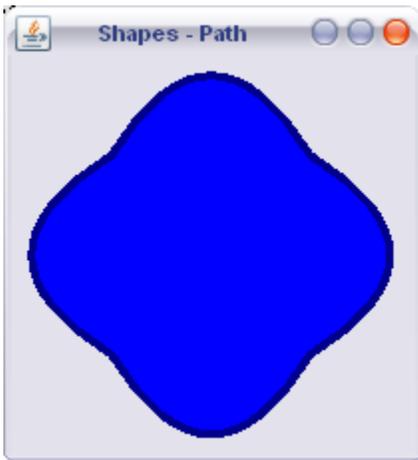
The first operation must be a `moveTo`.

**Example**

```

path( borderColor: 'darkBlue', fill: 'blue',
borderWidth: 4 ){
    moveTo( x: 50, y: 50 )
    quadTo( x1: -30, y1: 100, x2: 50, y2: 150
)
    quadTo( x1: 100, y1: 230, x2: 150, y2:
150 )
    quadTo( x1: 230, y1: 100, x2: 150, y2: 50
)
    quadTo( x1: 100, y1: -30, x2: 50, y2: 50
)
}

```



#### GraphicsBuilder - Shapes - Pin

Draws a pin (like the ones used in webmaps) centered at [cx,cy].

##### Properties

Property	Default	Notes
cx	5	
cy	5	
radius	5	

height	<code>radius * 2</code>	
angle	<code>0</code>	

### GraphicsBuilder - Shapes - Polygon

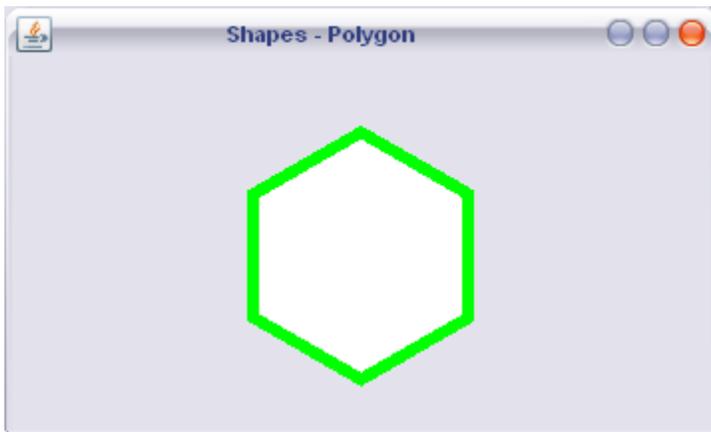
Draws a closed, two-dimensional region within a coordinate space. This region is bounded by an arbitrary number of line segments, each of which is one side of the polygon. Internally, a polygon comprises of a list of (x,y) coordinate pairs, where each pair defines a vertex of the polygon, and two successive pairs are the endpoints of a line that is a side of the polygon. The first and final pairs of (x,y) points are joined by a line segment that closes the polygon.

#### Properties

Property	Default	Notes
points		An even list of points

#### Example

```
polygon( points: [175, 38, 229, 69, 229,
131, 175, 162, 121, 131, 121, 69],
borderColor: 'green', borderWidth:
6, fill: 'white' )
```



### GraphicsBuilder - Shapes - Rays

Draws a set of rays centered at [cx,cy].

#### Properties

Property	Default	Notes
cx	<code>5</code>	

cy	5	
radius	5	
rays	2	can not be less than 2
extent	0.5	must be in the range [0..1]
rounded	false	
angle	0	

### GraphicsBuilder - Shapes - Rect

Draws a rectangle defined by a location (x,y) and dimension (width x height). If arcWidth and arcHeight are defined then it will draw a rounded rectangle.

#### Properties

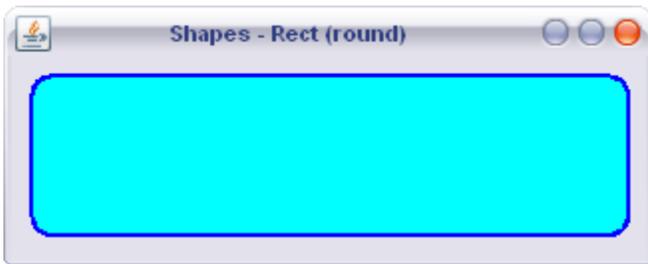
Property	Default	Notes
x	0	
y	0	
width	10	
height	10	
arcWidth		
arcHeight		

#### Examples

```
rect( x: 10, y: 10, width: 300, height: 80,
borderColor: 'darkRed', borderWidth: 2,
fill: 'red' )
```



```
rect( x: 10, y: 10, width: 300, height: 80,  
borderColor: 'blue', borderWidth: 2, fill:  
'cyan',  
    arcWidth: 20, arcHeight: 20 )
```



### GraphicsBuilder - Shapes - RegularPolygon

Draws a regular polygon centered at [cx,cy].

#### Properties

Property	Default	Notes
cx	5	
cy	5	
radius	5	
sides	3	can not be less than 3
angle	0	

### GraphicsBuilder - Shapes - RoundRect

Draws a rounded rectangle anchored at [x,y]. Each corner may have a different round setting.

#### Properties

### GraphicsBuilder - Shapes - Star

Draws a star shape. A star is defined by two radii and a number of branches. Each branch spans between the two radii. The inner radius is the distance between the center of the star and the origin of the branches. The outer radius is the distance between the center of the star and the tips of the branches.

*Properties*

Property	Default	Notes
cx	5	
cy	5	
ir	3	Can not be equal or bigger than or
or	8	Can not be equal or smaller than ir
count	5	Can not be less than 2
angle	0	

**GraphicsBuilder - Shapes - Text**

Draws a string at the specified coordinates.

*Properties*

Property	Default	Notes
text	'Groovy'	
x	0	
y	0	

*Example*

```
font( face: "Helvetica", style: Font.ITALIC,  
size: 48 )  
text( text: 'Groovy rocks!', x: 20, y: 20,  
borderColor: 'darkGreen', fill: 'green' )
```



### GraphicsBuilder - Shapes - Triangle

Draws an equilateral, isosceles or right triangle anchored at [x,y].

#### Properties

Property	Default	Notes
x	0	
y	0	
width	10	
height		will draw an isosceles or a right triangle if set
rightAngleAt		will draw a right triangle if set, value must be any of [start,end]
rotateAtCenter		will rotate the triangle at its center instead of using (x,y) as anchor
angle	0	

### GraphicsBuilder - Shapes - Xpath

Draws an arbitrary geometric path. Paths are described by a series of pathOperations. Xpath is different to [path](#) because it uses the Batik library for rendering and adds a new operation: `xArcTo`

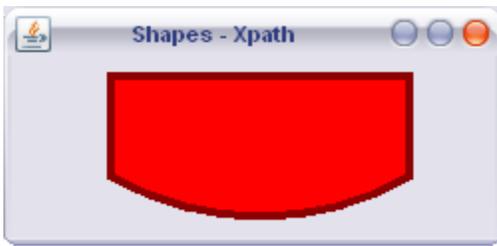
Operation	Description
<code>xmoveTo[x,y]</code>	adds a point to the path by moving to the specified coordinates (x,y)
<code>xlineTo[x,y]</code>	adds a point to the path by drawing a straight line from the current coordinates to the new specified coordinates (x,y)

<code>xcurveTo[x1,y1,x2,y2,x3,y3]</code>	adds a curved segment, defined by three new points, to the path by drawing a Bézier curve that intersects both the current coordinates and the specified coordinates (x3,y3), using the specified points (x1,y1) and (x2,y2) as Bézier control points
<code>xquadTo[x1,y1,x2,y2]</code>	adds a curved segment, defined by two new points, to the path by drawing a Quadratic curve that intersects both the current coordinates and the specified coordinates (x2,y2), using the specified point (x1,y1) as a quadratic parametric control point
<code>xhline[x]</code>	adds a point to the path by drawing an horizontal line to the specified coordinates (x,current.y)
<code>xvline[y]</code>	adds a point to the path by drawing a vertical line to the specified coordinates (current.x,y)
<code>xshapeTo[shape,connect]</code>	appends the geometry of the specified Shape, shape operation or outline operation to the path, possibly connecting the new geometry to the existing path segments with a line segment
<code>xclose</code>	closes the current subpath by drawing a straight line back to the coordinates of the last <code>moveTo</code>
<code>xarcTo[x,y,rx,ry,angle,sweep,largeArc]</code>	adds an elliptical arc, defined by two radii (rx,ry), an angle from the x-axis, a flag to choose the large arc or not, a flag to indicate if we increase or decrease the angles and the final point of the arc (x,y)

The first operation must be a `moveTo`.

*Example*

```
xpath( borderColor: 'darkRed', fill: 'red',
borderWidth: 4 ){
  xmoveTo( x: 50, y: 10 )
  xhline( x: 200 )
  xvline( y: 60 )
  xarcTo( x: 50, y: 60, rx: 150, ry: 150,
angle: 90, sweep: true, largeArc: false )
}
```



## GraphicsBuilder - Strokes

Strokes allow customization on [outlines](#) and a [shape's](#) border. Some of the provided strokes come from [Jerry Huxtable's stroke set](#)

[BasicStroke](#)

[CompositeStroke](#)

[CompoundStroke](#)

[Shapetroke](#)

[Stroke](#)

[TextStroke](#)

[WobbleStroke](#)

[ZigzagStroke](#)

### GraphicsBuilder - Strokes - BasicStroke

Creates a java.awt.BasicStroke used to draw the contour of [shapes](#) and [outlines](#).

#### Properties

Property	Default	Notes
width	1	
cap	'square'	any of [butt,round,square]
join	'miter'	any of [bevel,round,miter]
miterlimit	10	
dash		
dashphase	0	

### GraphicsBuilder - Strokes - CompositeStroke

Creates a stroke as a composite of two strokes, meaning that the second stroke will be used to draw the outline created by the first stroke.

#### Properties

Property	Default	Notes
----------	---------	-------

stroke1		must be a java.awt.Stroke or a <a href="#">StrokeProvider</a>
stroke2		must be a java.awt.Stroke or a <a href="#">StrokeProvider</a>

This operation accepts nested [strokes](#), they will be applied in appearing order. If more than 2 strokes are nested only the first 2 will be taken into account.

### GraphicsBuilder - Strokes - CompoundStroke

Creates a stroke as a composite of two strokes using an Area operation.

#### Properties

Property	Default	Notes
stroke1		must be a java.awt.Stroke or a <a href="#">StrokeProvider</a>
stroke2		must be a java.awt.Stroke or a <a href="#">StrokeProvider</a>
operation	'add'	must be any of [add,subtract,intersect,xor]

This operation accepts nested [strokes](#), they will be applied in appearing order. If more than 2 strokes are nested only the first 2 will be taken into account.

### GraphicsBuilder - Strokes - Shapetroke

Creates a stroke that uses [shapes](#) to define the outline.

#### Properties

Property	Default	Notes
shapes	[ ]	a list of java.awt.Shape or <a href="#">shapes</a>
advance	10	

This operation accepts nested [shapes](#) and [shape](#).

### GraphicsBuilder - Strokes - Stroke

Placeholder for previously defined strokes, works in the same way as [paint](#) does for [paints](#).

#### Properties

Property	Default	Notes
----------	---------	-------

stroke		must be a java.awt.Stroke or a <a href="#">StrokeProvider</a>
--------	--	---

### GraphicsBuilder - Strokes - TextStroke

Creates a stroke that draws a text along the stroked outline.

#### Properties

Property	Default	Notes
text	'Groovy'	
font		
stretch	false	
repeat	false	

### GraphicsBuilder - Strokes - WobbleStroke

Creates a randomly generated outline with variable width.

#### Properties

Property	Default	Notes
detail	2	
amplitude	2	

### GraphicsBuilder - Strokes - ZigzagStroke

Creates a zig-zag like pattern for stroking an outline.

#### Properties

Property	Default	Notes
stroke		must be a java.awt.Stroke or a <a href="#">StrokeProvider</a>
amplitude	10	
wavelength	10	

This operation accepts nested [strokes](#), they will be applied in appearing order. If more than 1 strokes are nested only the first will be taken into account.

### Graphicsbuilder - SVG

Pending

## GraphicsBuilder - Swing

Swing components can be embedded using the `swingView` node. It will use a fresh copy of `SwingBuilder`, but you can specify your own [SwingBuilder](#) instance or some other builder like [SwingXBuilder](#) or [JideBuilder](#).

Example

```
swingView {
  panel( x: 10, y: 10 ) {
    GridLayout( cols: 1, rows: 4 )
    ['red', 'blue', 'green'].each { color ->
      button( label: color, foreground:
color,
              actionPerformed: {e->
p1.color2 = color} )
    }
    button( label: 'reset',
actionPerformed: {e-> p1.color2 = 'white'} )
  }
}

rect( x: 90, y: 10, width: 100, height: 100
){
  gradientPaint( id: 'p1', y2: 50 )
}

line( x1: 10, x2: 190, y1: 125, y2: 125 ){
  stroke( width: 2, dash: [12,12],
dashphase: 0 )
}

swingView {
```

```
    panel( id: 'panel', x: 90, y: 140,  
preferredSize: [100,100],  
          opaque: true, background: 'white'  
)  
}
```

```
def changeBackground = { shape ->  
    panel.background =  
ColorCache.getInstance().getColor(shape.fill  
) }
```

```
def borderOn = { e ->  
e.sourceShape.borderColor = 'black' }
```

```
def borderOff = { e ->  
e.sourceShape.borderColor = false }  
rect( x: 10, y: 140, width: 60, height: 20,  
borderColor: false, fill: 'red',  
      mouseClicked: {e->  
changeBackground(e.sourceShape)},  
      mouseEntered: borderOn, mouseExited:  
borderOff )
```

```
rect( x: 10, y: 165, width: 60, height: 20,  
borderColor: false, fill: 'blue',  
      mouseClicked: {e->  
changeBackground(e.sourceShape)},  
      mouseEntered: borderOn, mouseExited:  
borderOff )
```

```
rect( x: 10, y: 190, width: 60, height: 20,  
borderColor: false, fill: 'green',  
      mouseClicked: {e->  
changeBackground(e.sourceShape)},  
      mouseEntered: borderOn, mouseExited:
```

```
borderOff )
rect( x: 10, y: 215, width: 60, height: 20,
borderColor: false, fill: 'white',
      mouseClicked: {e->
```

```
changeBackground(e.sourceShape)},  
    mouseEntered: borderOn, mouseExited:  
borderOff )
```

## GraphicsBuilder - Transformations

Transformations can be applied to [Shapes](#), [Outlines](#), [groups](#), [images](#) and [radialGradients](#). All operations can be combined in a single `transformations` block if needed.

When applied to [images](#), there is an extra property `interpolation` useful for giving hints, its value may be any of [bicubic,bilinear,nearest].

### Available operations

[Matrix](#)

[Rotate](#)

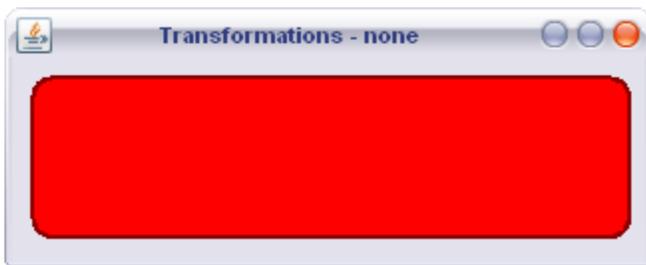
[Scale](#)

[Skew](#)

[Translate](#)

None

```
rect( x: 10, y: 10, width: 300, height: 80,  
arcWidth: 20, arcHeight: 20,  
    borderColor: 'darkRed', borderWidth:  
2, fill: 'red' )
```



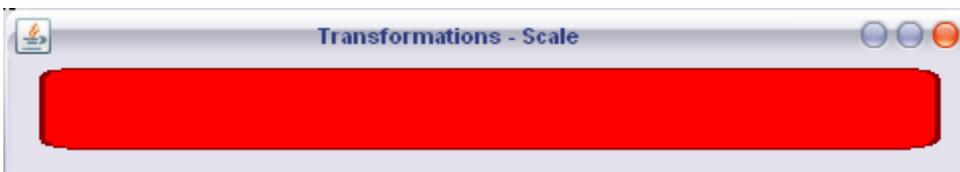
Translate

```
rect( x: 10, y: 10, width: 300, height: 80,  
      arcWidth: 20, arcHeight: 20,  
      borderColor: 'darkRed', borderWidth:  
2, fill: 'red' ){  
  transformations {  
    translate( x: 50, y: 20 )  
  }  
}
```



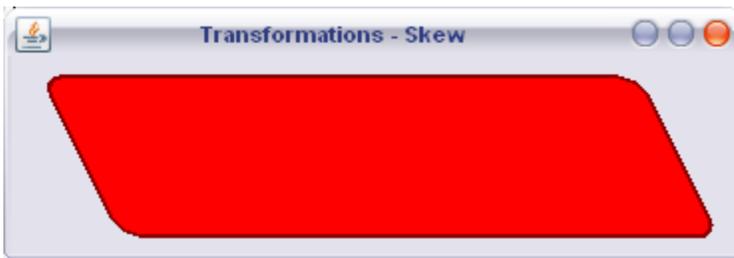
Scale

```
rect( x: 10, y: 10, width: 300, height: 80,  
      arcWidth: 20, arcHeight: 20,  
      borderColor: 'darkRed', borderWidth:  
2, fill: 'red' ){  
  transformations {  
    scale( x: 1.5, y: 0.5 )  
  }  
}
```



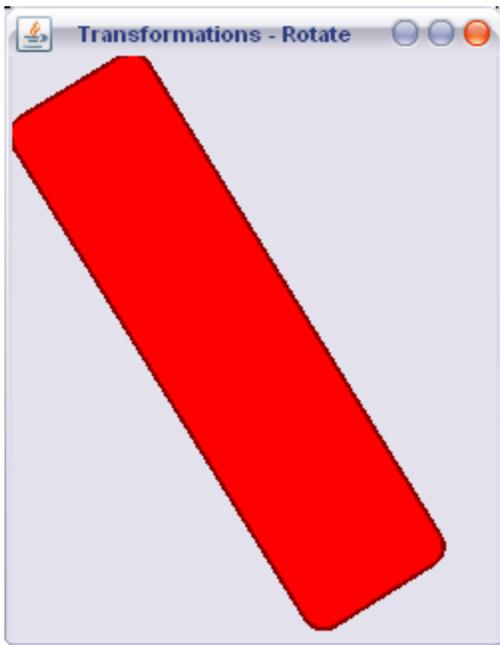
## Skew

```
rect( x: 10, y: 10, width: 300, height: 80,  
      arcWidth: 20, arcHeight: 20,  
      borderColor: 'darkRed', borderWidth:  
2, fill: 'red' ){  
  transformations {  
    skew( x: 0.5, y:0 )  
  }  
}
```



## Rotate

```
rect( x: 10, y: 10, width: 300, height: 80,  
      arcWidth: 20, arcHeight: 20,  
      borderColor: 'darkRed', borderWidth:  
2, fill: 'red' ){  
  transformations {  
    rotate( angle: 45, x: 50, y: 50 )  
  }  
}
```



## GraphicsBuilder - Tutorials

You may find a multi-part tutorial at [groovy.dzone.com](http://groovy.dzone.com), links follow

- [Shapes](#)
- [Outlines & Shapes](#)
- [Paints & Colors](#)
- [Strokes](#)

Some other useful links

- [Java2D the Groovy way](#)
- <http://jrroller.com/aalmiray/tags/graphicsbuilder>

## Grapplet

### Contribution Overview

Grapplet provides a way for running Groovy on an applet, adding extra functionality to JS objects and arrays, for example arrays behave like Java Lists, so all GDK methods available to List and Collection can be used in JS arrays.

Grapplet will automatically look for all <script> tags available in the page that have its language property set to "text/x-groovy". This was inspired by a post on Dion's blog: [Running Ruby in the browser via script type="text/ruby"](#).

Once Grapplet is running on a page, you can evaluate any Groovy script by calling evaluateScript().

### *Team Members*

Andres Almiray [aalmiray at users dot sourceforge dot net]

### Download

### *Distributions*

Pending.

## ***Installing***

Pending.

## ***Pre-requisites***

None

## **Documentation**

In order to run Grapplet it needs to be signed, follow the next instructions to use a self-signed certificate (recommended for testing)

In order to run Groovy on a browser you'll need to sign the applet. Follow the steps to sign an applet with your own certificate.

1. Create a keystore which will hold the certificate.

I created an external keystore so I wouldn't mess up my personal security settings while finding out the correct way to do it. All you have to do is issue the following command:

```
keytool -genkey -keystore groovy -storepass groovy -keypass groovy \  
-alias groovy
```

2. Trust your own certificate.

Unless you want to spend some bucks on this experiment I recommend you selfcert your certificate. To selfcert your newly created certificate, issue the following command:

```
keytool -selfcert -keystore groovy -storepass groovy -keypass groovy \  
-alias groovy
```

3. Export your certificate. Export your certificate to an external file with the following command:

```
keytool -export -keystore groovy -storepass groovy -keypass groovy \  
-alias groovy -file groovy.cer
```

4. Sign the jar. This will attach the certificate to the jar and add entries to the jar's manifest.

```
jarsigner -keystore groovy -storepass groovy -keypass groovy \  
grapplet-0.1.jar groovy
```

5. Verify your jar (just in case). You may verify that your jar has indeed been signed and includes the certificate, for more information on jarsigner's output refer to the command's help (jarsigner -help):

```
jarsigner -verify -verbose -certs -keystore groovy grapplet-0.1.jar
```

6. Configure your local security settings. For this step you must touch \$JRE\_HOME/lib/security/java.policy and \$JRE\_HOME/lib/security/java.security, in windows \$JRE\_HOME usually points to "c:/Program Files/Java/jdk1.x.x/".

1. Add the following lines at the end of java.policy:

```
grant {
  java.lang.RuntimePermission "usePolicy";
};
```

2. Create a file named '.java.policy' at \$USER\_HOME with the following contents:

```
keystore "file:${user.home}/groovy";
grant signedBy "groovy" {
  permission java.security.AllPermission;
};
grant codeBase "http://localhost" {
  permission java.security.AllPermission;
};
```

3. Copy the keystore 'groovy' and 'groovy.cer' (just in case) to \$USER\_HOME.

7. Copy the binary dist to your webserver. If you're using Apache copy grapplet.html, groovy.js, grapplet-0.1.jar, groovy-all-1.0.jar, groovy and groovy.cer to \$APACHE\_HOME/htdocs/grapplet

## Developers

### Source Control

<http://svn.codehaus.org/groovy-contrib/grapplet>

### Building

Grapplet uses Maven2 as its build tool, which means that if you want to build your own version of Grapplet from source you'll need to have it installed. Follow the instructions at <http://maven.apache.org> Once Maven2 is installed you will also need to install the java-plugin into your maven repository (but it wouldn't hurt to check at <http://mvnrepository.org> if it is already there). Usually the plugin is located at \$JDK\_HOME/jre/lib/plugin.jar

You can install it on your local Maven2 repo with the following command

```
mvn install:installFile -DgroupId=com.sun.java-plugin -Dversion=<jdkversion> \
  -Dpackaging=jar -DartifactId=java-plugin \
  -Dfile=$JDK_HOME/jre/lib/plugin.jar
```

where <jdkversion> is the version number of the selected jdk. Grapplet has version 1.6.0 configured, if you change version you'll have to update pom.xml

After you have the required dependencies installed, you may generate the package by typing

```
mvn package
```

Now you'll have to sign grapplet-<version>.jar, copy it and groovy.js to your webapp or webserver dir. The file src/html/grapplet.html should give you some pointers in how it should be configured.

The next section will describe the process of self-signing the jar.

### Contributing

Please contact the team members by e-mail.

## Community

### *Mailing List(s)*

<http://groovy.codehaus.org/Mailing+Lists>

### *Issue tracker*

<http://jira.codehaus.org/secure/BrowseProject.jspx?id=10242>

## Griffon

Griffon is now a top level Codehaus project at <http://griffon.codehaus.org>

### Where do I Start ?

- [Installing Griffon](#)
- [Griffon Quick Start](#)

### Why Griffon?

- Liger was taken, [apparently](#).
- Griffons have large talons.
- Excellent skills in [jump and spot](#) as well.
- It's a mashup, of an Eagle and a Cougar  
(since all of the Griffon committers are in North America right now, we are referring to the New World animals, instead of the traditional Eagle and Lion. We could have also gone with the Puma, Mountain Lion, or Panther instead, but I thought Cougars were the most majestic of the North American big cats.)

### What's in Griffon?

A [Roadmap](#) has been drawn up.

### New Feature Development

Names subject to change

- A desktop variant of the Grails/Rails project structure could be supported, with deployment to Applets, WebStart, or vanilla client Jar files.
- **UberBuilder** - a compositing FactoryBuilderSupport class, which allows seamless merging of existing Builders
- **GUIBuilder** - An UberBuilder with SwingBuilder, SwingXBuilder, and GraphicsBuilder pre-loaded.

### Groovy Core R&D for Desktop stuff

Quicker cycling and more risky experimental changes to some core Groovy classes will be done here as well

- **FactoryBuilderSupport** - needed changes for UberBuilder to work
- **SwingBuilder** - needed changes for UberBuilder to work

## Future Directions

There are lots of things Griffon could move into, for example

- A home for the multiple GUI builders for Groovy, It may be worthwhile to merge SwingXBuilder, JIDEBuilder, and GraphicsBuilder under a single code base (still with separate releases).
- GORM integration from Grails 1.1
- JavaFX Wrappers for the GUI Builders

## Development stuff

- [Source repository](#)
- A sample of Griffon in Action: [Greet](#), [Source code](#)

## Download Griffon

Griffon is now a standalone project. To download griffon go to the download page on the [Griffon Project Page](http://griffon.codehaus.org/Download) at <http://griffon.codehaus.org/Download>.

## Greet

Greet is one of the marquee demo applicaitons of the Griffon Framework. It is a twitter client.

[Java WebStart Link](#)

## Griffon JavaOne 2008 Dev Notes

Griffon - next steps

arch Q's

- Which variables to include in the view builds, and how to include them.

Need something like site-mesh

- a templating and naming convention

Config

- look and feel
- data layers
- preferred place for @Grab

Compisiting builder

- standrd pattterns so that we don't bring in

Standard Directory Layout

- Piggyback from Grails
- MVC separation

Assembly

- deploy as JNLPS?
- deply as big fat JAR?

plugins? How would they look in Griffon?

- like adding flamingo, SwingLabs compontnts

- auto register builder info, such as prefixes
- SpringSecurity like item, allows components to be added/deleted/enabled based on user profiles
- types
- mapping plugin
- reporting plugin
- drag and drop
  - etc.

Comparisons to JMatter?

- groffon should allow any application

## Griffon Quick Start

### Quick Start

The following makes it simple to start a griffon project.

- [Quick Start](#)
  - [Create a Griffon project](#)
  - [Create an Application Model](#)
  - [Create the Controller Logic](#)
  - [Add Content to the View](#)
  - [Run the Application](#)
- [What's Next](#)

### *Create a Griffon project*

Once you have [installed](#) Griffon you can use the built-in target for creating new projects:

```
griffon create-app
```

The target will prompt you for the name of your project and create the project structure below:

```

%PROJECT_HOME%
  + griffon-app
    + conf          ----> location
of configuration artifacts like builder
configuration
    + keys          ---->
keys for code signing
    + webstart     ---->
webstart and applet config
    + controllers  ----> location
of controller classes
    + i18n         ----> location
of message bundles for i18n
    + lifecycle    ----> location
of lifecycle scripts
    + models       ----> location
of model classes
    + resources    ----> location
of non code resources (images, etc)
    + views        ----> location
of view classes
    + lib
    + scripts      ----> scripts
    + src
      + main       ----> optional;
location for Groovy and Java source files
                                (of types
other than those in griffon-app/*)

```

Make sure you are in the root directory of your project (for argument sake "DemoConsole", a simple script evaluator) by typing

```
cd DemoConsole
```

The "create-app" target created a Griffon MVC Triad for you in the models, views, and controllers directory named after the application. hence you already have a model class DemoConsoleModel in the models directory.

The application model for the quick start is simple: the script to be evaluated and the results of the evaluation.

## **DemoConsoleModel.groovy**

```
import groovy.beans.Bindable  
  
class DemoConsoleModel {  
  
    String scriptSource  
    @Bindable def scriptResult  
    @Bindable boolean enabled = true  
  
}
```

### ***Create the Controller Logic***

The controller for our quick start app is simple: throw the contents of the script from the model at a groovy shell.

## DemoConsoleController.groovy

```
import java.awt.event.ActionEvent

class DemoConsoleController {

    GroovyShell shell = new GroovyShell()

    // these will be injected by Griffon
    def model
    def view

    def executeScript(ActionEvent evt = null)
    {
        model.enabled = false
        doOutside {
            def result
            try {
                result =
shell.evaluate(model.scriptSource)
            } finally {
                edt {
                    model.enabled = true
                    model.scriptResult = result
                }
            }
        }
    }
}
```

The Griffon framework will inject references to the other portions of the MVC triad if fields named `model`, `view`, and `controller` are present in the model or controller. This allows us to access the view widgets and the model data if needed

The `executeScript` method will be used in the view for the button action. Hence the `ActionEvent` parameter, and the default value so it can be called without an action event.

Finally, the Griffon framework can be configured to inject portions of the builders it uses. By default, the Threading classes are injected into the controller, allowing the use of the `edt`, `doOutside` and `doLater` methods from the `SwingBuilder`.

Also, the threading may look a bit obsessive. But good thread management is essential to a well functioning Swing application.

### ***Add Content to the View***

The view classes contain the visual components for your application.

## DemoConsoleView.groovy

```
application(title: 'DemoConsole', pack:true,
locationByPlatform:true) {
    panel(border:emptyBorder(6)) {
        BorderLayout()

        scrollPane(constraints:CENTER) {
            textArea(text:bind(target:model,
targetProperty:'scriptSource'),
                enabled: bind {model.enabled},
                columns:40, rows:10)
        }

        hbox(constraints:SOUTH) {
            button("Execute",
actionPerformed:controller.&executeScript,
                enabled: bind {model.enabled})
            hstrut(5)
            label("Result:")
            hstrut(5)
            label(text:bind {model.scriptResult})
        }
    }
}
```

The view script is a fairly straightforward SwingBuilder script. Griffon will execute these groovy scripts in context of it's UberBuilder (a composite of the SwingBuilder and whatever else is thrown in).

### *Run the Application*

To start your Griffon app run the following target

## griffon run-app

This will run the application as a Java application. You can also use the `run-webstart` target to run the application from a WebStart/JNLP file.

The `run-app` script implies the execution of the `package` script. The `package` script creates file artifacts suitable for a Java application, a WebStart application, and an Applet, with code signed by a self-signed certificate. All from the same source tree. By default they go in the 'target' directory.

## ls target

Try out the applet by bringing up the `applet.html` file in a browser.

### What's Next

This is just the first pass at the framework. The sky's the limit!

### Griffon RoadMap

#### 0.1 Release: First cracks in the Egg

- Grails style build tree
- Initial CompositeBuilderSupport refactoring  
Align any FactoryBuilderSupport refactored features with Groovy 1.7
- JNLP packaging
- Two demo apps, Greet and an enterprisey style app.

#### Sometime before 1.0:

- Plugin architecture like Grails
- Standard plugin
  - GORM support
  - Spring support
  - Authentication
  - Other GUI toolkits like JIDE, SwingX, Flamingo
- Standalone deployment
- Pre 6u10 Applet deployment
- SWT Builder?
- A third demo app, fronting a Grails app
- Widget tags, kinda like taglibs. [MarkMail Archive](#)

### Installing Griffon

#### *Installation from Download*

#### *Prerequisites*

In order to start using Griffon you need to have the following:

- An installation of Java SDK 1.5 or higher and have set your JAVA\_HOME variable to the SDK install location. Java 1.6 is strongly recommended.

#### Steps

1. [Download](#) the latest Griffon release
2. Extract the archive into an appropriate location; typically C:\griffon on Windows or ~/griffon on Unix
3. Create a GRIFFON\_HOME environment variable that points to the path where you extracted the archive (eg C:\griffon on Windows or ~/griffon on Unix)
4. If you have not set the JAVA\_HOME environment variable yet, create JAVA\_HOME environment variable that points to the path where you have installed Java
5. Append a reference to the "bin" directory within the Griffon directory to your PATH variable (eg %GRIFFON\_HOME%\bin on Windows or \$GRIFFON\_HOME/bin on Unix). Note that, for Windows, both PATH and GRIFFON\_HOME must be defined at the same environment variable level (eg. 'System variables') rather than across environment variable levels (eg. PATH under 'System variables' and GRIFFON\_HOME under 'User variables')
6. Type "griffon" at the command line, if a help message is displayed you are ready to [start using Griffon!](#)
7. If you get an error message, try to chmod +x the griffon script inside the bin directory.

#### Installation from Subversion

##### Prerequisites

In order to start using Griffon from SVN you need to have the following:

- An installation of Java 1.5 or higher and have set your JAVA\_HOME variable to the install location
- A working installation of [Apache Ant](#) 1.6.5 or higher and have set your ANT\_HOME variable to the install location
- JUnit (to run the build tests - put junit.jar in your ANT\_HOME\lib directory)
- An installation of [Subversion](#)

##### Steps

1. Checkout Griffon from the Griffon Module in the [Griffon SVN repository](#)
  - a. Go into an empty directory where you will keep your griffon sources, create one if you want
  - b. Run this command: `svn co http://svn.codehaus.org/griffon/core/trunk griffon-src`
2. A subdirectory named griffon-src will be created, let's call that location GRIFFON\_SRC
3. Go into GRIFFON\_SRC
4. Make sure you can run ant : `ant -version`
  - a. If you have an error, try using the ant provided in the sources (you may need to give the ant script executable permission before): `./ant/bin/ant -version`
5. Decide of a directory where you want griffon to be installed; typically C:\griffon on Windows or ~/griffon on Unix
6. Set the GRIFFON\_HOME environment variable to that location (make sure it is not below the checkout location)
7. Install griffon by running "ant install"
8. Once the build runs with success, follow the instructions as if you've downloaded a binary distribution (see "Installation from Download" above).
9. That's it! [Start developing with Griffon!](#)

## UberBuilder

### Hierarchy

- Object
  - GroovyObjectSupport
    - Binding

- FactoryBuilderSupport
  - griffon.builder.UberBuilder
    - griffon.gui.GUIBuilder

The logic to support all of the non-specific builder magic will go into UberBuilder. GUIBuilder will be an instance of UberBuilder with specific factories pre-loaded into the cache.

### Constructor

The only constructor will be

```
UberBuilder(Object[] builders) {  
    builders.each {if (it) uberInit(it)}  
}
```

This is driven by the use case. Each uberInit method will exploit the dynamic dispatch mechanism of Groovy to match to the proper uberInit method.

Consider the following calls

```
UberBuilder('ant', 'swing', 'DOM')  
UberBuilder(AntBuilder, SwingBuilder,  
DOMBuilder)  
UberBuilder(new AntBuilder(), new  
SwingBuilder(), new DOMBuilder())
```

All three of these would create an UberBuilder with ant, swing, and dom factories pre-loaded. Other factories may be addable if the child instance exposes the factory registration methods publicly. (some UberBuilders may not want to).

First, there will be an internal registry mapping, mapping some object (usually strings) to the relevant registrations for the UberBuilder. Hence the fallback case:

```

public final uberInit(Object builderKey) {
    def builder =
buildersRegistry[builderKey]
    // make sure we won't self-loop
    if
(builder?.metaClass?.respondsTo(builder,
'uberInit', builder.class)?.size() > 1) {
        // if we get more than one, we have
more than this base case, so look it up
        return uberInit(buidler)
    }
}

```

We basically try to init again if we don't get a match, except we look at the internal registry cache to 'de-reference' the symbol. Usually a String, but there may be instances where we may want to intercept a class. (should we move the dereferencing to the constructor?)

#### String

This is always a de-reference. We may not even need an uberInit method, except for clarity.

#### Class

If the class is assignable to FactoryBuilderSupport, we attempt to no-args construct it. If we are successful we feed it to uberInit(FactoryBuilderSupport), if not we feed it to uberInit(Object) before failing.

#### FactoryBuilderSupport

There are two approaches we can use here, not sure which is best.

1. We can wrap the builder's nodes using withBuilder and re-direct them into the proxy
2. We can take all of the registered factories and stuff them into this UberBuilder.

#### Map

Maps will be handled based on the type of the value argument. We may even want to allow 'unwrapped' calls via type tricks, ie

```
uberBuilder(J: SwingBuilder, SwingXBuilder,  
Ant:AntBuilder, '<':DOMBuilder)
```

#### *Factory value*

The key will be registered as a node name and the factory will be the value

#### *FactoryBuilderSupport*

The key will be a prefix to each of the nodes and it will be otherwise treated as a call with just the FactoryBuilderSupport.

## **Groosh**

Provides a shell-like capability for handling external processes.

### **Module Overview**

Groosh is a Unix like shell written in Groovy. It has also been known as the Process Module but was renamed (back) when it was updated to work with Groovy 1.0.

The current version of Groosh is 0.3.6

#### *Team Members*

- Yuri Schimke - Founder and original contributor
- Alexander Egger [alexander.egger at gmail.com] - Current maintainer

## **Download**

#### *Distributions*

Source code and binary releases are available at <http://kenai.com/projects/groovy-groosh>.

Groosh can be used via Grapes.

Note: If Ivy (Grapes) can not download Groosh add the file

```
grapeConfig.xml
```

to your

```
~/ .groovy
```

directory with the following content:

```

<ivysettings>
  <property name="ivy.checksums" value="" />
  <settings
defaultResolver="downloadGrapes" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/ivy-[revision].xml" />
          <artifact
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/[type]s/[artifact]-[revisio
n].[ext]" />
            </filesystem>
            <!-- todo add 'endorsed groovy
extensions' resolver here -->
            <ibiblio name="codehaus"
root="http://repository.codehaus.org/"
m2compatible="true" />
            <ibiblio name="ibiblio"
m2compatible="true" />
            <ibiblio name="java.net2"
root="http://download.java.net/maven/2/"
m2compatible="true" />
          </chain>
        </resolvers>
      </ivysettings>

```

There seems to be some problem with checksums generated by Maven 2.2.1

### ***Installing***

No installation required. Groosh can be added to Groovy using Grapes.

Try the following script to check if it worked:

```
@Grapes ( [  
  @Grab (group= 'org.codehaus.groovy.modules' ,m  
odule= 'groosh' ,version= ' [0.3.6, ) ' ) ,  
  @GrabConfig (systemClassLoader= true)  
])  
  
import groosh.Groosh  
Groosh.withGroosh (this)  
  
ls () >> stdout
```

### Pre-requisites

Groosh is based on Groovy 1.7.0 or higher.

### Documentation

The following example shows Groosh in action:

```
//Read a text file and write it to stdout
@Grapes([
  @Grab(group='org.codehaus.groovy.modules',module='groosh',version=' [0.3.6,)' ),
  @GrabConfig(systemClassLoader=true)
])

import groosh.Groosh
Groosh.withGroosh(this)

cat('test_scripts/blah.txt') >> stdout
```

Another example :

```

//Count the lines of all .java files in this
directory and all its subdirectories.
//Write the total number of lines to stdout.

@Grapes([
  @Grab(group='org.codehaus.groovy.modules',m
odule='groosh',version='[0.3.6,)''),
  @GrabConfig(systemClassLoader=true)
])

import groosh.Groosh
Groosh.withGroosh(this)

def f = find('.', '-name', '*.java', '-ls')
def total = 0
def lines = grid { values,w ->
  def x = values[2,4,6,10]
  def s = x.join(' ')
  w.println(s)
  total += Integer.parseInt(values[6])
}

f | lines
lines >> stdout
println "Total: ${total}"

```

Sometimes the name of a shell command conflicts with a Groovy method (for example "grep"). This means that

```
gsh.grep(...)
```

does not execute the shell command, but the Groovy method `grep(...)`.

As a workaround for that you may prefix any shell command with `_` this means the example above becomes

```
gsh._grep(...)
```

The following example shows a more elaborate example. It uploads photos to a flickr account using the command line tool `flickrurl`. A photo set of this images is created and named after the current directory.

```
import static groosh.Groosh.groosh
import static
org.codehaus.groovy.groosh.stream.DevNull.de
vnull

ids = [:]
shell = groosh()

//get all images in this folder and upload
it to flickr
//remember the photo id we get from flickr
shell.ls().grep(~/*jpg/).each {
    println "Uploading file $it to flickr"
    flickrurl =
shell.flickrurl("upload",it,"friend","family
")useError(true)
    id = flickrurl.grep(~/*Photo
ID.*/*)[0].split(":")[1].trim()
    ids[it] = id
    println "Photo ID is: $id"
}
//we need to know the first photo id
firstKey = ids.keySet().toList()[0]
```

```
//create a set with the name of the
directory we are in right now
//use the id of the first photo as set cover
setName = shell.pwd().text.split("/")[-1]
println "Creating set: $setName"
flickrurl =
shell.flickrurl("photosets.create",setName,s
etName,ids[firstKey]).useError(true)
id =
flickrurl.grep(~/*Photoset.*/*)[0].split("
")[2].trim()
println "Photoset ID is: $id"
```

```
//make a backup of the ids in a file for
later reference
println "Writing ids to a file"
file = new File(shell.pwd().text.trim() +
"./flickrset")
file << "Photoset:" << id << "\n"
ids.each {
    file << it.key << ":" << it.value <<
"\n"
}
```

```
//the first photo is already part of the
photo set so lets remove it
ids.remove(firstKey)
```

```
//add the remaining photos to the photo set
ids.each {
```

```
println "Adding photo to set at flickr:  
$it"
```

```
shell.flickcurl("photosets.addPhoto",id,it.v  
alue) | devnull()
```

```
}
```

```
println "DONE"
```

## Developers

### *Source Control*

The Groosh source code is available from <https://svn.kenai.com/svn/groovy-groosh~subversion>.

### *Building*

#### **Building with Netbeans**

Open the Kenai project in Netbeans and run build.

#### **Building without Netbeans**

Check out the source from <https://svn.kenai.com/svn/groovy-groosh~subversion>

Groosh uses Maven 2 for building. You have to have Maven 2 installed. Just execute mvn in the groosh directory groosh gets build.

### *Contributing*

Please contact Alexander Egger [alexander.egger at gmail.com] by e-mail.

## Community

For mailing lists, issue trackers, forums etc see <http://kenai.com/projects/groovy-groosh>

## Groovy Jabber-RPC

### Groovy Jabber-RPC

## Jabber-RPC

allows you to make XML-RPC calls using the Jabber protocol. Groovy has a Jabber-RPC implementation which allows you to create a local Jabber-RPC server and to make calls on remote Jabber-RPC servers. Jabber servers are widely available and very easy to set up and run. The Google GTalk service uses Jabber and the Groovy Jabber-RPC package works over GTalk.

We use the excellent

## Smack

Jabber library from Jive Software to handle the protocol details.

## The Server

It's really easy to set up a server which provides a set of remotely callable functions.

1. Create a server object

```
import groovy.net.xmlrpc.*
import org.jivesoftware.smack.XMPPConnection

def server = new JabberRPCServer()
```

2. Add some methods

```
server.echo = {return it} // the closure is now named "echo" and is
remotely callable
```

3. Start the server

```
def serverConnection = new XMPPConnection("talk.example.org", 5222,
"example.org")
serverConnection.login("myServerId", "myServerPassword") // logging
in as myServerId@example.org
server.startServer(serverConnection)
```

4. You're done!

## The Client

It's pretty easy to make the remote calls too

1. Create a proxy object to represent the remote server

```
def clientConnection = new XMPPConnection("talk.example.org", 5222,
"example.org")
clientConnection.login("myClientId", "myClientPassword") // logging
in as myClientId@example.orgm
def serverProxy = new JabberRPCServerProxy(clientConnection,
"myServerId")
```

2. Call the remote method via the proxy

```
println severProxy.echo("Hello World!")
```

3. As long as myClientId@example.org and myServerId@example.org are buddies then the call will be made and the result returned

## GroovyJMS

### Introduction

GroovyJMS provides set of Groovy-style APIs, enhanced JMS API with [TMPGroovy Categories](#), and a powerful JMSPool bean, that make it very easy to use JMS in Groovy application. It transparently handle connection and session lifecycle, and adds some thread-safe convenient methods to the JMS APIs.

Take a simple example:

```
jms{
  "loginService".send [user:'groovy',
password:'jms']
  // or [user:'groovy',
password:'jms'].sendTo "loginService"
}
```

And a complex one:

```
// create a JMS Thread Pool with 10 threads
at max
def pool = new JMSPool(maximumPoolSize:10)
// subscribe to three destinations, each
with 2 threads
pool.onMessage([fromQueue:['queue0', 'queue1']
,fromTopic:'topic0',threads:2]){ m ->
println m}
```

This module is in its early stage. The provided example are fully functional, but some relatively complex scenario may require directly JMS API usage

You may visit the v0.1 page or the latest v0.2 docs

- [GroovyJMS - v0.1 Docs and Example| svn v0.1](#)
- [GroovyJMS Docs- v0.2 is in svn trunk | svn trunk](#)

Or check other information at:

- [GroovyJMS Roadmap and Planning](#) - Roadmap, Planning, and JMS API support status
- [GroovyJMS Design Docs](#) - Draft
- [Groovy Messaging Service API](#) - Summary of latest discussion
- [GroovyJMS Reference Links](#)

## GroovyJMS Design Docs

This page is under construction.

### Basic Operations

#### *Objects*

##### JMS

##### JMSPool

- Constructor arguments, refer to the [JavaDoc of ThreadPoolExecutor](#)
  - corePoolSize: 10, maximumPoolSize: 20, keepAliveTime: 360, unit: TimeUnit.MILLISECONDS

#### *APIs*

##### receive()

- for a queue, receive is the standard JMS receive method.
- for topics, the receive() method is a blocking operation that create a 'temporary subscription'. The 'temporary subscription' wait for the first message and then unsubscribe and return the message. If a 'within' expiry time is provided, the 'temporary subscription' will be unsubscribed when the 'within' time is reached, and return null. This feature is typically used with message selector, and is useful for simulating a synchronous operation with the asynchronous JMS infrastructure. e.g. a client use a new thread to send a request with some delay, and in the main thread listen to the server response topic in synchronous manner.

```
def pool = new JMSPool()
pool.send
toQueue:'chatService',message:\[user:'groovy100',room:100\],delay:500 //it
sends a message in another thread, the delay is avoid getting a result
before the subscribed to the topic
def result = pool.receive
fromTopic:'signedOnBroadcast',messageSelector:"newUser = 'groovy100'"
if (result) // you have successfully signed on
else // retry? or tell user to sign in again.
```

### Shared Features

Features that are used across APIs are listed in this section

#### Message Selector

- In JMS, Message Selector is a String that defines some filtering conditions for receiving messages. Refer to the [JMS Javadoc](#) for details
- In Groovy Messaging Service API, you may use the messageSelector parameter in receive and onMessage, the message selector parameter takes a String, a List, a Map, or a Closure with one parameter
  - For a String, the String will be used for every destination specified in the same call
  - For a List, it will be matched against the List of Queue and Topic respectively. If the number of destination is less than the number of message selector items, the excess message selectors will be ignored. If the number of destination is more than the number of message selector items, the excess destination will have no message selector. You could use null for destination or message selector in doing matching. When both fromQueue and fromTopic are used, the two list of destinations will be matched respectively with the same list of message selector.
  - For a Map, the key of the map parameter will be matched against the destination name.
  - For a Closure, it works similar to a map and it passes in the destination name or destination and expect a String of return value as the message selector

## Advanced Features

### JMSPool

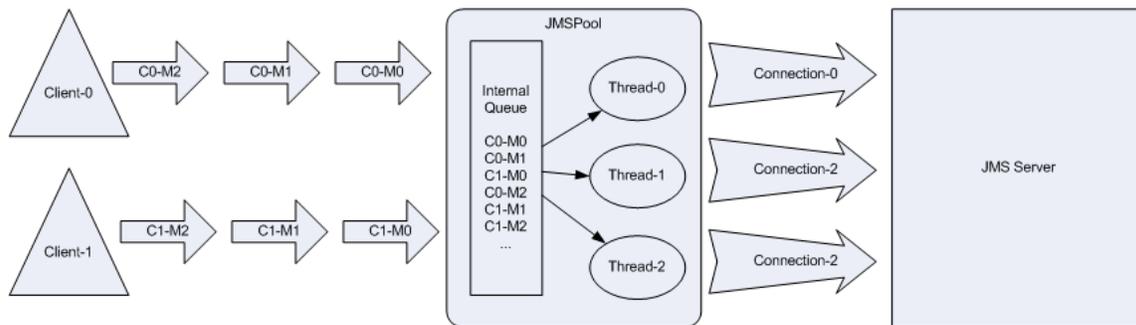
#### Overview

- JMSPool is an advanced JMS instance that allow multiple threading messaging for incoming and outgoing messages. It is built on Java 5 concurrency library and also utilize the ActiveMQ Pooled Connection Factory. So only Java 5 and AMQ are supported.
- For simple JMS usage,
  - a new connection and session are created on every JMS execution block. Resources are closed after usage.
  - user has to manage threads and concurrency. Without special coding, messages are sent and received in a single connection and session.
- The currently JMSPool is fairly simple. It does not support features like transaction, connection recovery etc. If you need a full feature JMS Pooling product, you are highly recommended to use Spring JMS or Jencks.
- With JMSPool
  - You could use a JMSPool instance with multiple threads for incoming and outgoing messaging
  - The threads are managed by Java 5 ThreadExecutor
  - Each thread uses a JMS instance
  - All closure in any thread are run in the use(JMSCategory){ } context. i.e. you could freely use GroovyJMS APIs

#### Outgoing messages (jms.send())

- For outgoing messages to Queue or Topic, it works like the following diagram

## JMSPool Outgoing Messaging Scenario



- Without the pool, multiple clients will have to send message in a serial manner. The JMSPool supports clients from multiple threads to send message asynchronously. When there are more concurrent message than worker threads, the messages will be put on a an internal buffer.
- Worker threads are spawn to handle connection in parallel, each thread makes a connection to the JMS server.
- In the initial implementation, the main advantage of the JMSPool over the original JMS are:
  - user could easily send outgoing message in parallel, e.g.

```
def pool = new JMSPool(maximumPoolSize:10)
10.times{ pool.send toQueue:'testQueue', message:"this is
message #${it}"}
```

It spawns 10 threads to send 10 messages in parallel.

- Asynchronous sending  
every user call is returned asynchronously after it reaches the internal buffer, instead of waiting for the message to reach the external JMS server. It potentially provide a better throughput when you have a large volume of messages to send. Notice that this is not necessary the ideal behavior in every case and you should be cautious when using it, and **without a recovery mechanism in place (as in the initial implementation), your outgoing message may lose** if there is exception in delivery and you'll only get an exception via the global exception handler. **T ODO** verify the exception implementation
- connection reuse  
every thread get a connection from the AMQ connection pool to send message. The connection pool is supposed to provide connection re-use. Session is not shared.
- parallel messaging for multiple destination (**NOT implemented yet**)

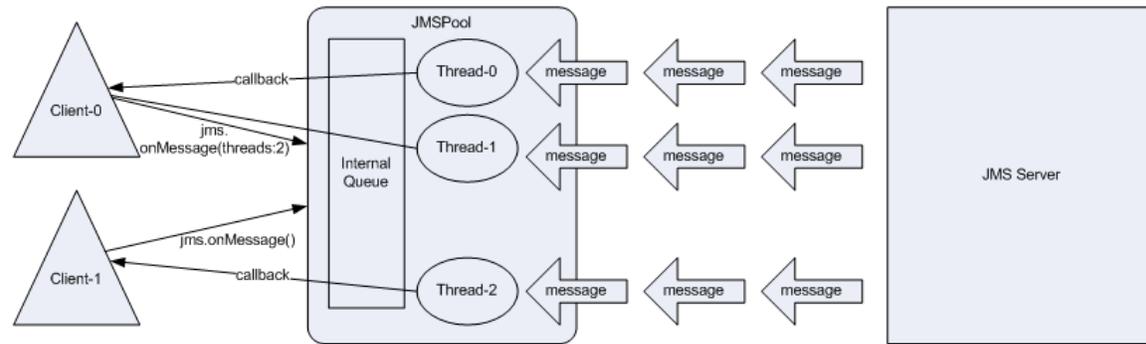
```
def pool = new JMSPool(maximumPoolSize:10)
pool.send toQueue:['queue0', 'queue1'], toTopic:'topic0',
message:"a msg for 3 dest", threads:3} // or threads:'auto'
```

A single send operation is split to three threads, and the message is sent in parallel. The number of threads should be less than the maximumPoolSize.

### Async incoming message

- Diagram

## JMSPool Async (onMessage) incoming message Scenario



- JMSPool assumes you want to use one thread per "destination and message selector" in each call. (**TODO: message selector is not implemented**) When there are multiple destination and message selector combination,
- `onMessage` allows user to receive messages one by one in an asynchronous basis. It works differently depends on the destination:
  - for a queue, multiple threads will be spawn with the same listener to the queue. As messages are received one by one per thread. You are able to receive messages at a much higher rate.
  - for a topic, it doesn't make sense to subscribe with more than one thread. So the threads parameter will be ignored.
  - for multiple queues and topics, every queue will be listened by the specified number of threads, and every topic will be subscribed by one thread only. (as the 'threads' parameter is ignored)
- **TODO:** perhaps the 'threads' parameter could be implemented in a way similar to Message Selector and take a list or map of value

### Sync receive message

- synchronous message receiving, e.g. `jms.receive fromQueue:"myQueue"`, works very similar to its asynchronous counterpart. But behave differently in the following ways:
  - when the receive method is used to receive all messages (it is by default if a 'limit' parameter is not used), there is no real need to use multiple thread and it may not improve performance unless you have a lot of messages. The 'threads' parameter will be honored and you should use it according to your need.
  - For queues, when multiple threads are used, it waits for every thread to complete their works, and aggregate all results to return to you. The 'with' closure for queue will be called once and only once with the aggregated result in the first parameter.
  - For topics, only one thread will be used for each topic-selector pair. When the number of topic-selector pairs is less than the number of 'threads', e.g. `fromTopic:['t0','t1','t2'], threads:2`, a 'temporary subscription' will be issued for the first 2 topics immediately, and only after one of them has return, a 'temporary subscription' will be issued for the 3rd topic. And only after all 3 topics are done, you'll get the aggregated return value.

## GroovyJMS Docs

This docs is for v0.2+

### Changes

#### v0.2

1. It is no longer a mandatory to provide a JMS ConnectionFactory. For a single JVM simple usage, you may

use the default in-memory, non-persistent ActiveMQ connection factory, or you could configure your JMSProvider in several ways.

2. The library is no longer used as a Groovy Category. The Groovy Category are hidden. There are a few different ways to use:

```
// 1
jms{ //require static import of groovy.jms.JMS.jms
  // your jms code
}

// 2
def jms = new JMS(){
  // your jms code
}

// and for further execution
jms.run{
  // more jms code
}

// 3
def jms = new JMS()
jms.xxx ; // this is for using the Groovy Messaging Service API
jms.close() //must be closed manually if JMS code are not executed in the
JMS Context
```

3. The base Category API are slightly modified and are refactored to the JMSCoreCategory. Two sets of new APIs: GroovyJMS API and Groovy Messaging Service API are provided. The latter are designed base on the Groovy Sql.

## Usage

### Concepts

- **GroovyJMS Execution Context**

- GroovyJMS mainly provide an execution context that you can use enhanced JMS API and convenient methods, and access to implicit variables

```
new JMS(){
  // inside the execution context, where magics occur
}

import static groovy.jms.JMS.jms;
jms{
  // inside the execution context, where magics occur
}
```

- GroovyJMS is based on JMS and preserve any JMS concepts and keywords. You could mix the usage of GroovyJMS API and JMS API in the execution context.
- **GroovyJMS Core API**
  - GroovyJMS Core API is based on the JMSCoreCategory, and shall be used in Execution Context.

APIs are provided for JMS Resource and Destination

- **JMS Resource - ConnectionFactory, Connection and Session.** You could create any of them outside the execution context and pass to the constructor of GroovyJMS, or you could use the implicitly created instances.

- Refer to the next section about JMS ConnectionFactory
- Inside the execution context, getConnection() and getSession() methods are attached to any class, so you could "connection" and "session" like an pre-declared variable. The connection and session are attached to a threadlocal JMS instance

```
new JMS(){
  println connection; // by default: ActiveMQConnection
  {id=ID:hostname-1212-1223775533432-2:0,clientId=ID:hostname-121
  2-1223775533432-3:0,started=false}
  println session; // by default: ActiveMQSession
  {id=ID:hostname-1212-1223775533432-2:0:1,started=false}
}
```

- JMS Resources are used interchangeably when creating lower level JMS Resource

```
def factory; //injected JMS ConnectionFactory
new JMS(){
  def conn0 = factory.connect(), conn1 = connection
  def session0 = factory.session(), session1 =
  connection.session()
}
```

- GroovyJMS provides high level API that use a single connection and session per thread. Unless you want a fine-grained control over the re-use of Connection and Session, it is not necessary to care about these resources.

- **JMS Destination - Queue, Topic**

- There are two types of JMS destination. GroovyJMS provides high level API to operate on any Queue or Topic, unless you need to get a reference for JMS Destination, you don't need to care about JMS Destinations.
- Short API are provided for creating JMS Destinations. Similar to JMS Resource, calling subject is interchangeable.

```
def factory; //injected JMS ConnectionFactory
new JMS(){
  def topic0 = factory.topic(), topic1 = connection.topic(),
  topic2 = session.topic()
  def queue0 = factory.queue(), topic1 = connection.queue(),
  topic2 = session.queue()
}
```

- The return value of topic() and queue() are JMS Topic and Queue respectively. However, in GroovyJMS Execution Context, you could use some operation APIs such as send(), receive() etc. directly on JMS Destination.
- APIs

```
topic.subscribe( subscriptionName:'x', messageSelector:'y',
noLocal:true, durable:false) ; //by default, durable is true,
subscriptionName is generated unique name, messageSelector is
null, noLocal is false, durable is true
```

- Message - MapMessage
  - mapMessage.toMap()
- **GroovyJMS API**
  - GroovyJMS API is based on the JMSCoreCategory, and shall be used in Execution Context
  - For simple usage, it is the only thing you need to learn. GroovyJMS API uses keywords from JMS API.
    - "send" and "receive" for Queue messages
    - "publish" and "subscribe" for Topic messages
  - You could call the API in two style
    - Message as subject, and sendTo

```
"Hello world".publishTo "topics/test"
[hello:'world'].publishTo "topics/test"
"Hello world".sendTo "queue/test"
[hello:'world'].sendTo "queue/test"
```

Ideas:

```
"abc".send() // send to the last used queue/topic
? allow user to configure a boolean to reverse the use of
send() ?
```

- Or use the destination/Message as subject

```
"topics/test".publish "Hello world"
"topics/test".publish [hello:'world']

"queue/test".send "Hello world"
"queue/test".send [hello:'world']
"queue/test".send( "Hello world", [replyTo:'queue/replyQueue'])

"topic/test".subscribe(){ println it.text }
"topic/test".subscribe(
//"topics/test".subscribe().with{ println it.text } //
"topic".subscribe() [w/o closure] is a synonym of
connection.topic()
"queue/test".receive() //to be implemented
"queue/test".receiveAll() //to be implemented
"queue/test".receiveAll( within:200.ms) //to be implemented

Message receivedMessage = "queue/test".recieve()
receivedMessage.reply "this is my answer"
```

- For certain operations, you don't even need a subject

```

subscribe("topics/test"){ println it.text }
//subscribe("topics/test").with{ println it.text } //synonym
def message = receive("queue/test")
def messages = receiveAll("queue/test")

//subscribeTo("topics/test").with{ println it.text }
//subscribeTo is a synonym of connection.topic()
//receive("queue/test").with{} //receiveFrom is a synonym of
connection.queue() , with is the default with{} in Groovy, it
is optional
//def messages = receiveAllFrom("queue/test").with{ it }
//receiveFrom is a synonym of connection.queue().receiveAll()
//receiveAllFrom("queue/test").each{ } //directly use the
iterator

//other Ideas, using map
send 'toTopic':'topic/test', 'message',[key:'value'],
'from':'fromclient'

```

- API in a spreadsheet [<http://www.editgrid.com/publish/html/user/mingfai/24976908/A2:M24>]

- **Groovy Messaging Service API**

- Groovy Messaging Service API do not need to run in the Execution Context.
- Groovy Messaging Service API is designed base on the Groovy core Sql API.

Groovy SQL	Possible JMS Usage	Remarks/Comment
Sql.newInstance()	JMS.newInstance(jmsConnecti onFactory)	or just use new JMS()
sql.eachRow	jms.eachMessage("myQueue") {m-> } jms.eachMessage("myQueue", [within:100]){}	jms.reply() could be used inside the closure  <b>reply is not implemented</b>

<pre>sql.firstRow</pre>	<pre>jms.firstMessage("myQueue") {   println it.text }</pre>	<pre>jms.onMessage(topic:"mytopic") { m -&gt; } jms.onMessage(topic: "mytopic",durable:false){m -&gt; } jms.onMessage(queue:"myQueue") { m -&gt; } jms.onMessage(queue:"myQueue", topic: "mytopic"){ m -&gt; } //subscribe queue and topic at the same time  //onMessage supports only one listener per session; if you need multiple subscription, use core API subscribe method and specific different subscriptionName  jms.onMessage( topic: ['topic0', 'topic1']{ m -&gt; } jms.onMessage( queue: ['queue0', 'queue1']{ m -&gt; } jms.onMessage( topic:'mytopic', queue: ['queue0', 'queue1']{ m -&gt; }  jms.stopMessage( topic:"myTopic") // unsubscribe the listener jms.stopMessage( topic:['myTopic','myTopic2']) //T <b>ODO not implemented</b></pre>
-------------------------	--	--

<pre>sql.execute()</pre>	<pre>def result = jms.receive(   fromQueue:'myQueue',   within:1000) jms.receive( fromQueue:   'myQueue', within:1000) { println it } jms.receive(   fromQueue:['queue0','queue1'],   within:1000, with:   { println it})</pre>	<p>fromTopic and fromQueue could be used together fromQueue always call receiveAll (to be enhanced to take a parameter to return only 1 message) if fromQueue has more than one queue, it return all messages of all queues fromTopic is asynchronous durable subscription, if both fromQueue and fromTopic are used together, any immediately available messages in any fromQueue will return first, and new message will be delivered to the same with closure in a later time either use <code>jms.receive([:]){ }</code> or <code>jms.receive([xxx:yyy, with:{}])</code> ; the former override the later within is a per queue timeout interval, not total; if there are more than one queue, each are retrieved in sequence</p>
<pre>sql.executeUpdate()</pre>	<pre>jms.send( toTopic:'myTopic',   message:[key:value], replyTo:"")</pre>	<p>toTopic and toQueue could be used together toTopic and toQueue support collection</p>

- All return value or closure handler result are JMS Message. It might be changed to String, Map, Byte[], Stream and Object in the future.
- Similar to Sql, fine-grained control over connection/session could be done by passing in a JMS Connection or Session in the JMS.newInstance(), and user has to close the session/connection themselves

### **JMS Connection Factory**

To use GroovyJMS, you need to decide which JMS implementation you'll use. There are several options:

1. use the default in-memory, not-persistent ActiveMQ broker and connection factory, this is created by the ActiveMQJMSProvider class. Notice that the ActiveMQJMSProvider will start a ActiveMQ broker if it is not existed already, and will add a "vm://localhost" transport connector URL if not existed. In this case, the syntax is as simple as:

```
new JMS() {
  // your jms code
}
```

2. Provide a JMSProvider by
  - a. specific a "groovy.jms.provider" system property that point to a class that implements groovy.jms.provider.JMSProvider
  - b. set a JMSProvider to the JMS.provider static variable, e.g.

```
JMS.provider = { return new
ActiveMQConnectionFactory("vm://localhost") } as JMSProvider
```

3. Provide Connection Factory or Connection in runtime. it is the most recommended approach. You may utilize your JEE container or dependency injection framework to inject the required JMS resources to your class, and support the connection or factory to GroovyJMS.

```
def myConnection; //injected
new JMS( myConnection ){
  // your jms code
}

// OR
def myConnectionFactory; //injected
new JMS( myConnectionFactory){
  // your jms code
}
```

#### So how cool is GroovyJMS in compare to JMS?

- simplified api
- subscribe to multiple topics and queues at the same time
- simple listener implementation by using Closure
- Groovish API , e.g. jms.eachMessage(), jms.onMessage()
- English language style api: jms.send toQueue:'myQueue',message:'hello world'
- etc.

#### Other important information

- **autoClose**
  - by default, all connection are closed at the end of an execution context, or after any operation for Groovy Messaging Service API.
  - For Groovy Messaging Service API, if you need to do more than one operation, you have to set autoClose to false, and call close() explicitly

```
def jms = JMS.newInstance()
jms.setAutoClose(false)
JMS.close()
```

- When using in execution context, it's unommon to disable autoClose. As one of the purpose of the execution context is to autoClose resource for you. If you need to use execution context and want to disable autoClose, one of the ways is:

```
def jms = new JMS(){
    jms.setAutoClose(false)
    // your jms code
}
```

- Or you'd better directly use the JMS Category without setting up an execution context.

```
use(JMSCategory){
    jms.session() ; // notice that you need to establish a session
    before doing anything with JMSCategory
    "queue".send("message")
    assertNotNull("queue".receive(waitTime:100))
}
```

### Unsupported Features or Limitation

- nested usage is not supported yet. you may still use the lib in nested manner but you have manage the Connection and Session carefully.
- For example:

```
def connFactory, conn
new JMS(conn){
    // outer JMS code
    new JMS(conn){
        // inner JMS code
    }
    //exception may be thrown at the end as the connection is closed
}
```

- you are recommended to avoid nested usage. If you use it, try to use a multiple connection.
- There is no support for transaction. all messages are auto-committed

### GroovyJMS Reference Links

This page provides some links to resources related to GroovyJMS

### Projects

#### JMS Implementations

- JMS
  - API: <http://java.sun.com/javaee/5/docs/api/javax/jms/package-summary.html>
- Grails
  - [JMS Plugin](#)
  - [Proposal for Messaging Integration](#)
- Open Source JMS Implementations
  - [ActiveMQ](#)
  - [JBoss Messaging](#)

#### JMS Related Projects

- JMS Pool
  - <http://jencks.codehaus.org/>
- JMS Tools
  - <http://www.hermesjms.com/>

### Articles about JMS

- [Performance testing open source JMS part 1](#)
  - JBoss Messaging benchmark
- [Async vs. Sync JMS Listener, MDB vs. Spring 2 MDB POJO's](#)
  - about connection pooling
- [Best practices to improve performance in JMS](#)

### GroovyJMS Roadmap and Planning

#### Draft Roadmap

- 0.1 - current release
  - The 0.1 release provides a set of JMS Category APIs for community review
  - develop a wiki page that provide examples of the GroovyJMS usage
- 0.2 - next release - end of 2008
  - Adopt general Groovy style and standard, e.g. JDK Logger
  - First revision to **Groovy-style API**
  - Provide API for all key JMS usages
    - In 0.1, the API support basic usage only. User has to use JMS API directly if they want to call certain JMS API that takes more configurable arguments. e.g. to send a Map message would require users to construct a JMS Message in v0.1
  - Clarify possibly Groovy messaging usage
    - ~~Check if it is possible to enhance Groovy Category to provide a pre-execute and post-execute API. This makes a big difference to the GroovyJMS API~~ Revised to use a Closure in Closure pattern, user code will be changed from use(JMS) { jmscode } to **new JMS(){ jmscode }**
    - Develop a wiki page with a table that list key JMS usage, current GroovyJMS API and proposed Groovy usage
- 0.3
  - Implement an optional **Groovy Builder for constructing ActiveMQ Broker**, and adjust default behavior.
  - Continuous to evolve and add Groovy-style API
  - Integrate with **Grails JMS Plugin**
  - Provide **build script**
  - **Full test coverage**
- 0.5 - begin to evolve to a Groovy Messaging Service
  - **No longer assume user to have JMS knowledge**
  - Use **Grape/@Grab** to provide underlying JMS implementation, by default Apache ActiveMQ, so users may have an option to use JMS without any special works to download JMS implementation jar files. JMS Implementation configuration could be done in the GroovyJMS Builder.
  - ~~Implement an **annotation** that make a method bind to the GroovyJMS context transparently. so user needs not to use new JMS() and could directly use GroovyJMS resource and API~~ the new jms{ } usage is very clean and it's not really necessary to use annotation
  - Provide English language style / Domain Specific Language way for messaging.
  - Provide an example that demonstrate **support for JBoss Messaging 2.0**
  - **Benchmark** the performance difference between implementing the project with Groovy and Java. **Port to Java** if there are significant performance difference
- 1.0 - Provide a full function **Groovy Messaging Service** as the standard way of using messaging enterprise

Groovy applications

- no assumption of JMS dependency
- **support non-JMS features** such as Message Group, Virtual Topic as standard features

### TODO List

- Put the v0.1 source code to svn
- Remove Log4j dependency and the MDC usage
- API Implementation
  - for send(), support Map and other type of JMS messages
  - for subscribe(), support closure with explicitly casting to MessageListener
- Naming convention
  - following JMS to use createQueue rather than queue()? No, createQueue is for creating a Queue. For queue(), it isn't really just a Queue but a Queue with MessageProducer functionalities

### Status of JMS API Support

refer to the linked spreadsheet

- <http://www.editgrid.com/user/mingfai/GroovyJMS>

### GroovyJMS - v0.1 Docs and Example

This page is kept for archive. Please visit the [GroovyJMS doc page](#) for the latest documentation.

### Examples

Let's start with some examples first:

#### Setup JMS Connection Factory

1. Download the [GroovyJMS-v0.1.zip](#) and extract the groovy.jms.JMS.groovy to your project
  - remarks:
    - the current release is for review. it includes all necessary jar and IntelliJ IDEA profile for running the unit test. You basically only need one JMS.groovy file plus any depending jars, that may have existed in your project already.
    - The attached zip bundle with **ActiveMQ 5.3 snapshot compiled with JDK 6**. You are highly recommended to try the new 0.2 snapshot by checking out from svn
2. For every example, it's assumed a JMS Connection Factory called "jms" is existed. For example, you could create a ActiveMQ connection factory programmatically with:

```
ConnectionFactory jms = new  
ActiveMQConnectionFactory(brokerURL:  
"vm://localhost");
```

#### Simple usages

1. Subscribe to a Topic message

```
use(JMS){
  jms.topic("greeting").subscribe({Message m \-> println "hey i got a
message. it says, '${m.text}'"} as MessageListener);
  jms.close(); //optional
}
```

## 2. Send a message to a Queue or Topic

```
use(JMS) {
  jms.topic("greeting").send("I'm joining the JMS party"); // use
  jms.queue("greeting queue") for sending to a queue
  jms.close(); //optional
}
```

## 3. Receive a Queue message

```
use(JMS){
  Message message = jms.queue("greeting").receive(1000); // it does \*not\*
mean get 1,000 message, see the note below this box
  List<Message> messages = jms.queue("greeting").receiveAll(1000); // this
retrieve all messages within the 1000ms timeout interval
  jms.close(); //optional
}
```

The receive parameter means "receives the next message that arrives within the specified timeout interval", check JMS JavaDoc for details:

<http://java.sun.com/javaee/5/docs/api/javax/jms/MessageConsumer.html>

## 4. Reply to a message

```
// the first guy
use(JMS){
  Queue replyQueue = jms.session().createQueue("replyQueue"); // notice
that createQueue is the original JMS API, it's just an example, u could
use jms.queue("replyQueue")
  jms.queue("greeting").send("hey, please reply to me privately",
[JMSCorrelationID: 'privatePlease', JMSReplyTo: replyQueue])
  jms.close(); //optional
}

// another guy in another thread
use(JMS){
  jms.queue("greeting").receive(1000)?.with{
    // please do something here, otherwise the example does not make sense
    it.reply("hey, let me tell you secretly")
  };
  jms.close(); //optional
}
```

## Advanced Usages/Issues

### 1. Connection and Session re-use

By default, it reuses a single JMS Connection and Session. The connection and session are created when any of connect(), session(), topic() or queue() method is first called, and the connection and session are bound to internal ThreadLocal variables until close() is called. So,

- connection and session are not shared across threads, it's thread-safe
- you have to call close() to remove a session in order to use a new connection and session.

### 2. When to close()?

- The close() method call a connection.start(), unset all ThreadLocal variables, and call the connection.close() to close the connection.
- For server applications, every request are run in its own thread, and the thread may be re-used. It's recommended to close() a connection explicitly. Notice that it may not necessarily a problem to re-use a thread scope session.

### 3. How to obtain reference to JMS resource and reuse?

- For singleton services in server applications, notice that the GroovyJMS is thread safe in a way different thread has its own connection and session. If you want to share the Connection and/or Session, simply create the connection and session. If you want to utilize the convenient connect() and session() methods, you could use:

```
def Connection jmsConnection;
use(JMS){
    jmsConnection = jms.connect();
    jms.cleanupThreadLocalVariables(); //clear all ThreadLocal variables
}
```

- It's a matter of taste. You for sure can call the original jms.createConnection() directly. But if you will do some jms operation, it will save some code. You could also retrieve any created Connection and Session with:

```
//no need the "use(JMS)" for this case
Connection connection = JMS.connection.get();
Session session = JMS.session.get();
```

- Because dynamic method are added to different JMS objects, the following are essentially the same:

```
jms.queue("myQueue")
jms.connect().queue("myQueue")
Connection conn = jms.connect(); conn.queue("myQueue")
jms.session().queue("myQueue")
// but no jms.connect().session().queue("myQueue"); it's too much\!
Please use the JMS createSession() api
```

### 4. When you need to call start()?

- a. If you have stopped a connection, you have to start it. otherwise, you won't get any message. Refer to the JMS specification. The only difference of GroovyJMS is that you could call start() directly on a ConnectionFactory.
- b. When a connection is first obtained, the start() method is called. You need to explicitly stop it if you don't want to receive message.
- c. If you don't close your connection and keep reusing it, you may probably need to call start() at some

points.

## 5. Exception Handling

a. it creates a exception listener that log to a log4j logger

```
conn.setExceptionListener({JMSEException e \-> logger.error("JMS  
Exception", e)} as ExceptionListener);
```

b. For some operations, it may throw RuntimeException. e.g. if you call reply on a message that does not have a reply address. It may be better to throw JMSEException. (looking for your comment, notice that JMSEException is not RuntimeException)

6. Notice that GroovyJMS simply add methods to the default JMS API. You could do whatever you like directly with JMS API. The JMS.groovy has under 250 line of code (incl. min comments) that it should not difficult to understand.

### Dependency, Limitations, and TODO

- You can only send TextMessage or manually create a JMS Message for now! 😊 If anyone needs, i could add the convenient method to send other types of JMS message such as Map Message.
- It obviously require JMS api and a JMS implementation. You have to provide the JMS Factory, the following is an example to setup ActiveMQ with Spring:

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:amq="http://activemq.apache.org/schema/core"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd  
http://activemq.apache.org/schema/core  
http://activemq.apache.org/schema/core/activemq-core-5.1.0.xsd">  
  
    <amq:broker useJmx="false" persistent="false">  
        <amq:transportConnectors>  
            <amq:transportConnector uri="vm://localhost"/>  
        </amq:transportConnectors>  
    </amq:broker>  
    <amq:connectionFactory id="connectionFactory"  
brokerURL="vm://localhost"/>
```

And you'll need the following jars:

**For ActiveMQ core:**

**activemq-all-\*.jar // tested with  
activemq-all-5.3-SNAPSHOT.jar only**

**For Spring AMQ prefix:**

**xbean-spring-2.6  
activeio-core.jar (it's not needed if you  
don't use the io persitence)**

**Groovy Messaging Service API**

Content on this page is under development, check the following email thread for the time being: <http://www.nabble.com/Groovy-JMS-Category-td19861454.html>

**Design Requirement**

- API must be in Groovy-style
  - use typical Groovy syntax like with, each etc.
  - sounds like English language
- Follow JMS concept and keywords, but need to to use exact name and API
- Simple and Easy to use
- Thread-safety
- JMS resource re-use and direct JMS usage
  - Allow provision of ConnectionFactory, Connection or Session for reuse.
  - Return sensible JMS resource for reuse or keeping reference

**AS-IS, v0.1 release**

- assumed to be start with a user provided JMS ConnectionFactory named "jms". In future release, this should not be required.

Type	JMS Usage (In Groovy Syntax with Static Type for clarity)	Proposed/Current API	Reference
Create Connection	Connection conn = jms.createConnection()	jms.connect()	v0.1
Create Session	Session session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE)	jms.session() conn.session()	v0.1

Create Topic	<pre>Topic topic = session.createTopic("test Topic");</pre>	<pre>jms.topic("testTopic") conn.topic("testTopic") session.topic("testTopic")</pre>	v0.1
Subscribe to Topic	<pre>TopicSubscriber subscriber = session.createDurableSu bscriber(topic, "sub-name") MessageListener listener = {Message m -&gt; println m.text } as MessageListener subscriber.setMessageLis tener(listener)</pre>	<pre>topic.subscribe( {} as MessageListener )</pre>	v0.1

## Groovy SQL-style API

Content in this section is moved to [GroovyJMS Docs](#)

### Proposed

- Create
  - `jms.createQueue` withName: "replyQueue"
  - `Queue replyQueue = jms.createQueue("replyQueue")`
- Topic subscription
  - `jms.subscribeTo topic/ subscribe toTopic: "greeting", withListener/with:{ Message m -> println "hey i got a message. it says, \${m.text}" }`
  - `jms.subscribe toTopic: "greeting", onMessage:{ Message msg -> println "hey i got a message. it says, \${m.text}" }`
  - `jms.subscribe("greeting"){println "hey i got a message. it says, \${m.text}" }`
  - `jms.subscribe("greeting") { println "hey i got a message. it says, \${m.text}" }`
- Queue receive
  - `jms.receiveMessage fromQueue: "greeting", withinTimeoutOf: 1000`
  - `jms.receiveMessage/receive fromQueue: "greeting", within: 1000.ms`
  - `jms.receiveAllMessages/receiveAll fromQueue: "greeting", within: 1000.ms`
  - `jms.receiveMessage fromQueue: "greeting", within: 1000.ms, withMessage: { Message msg -> reply withTextMessage }`
  - `jms.receive ("greeting", within: 1000.ms) { reply "hey, let me tell you secretly" }`
- Send message
  - `jms.send textMessage: "hey, please reply to me privately", onQueue: "greeting", withAttributes: [JMSCorrelationID: 'privatePlease', JMSReplyTo: replyQueue]`
  - `jms.send message: "hey, please reply to me privately",`



### ***Team Members***

- Yann Richet - Contributor to JMathTools Java project

GroovyLab is just provided to start a math engineering DSL sub-project of Groovy. If you need GroovyLab, GroovyLab also needs you...

### **Download**

#### ***Distributions***

Source release available at [GroovyLab website](#)

#### ***Installing***

Just extract the GroovyLab archive, and try to run examples cases using groovylab.bat or groovylab script: '

- groovylab examples/simpleTest.gvl
- groovylab examples/moreTest.gvl

#### ***Pre-requisites***

GroovyLab is based on Groovy 1.1 and Java 1.5.

### **Documentation**

The following example shows GroovyLab in action:

```
import static org.math.array.Matrix.*  
import static org.math.plot.Plot.*  
  
def A = rand(10,3)  
println A  
plot("A",A,"SCATTER")
```

```

import static org.math.array.Matrix.*
import static org.math.plot.Plot.*

def A = rand(10,3)    // random Matrix of 10
rows and 3 columns
def B = fill(10,3,1.0) // one Matrix of 10
rows and 3 columns
def C = A + B    // support for matrix
addition with "+" or "-"
def D = A - 2.0    // support for number
addition with "+" or "-"
def E = A * B    // support for matrix
multiplication or division
def F = rand(3,3)
def G = F**(-1)    // support for matrix
power (with integers only)

println A    // display Matrix content

plot("A",A,"SCATTER") // plot Matrix values
as ScatterPlot

def M = rand(5,5) + id(5) //Eigenvalues
decomposition
println "M=\n" + M
println "V=\n" + V(M)
println "D=\n" + D(M)
println "M~\n" + (V(M) * D(M) * V(M)**(-1))

```

## Groovy Monkey

Groovy Monkey is a dynamic scripting tool for the Eclipse Platform that enables you to automate tasks, explore the Eclipse API and engage in rapid prototyping. In fact, I think that if you are working on automating tasks in Eclipse or doing Plugin development in general, this tool is one for you. Groovy Monkey can allow you to try out code and do rapid prototyping without the overhead of deploying a plugin or creating a separate runtime instance.

Groovy Monkey is based on the Eclipse Jobs API, which enables you to monitor the progress in the platform seamlessly and allows you to write your scripts so that users can cancel them midway. Groovy Monkey is also based on the Bean Scripting Framework ([BSF](#)) so that you can write your Groovy Monkey scripts in a number of languages ( particularly Groovy ). In fact you can write in Groovy, Beanshell, Ruby or Python. The project update site is located at the Groovy-Monkey SourceForge site ( update sites: [Eclipse v3.2](#) or [Eclipse v3.1.2](#) ). Direct download of Groovy Monkey directly goto [http://sourceforge.net/project/showfiles.php?group\\_id=168501](http://sourceforge.net/project/showfiles.php?group_id=168501)

### **Requirements**

#### *Eclipse Version compatibility*

- ✓ Eclipse 3.1 : working [update site](#)
- ✓ Eclipse 3.2 : working [update site](#)

#### *Java Version compatibility*

- ✗ 1.4
- ✓ 5.0
- ✓ 6.0

### **Addition one: metadata keywords**

#### *LANG metadata keyword*

First, there is a new metadata keyword called LANG, which as is implied, determines what scripting language you wish to use. Here is an example of an Groovy Monkey base example ported to Groovy:

## FindSystemPrints\_groovy.gm

```
/*
 * Menu: Find System Prints > Groovy
 * Kudos: Bjorn Freeman-Benson & Ward
Cunningham & James E. Ervin
 * LANG: Groovy
 * Job: UIJob
 * License: EPL 1.0
 */

def files =
resources.filesMatching(".*\\.java")
for( file in files )
{
    file.removeMyTasks( metadata.path() )
    for( line in file.lines )
    {
        if( line.string.trim().contains(
'System.out.print' ) )
        {
            line.addMyTask( metadata.path(),
line.string.trim() )
        }
    }
}
window.getActivePage().showView(
'org.eclipse.ui.views.TaskList' )
```

Notice the LANG tag, that is all there is to that. There is also a New Groovy Monkey Script wizard available that has

the  
legal values in pulldown menus.

#### ***Job metadata keyword***

The Job metadata tag allows you to specify what kind of Eclipse Job that your Groovy Monkey script will be run in. By default it is set to Job, but UIJob and WorkspaceJob are also available. In Eclipse it is best to run almost all of your code from outside the UI Thread so UIJob is not recommended. To enable you to access UI elements from within your script there is a Runner DOM that enables your script to pass a Runnable object that can be called from the `asyncExec()` or `syncExec()` methods. For Groovy the JFace DOM allows you to pass a Closure directly to be invoked from either `asyncExec()` or `syncExec()`.

#### ***Exec-Mode metadata keyword***

The Exec-Mode metadata keyword allows you to specify whether the script should be run in the background (default) or foreground. The foreground mode has Eclipse directly pop up a modal dialog box that shows the user the progress of the script, the background mode does not.

#### ***Include metadata keyword***

The Include metadata keyword allows you to specify a resource in your workspace and to directly add it to the classloader of your Groovy Monkey script. Examples would obviously include jar files or directories.

#### ***Include-Bundle metadata keyword***

The Include-Bundle metadata keyword allows you to have an installed bundle be directly added to the classloader of your Groovy Monkey script.

#### ***Addition two: Outline view***

Secondly, the outline view is populated showing the binding variable names and types with their publicly available methods and fields. This can be useful since the DOMs are loaded on your eclipse ide as plugins and not in your workspace view. Even if you were to load in the DOMs into your workspace, there is still a great deal of switching that must be done.

You can double click on a type in the outline view and have it open the source directly in your editor, if you have included external plugins in your Java search path.

There is also an "Installed DOMs" view that shows the installed DOM plugins currently in your Eclipse workbench. The editor also includes a right click command to display a dialog that lists the and will install available DOMs to your script.

#### ***Addition three: Groovy SWT and Launch Manager DOMs***

Thirdly, there are new DOMs that are located on the update site that include direct access to a console for output, enable you to script your launch configurations together and a wrapper for the Groovy SWT project as a DOM.

Here is an example of a script, copied from the examples given in Groovy-SWT, ported into Groovy Monkey. The Groovy-SWT DOM is now included by default when you have the `net.sf.groovyMonkey.groovy` fragment installed. The `net.sf.groovyMonkey.groovy` fragment contains Groovy Monkey's support for the Groovy language.

## SWTTest.gm

```
/*
 * Menu: Test SWT
 * Kudos: James E. Ervin
 * License: EPL 1.0
 * Job: UIJob
 */

def subapp = jface.shell( window.getShell()
)
{
    GridLayout()
    group( text:'Groovy SWT',
background:[255, 255, 255] )
    {
        GridLayout()
        label( text:"groove fun !"
,background:[255, 255, 255] )
        label( text:"Email: ckl@dacelo.nl",
background:[255, 255, 255] )
    }
}
subapp.pack()
subapp.open()
```

Here is an example of a script that uses the Launch Configuration Manager DOM along with the JFace/SWT DOM and the Console output DOM. It prints out the list of all your available launch configurations and allows you to select which ones you wish to string together.

## LaunchManager.gm

```
/*
 * Menu: Launch Manager
 * Kudos: James E. Ervin
 * License: EPL 1.0
 * LANG: Groovy
 * Job: UIJob
 * DOM:
http://groovy-monkey.sourceforge.net/update/
net.sf.groovyMonkey.dom
 */
import org.eclipse.swt.SWT

def configurations =
launchManager.manager().getLaunchConfigurati
ons()
def selected = []
shell = jface.shell( text: 'Select desired
configurations:', location: [ 100, 100 ] )
{
    gridLayout()
    table = table( tooltipText: "Select
configurations to execute",
style: 'check,border,v_scroll,h_scroll,single
' )
    {
        gridLayout()
        gridData( horizontalAlignment:
SWT.FILL, verticalAlignment: SWT.FILL,
grabExcessHorizontalSpace: true,
```

```

                                grabExcessVerticalSpace:
true, heightHint: 400 )
  for( config in configurations )
  {
    tableItem().setText( "${config.name}" )
  }
}
  composite()
  {
    gridLayout( numColumns: 2 )
    button( text: 'run', background: [
0, 255, 255 ] )
    {
      onEvent( type: 'Selection',
closure:
      {
        for( item in table.items )
        {
          if( !item.checked )
            continue
          selected.add( item.text
)
        }
        shell.close()
      } )
    }
    button( text: "cancel", background:
[ 0, 255, 255 ] )
    {
      onEvent( type: 'Selection',
closure:

```

```
        {
selected.clear()
                shell.close()
        } )
    }
}
shell.pack()
shell.open()
while( !shell.isDisposed() )
{
    if( !shell.display.readAndDispatch() )
        shell.display.sleep()
}
selected.each
{
    out.println "${it}"
}
```

```
launchManager.launch( 'launch test',  
selected )
```

## GroovyRestlet

### Module Overview

GroovyRestlet is a simple DSL for constructing [Restlet](#) application in a simple, shortcutting syntax.

### Download

#### *Distributions*

GroovyRestlet is distributed as a single jar.

#### **Current release**

GroovyRestlet 0.3-SNAPSHOT, donwload [here](#)

### Changes

- Upgrade to Restlet 1.1 M3
- Upgrade to Spring 2.5.2
- `directory` constructor now only accept `root` attribute as valid URI.
- Bug fixes

#### **Old releases**

- GroovyRestlet 0.2, download from [here](#)
- GroovyRestlet 0.1, download from [here](#) for Restlet 1.1-SNAPSHOT

#### *Pre-requisites*

At current moment, GroovyRestlet depends following:

- Groovy 1.5
- Java 1.5
- Restlet 1.1-SNAPSHOT
- SLF4J 1.4.3 as logging tool
- Spring 2.5 if spring integration is required
- commons-lang 2.3

#### *Installing*

Drop GroovyRestlet and all dependencies in your Java classpath.

### Documentation

#### *Quick start*

First, create an instance of `GroovyRestlet` first.

```
GroovyRestlet gr = new GroovyRestlet()
```

**i** If you want Spring support, provide an instance of `ApplicationContext` as the constructor parameter.

```
GroovyRestlet gr = new  
GroovyRestlet(appCtx);
```

Then, prepare your Restlet building script using simple GroovyRestlet DSL syntax. Calling `GroovyRestlet.build(URI)` then done.

Check [GroovyRestlet User Guide](#) for detail user information.

#### Examples

Here is one GroovyRestlet DSL example, inspired from [Restlet tutorial 11](#).

```
builder.component{  
    current.servers.add(protocol.HTTP,  
8182)  
    application(uri:""){  
        router{  
            def guard =  
guard(uri:"/docs",  
scheme:challengeScheme.HTTP_BASIC,  
            realm:"Restlet  
Tutorials")  
            guard.secrets.put("scott",  
"tiger".toCharArray())  
            guard.next =  
directory(root:"", autoAttach:false)  
            restlet(uri:"/users/{user}",  
handle:{req, resp->
```

```

        resp.setEntity("Account
of user \"${req.attributes.get('user')}\"",
mediaType.TEXT_PLAIN)
    })

restlet(uri:"/users/{user}/orders",
handle:{req, resp->
        resp.setEntity("Orders
or user \"${req.attributes.get('user')}\"",
mediaType.TEXT_PLAIN)
    })

restlet(uri:"/users/{user}/orders/{order}",
handle:{req, resp->
        def attrs =
req.attributes
        def message = "Order
\"${attrs.get('order')}\" for User
\"${attrs.get('user')}\""
        resp.setEntity(message,
mediaType.TEXT_PLAIN)
    })

```

```
    }  
  }  
}.start()
```

For more examples:

- [Complete examples](#) for [Restlet tutorial](#)
- [SpringIntegrationExamples](#)

## Developers

### *Source Control*

<http://svn.codehaus.org/groovy-contrib/groovyrestlet/trunk>

### *Building*

GroovyRestlet uses Maven 2 as its building tool.

### *Contributing*

Feel free to contribute.

### *Mailing List(s)*

<http://groovy.codehaus.org/Mailing+Lists>

### *Issue tracker*

<http://code.google.com/p/groovy-restlet/issues/>

## GroovyRestlet User Guide

### *Introduction*

#### Very simple usage

An instance of `GroovyRestlet` is the starting point of everything.

You can programmatically create an instance of `GroovyRestlet`:

```
GroovyRestlet gr = new GroovyRestlet();//#1
```

```
GroovyRestlet grWithSpring = new  
GroovyRestlet(springApplicationContext);  
//#2
```

Constructor method of #2 indicates that this instance of `GroovyRestlet` will consult Spring's `ApplicationContext` when constructing its component. So the Spring integration is done.

Alternatively, you can choose to declare `GroovyRestlet` in Spring bean definition. By this way, `GroovyRestlet` will automatically use its parent context when constructing its components.

```
<bean id="groovyRestlet "  
class="org.lpny.groovyrestlet.GroovyRestlet "  
>
```

#### Spring Integration

When Spring's `ApplicationContext` is provided, `GroovyRestlet` will try to consult Spring context for creating its components.

Two special attributes are provided for integrating `GroovyRestlet` and Spring.

- **ofBean**: to tell `GroovyRestlet` that to create a component from Spring context by a bean name;
- **ofClass**: to tell `GroovyRestlet` that to create a component using Spring's [AutowireCapableBeanFactory](#)

**i** `GroovyRestlet` will only use `AutowireCapableBeanFactory` to create instance indicated by **ofClass** when Spring context is provided. Otherwise it will create an instance of that class using Java reflection

#### Technical detail

##### `GroovyRestlet`

As the only entry point of `GroovyRestlet`, this instance automatically declares a number of global variables in the Groovy context. Users are also be able to provide their special context variables. In order to do that, users can provide a map of customer context when calling `build` method.

```
groovyRestlet.build(userDefinedContext,  
scriptURI);
```

#### **⚠ Be careful**

This **userDefinedContext** is only visible for each call of `build` method. It will be merged into Groovy context and removed after each calling. And variables of **userDefinedContext** can override values of default context. Be careful when doing this.

##### `RestletBuilder`

`RestletBuilder` is a simple implementation of Groovy's `FactoryBuilderSupport`. By default, all constructor factories are registered. Users are able choose which constructors are to be used freely.

To do this programmatically, just set a list of constructor factories after a build instance is created.

```
RestletBuilder builder = new  
RestletBuilder()  
  
builder.setFactories(yourFactoryList);  
  
groovyRestlet.setBuilder(builder)
```

Alternatively, you can do this in Spring bean definition

```
<bean id="groovyRestlet"  
class="org.lpny.groovyrestlet.GroovyRestlet"  
>  
  <property name="builder"  
ref="restletBuilder"/>  
</bean>  
  
<bean id="restletBuilder"  
class="org.lpny.groovyrestlet.builder.Restle  
tBuilder">  
  <property name="factories">  
    <list>  
      <!-- any factory instance -->  
    </list>  
  </property>  
</bean>
```

#### Global variables

For convenience, `GroovyRestlet` add a number of Restlet-related variables to Groovy global context. Users can directly refer them in there building scripts.

Name	Usage	Example
<b>builder</b>	The reference of a builder	<code>builder.component()</code>
<b>protocol</b>	Shortcut to Restlet <a href="#">Protocol</a>	<code>protocol.HTTP</code>
<b>mediaType</b>	Shortcut to Restlet <a href="#">MediaType</a>	<code>mediaType.TEXT_PLAIN</code>
<b>status</b>	Shortcut to Restlet <a href="#">Status</a>	<code>status.CLIENT_ERROR_NOT_FOUND</code>
<b>challengeScheme</b>	Shortcut to Restlet <a href="#">ChallengeScheme</a>	<code>challengeScheme.HTTP_BASIC</code>
<b>redirectorMode</b>	Shortcut to mode of Restlet <a href="#">Redirector</a>	<code>redirectorMode.MODE_CLIENT_FOUND</code>
<b>routingMode</b>	Shortcut to mode of Restlet <a href="#">Router</a>	<code>routingMode.BEST</code>
<b>transformer</b>	Shortcut to mode of Restlet <a href="#">Transformer</a>	<code>transformer.MODE_REQUEST</code>
<b>global</b>	Special shortcut providing a set of global variables and methods	

At current moment, you can call `global.status(404)` to create a `CLIENT_ERROR_NOT_FOUND` status.

#### Constructors

Every constructor is support **ofBean** and **ofClass** attributes. The value of **ofBean** must be a Sting. **ofBean** is only effected when a Spring context is provided. The value of **ofClass** can either be a String or an instance of Java Class.

According to [Groovy Builders](#), constructors can be nested to each other. In that case, GroovyRestlet will automatically set up Restlet-specific relationship between parent and child instances. But user can disable this feature by setting **autoAttach** attribute as `false`.

In Restlet, the Router.attach method returns a Route instance. In order to do some special process on the Route instance, an attribute **postAttach** which refers to a Groovy closure is provided. For example:

```

builder.router{
    restlet(uri:"/users/{user}",
postAttach:{route->
    //do some post attaching process here
    //...
})
}

```

Another important attribute is **uri** which indicates a [URI Template](#) using by Restlet Router.

*restlet*

The generic constructor for all components of type of [Restlet](#). You can use **ofClass** attribute to indicate *restlet* to create an instance of a subtype. When doing that, using attribute **consArgs** to provide arguments array of corresponding constructor. For example:

```

builder.restlet(ofClass:"org.restlet.Transform
m", consArgs:[transformer.MODE_REQUEST,
sheet] as Object[])

```

Another important attribute of *restlet* is **handle**, which refers to a Groovy closure. Once this attribute provided, this closure will override the default [handle](#) method of the Restlet.

```

builder.restlet(handle:{request, response,
self->
    return new StringRepresentation("Hello
GroovyRestlet")
})

```

Sometimes it is necessary to access the instance of Restlet itself. You can do that by appending an additional argument in the closure argument list. GroovyRestlet will automatically pass the calling instance to the closure as shown in above example.

**i** This **handle** attribute is only effected when neither **ofClass** nor **ofBean** attribute is used.

### *server*

A shortcut to create a Restlet Server. Instance of the nested constructor of a `server` will become the [server's target](#). If more nested constructors are defined, only the last one will be selected.

## Attributes:

- **protocol**: the server protocol
- **protocols**: a list of server protocols
- **address**: the address of server
- **port**: the port of server

### *client*

A shortcut to create a Restlet Client.

## Attributes:

- **protocol**: the client protocol
- **protocols**: a list of client protocols

### *component*

A shortcut to create a Restlet Component. If no `VirtualHost` is specified, direct children of component constructor will be attached to Component's [default VirtualHost](#). For example,

```
builder.component{  
    application(uri: "")  
}
```

Which is equivalent to Java code:

```
Component component = new Component();  
component.getDefaultHost().attach("", new  
Application());
```

### *application*

A shortcut to create a Restlet Application. The direct child of application constructor, without attribute **autoAttach** as `false`, will be set as the [root](#) of the application instance. If more direct nested constructors are defined, only the last one will be selected.

### *router*

A shortcut to create a Restlet Router. A router constructor can have one or more nested children. Each child will be attached to the router by calling `Router.attach` method.

### *filter*

A shortcut to create a Restlet Filter.

## Attributes

- **before**: a closure to be called when `Filter.beforeHandle` is called;
- **after**: a closure to be called when `Filter.afterHandle` is called;
- **handle**: a closure to be called when `Filter.doHandle` is called;

### *redirector*

A shortcut to create a Restlet Redirector.

### *guard*

A shortcut to create a Restlet Guard.

## Attributes

- **scheme**: the challenge scheme, will use `ChallengeScheme.valueOf("None")` if no **scheme** is provided;
- **realm**: the authentication realm

### *directory*

A shortcut to create a Restlet Directory. A directory constructor can not nest child.

## Attributes:

- **root**: the root uri of the Directory

### *resource*

A shortcut to create a Restlet Resource. A resource constructor can not nest child; and it can be nested in filter and router.

## Attributes

`resource` constructor provides a set of closure attributes for easily handling RESTful calls.

- **store**: for handling **HTTP PUT** method
- **remove**: for handling **HTTP DELETE** method
- **accept**: for handling **HTTP POST** method
- **represent**: for handling **HTTP GET** method
- **head**: for handling **HTTP HEAD** method
- **options**: for handling **HTTP OPTIONS** method

These closures are only effected when no **ofClass** and **ofBean** attribute is specified.

## RestletTutorialExamples

Examples of how to construct Restlet applications mentioned in [Restlet Tutorials](#)

For example groovy scripts, see <http://svn.codehaus.org/groovy-contrib/groovyrestlet/trunk/src/test/groovy/org/lpny/groovyrestlet/examples/tutorials/>

## [Example of Part02](#)

Example about creating a [Restlet Client](#) instance.

You can use `shortcut client` constructor as shown in ``#1``; also you can use the generic `restlet` constructor to create client instance.

- ✓ you can use ``restlet(ofClass: class name)`` to create any instance of derived type of [Restlet](#). In this case, you need to specify attribute ``consArgs`` which are an array of constructing parameters.

## Create client

```
//#1 using shortcut
builder.client(protocol.HTTP)

//#2 generic restlet constructor
builder.restlet(ofClass:"org.restlet.Client"
, consArgs:[protocol.HTTP] as Object[])
```

## Example of Part03

*At current moment, server construction does not support ``restlet`` way*

Nesting here indicates a parent-child relationship.

- ✓ The child of a `server` component becoming the [target](#) property of the [Server](#).

## Create a server

```
builder.server(protocol:protocol.HTTP,port:8
182){
    restlet(handle:{req, resp->
        resp.setEntity("Hello World",
mediaType.TEXT_PLAIN)
    })
}.start()
//or you can define restlet first
def restlet = builder.restlet(handle:{req,
resp->
    resp.setEntity("Hello World",
mediaType.TEXT_PLAIN)
})
builder.server(protocol:protocol.HTTP,port:8
182, target:restlet).start()
```

### Example of Part05

## Restlet example

```
//using component shortcut constructor
def component = builder.component{
    current.servers.add(protocol.HTTP, 8182)

    restlet(uri:"/trace", handle: {req,
resp->
        println "To process request: ${req}"
        def message = """Resource URI:
${req.resourceRef}
Root URI : ${req.rootRef}
Routed part : ${req.resourceRef.baseRef}
Remaining part:
${req.resourceRef.remainingPart}
""")

        resp.setEntity(message,
mediaType.TEXT_PLAIN)
    })
}
```

### Example of Part06

## Serving static files

```
def ROOT_URI = "file:."
builder.component{
    current.servers.add(protocol.HTTP, 8182)
    current.clients.add(protocol.FILE)
    application(uri:"") {
        directory(root:ROOT_URI)
    }
}.start()
```

### Example of Part09

- ✓ Using `current` to reference to the current instance.

By default any nested component will be automatically attached to its parent component according to their parent-child relationship. Here [Guard](#) is automatically attached to its parent ([Application](#) here) as its [root](#). Adding attribute `autoAttach:false` can disable this feature.

## Guard

```
builder.component{
    current.servers.add(protocol.HTTP, 8182)
    application(uri:"") {

guard(scheme:challengeScheme.HTTP_BASIC,
realm:"Tutorial").secrets.put("scott", "tiger
".toCharArray())
        def dir =
directory(autoAttach:false, root:"")
            current.root.next=dir
    }
}.start()
```

### Example of Part10

In Restlet, an attaching operation (on Router.attach) returns an instance of [Route](#). It might be needed to do some post processing on a route. Attribute `postAttach` which refers a closure is used to support this.

## Redirector example

```
builder.component{
    current.servers.add(protocol.HTTP, 8182)
    application(uri:""){
        def router = router{
            def target =
                "http://www.google.com/search?q= {keywords}"

            redirector(uri: "/search", targetTemplate: target,
                mode: redirectorMode.MODE_CLIENT_TEMPORARY,
                postAttach: {route->
                    route.extractQuery("keywords", "kwd", true)
                })
        }
    }
}.start()
```

## Example of Part11

- You can implement the [handle method](#) of a [Restlet](#) using a groovy closure.

## Router

```
builder.component{
    current.servers.add(protocol.HTTP, 8182)
```

```
application(uri:""){
  router{
    def guard = guard(uri:"/docs",
scheme:challengeScheme.HTTP_BASIC,
realm:"Restlet Tutorials")
    guard.secrets.put("scott",
"tiger".toCharArray())
    guard.next = directory(root:"",
autoAttach:false)
    restlet(uri:"/users/ {user}",
handle:{req, resp->
    resp.setEntity("Account of
user
\"${req.attributes.get('user')}\",mediaType
.TEXT_PLAIN)
    })

restlet(uri:"/users/{user}/orders",
handle:{req, resp->
    resp.setEntity("Orders or
user
\"${req.attributes.get('user')}\",mediaType
.TEXT_PLAIN)
    })

restlet(uri:"/users/{user}/orders/{order}",
handle:{req, resp->
    def attrs = req.attributes
    def message = "Order
\"${attrs.get('order')}\", for User
```

```
\`${attrs.get('user')}\`"  
        resp.setEntity(message,  
mediaType.TEXT_PLAIN)  
    })
```

```

    }
  }
}.start()

```

## Example of Part12

Same as handle closure of a Restlet, you can implement a simple [Restlet Resource](#) using groovy closures. Following attributes are supported:

- `init`: `init` method
- `represent`: for `represent()` and `represent(Variant)` methods **HTTP GET**
- `store`: for `storeRepresentation()` method **HTTP PUT**
- `remove`: for `removeRepresentation()` method **HTTP DELETE**
- `accept`: for `acceptRepresentation()` method **HTTP POST**
- `head`: for `handleHead()` method
- `options`: for `handleOptions` method

- ✓ Specify ``self`` parameter in the last of parameter list. This special ``self`` instance indicates the resource instance.

### Resources

```

builder.component{
    current.servers.add(protocol.HTTP, 8182)

    application(uri:""){
        router{
            resource("/users/{user}",
                init:{ctx, req, resp, self->
                    self.getVariants().add(new
Variant(mediaType.TEXT_PLAIN))
                },
                represent:{variant, self->
                    return new StringRepresentation( "Account of
user \"$ {self.request.attributes.get('user')}".toString(),
                    mediaType.TEXT_PLAIN);
                })
            resource("/users/{user}/orders", ofClass:OrdersResource)
        }
    }
}.start()

```

## SpringIntegrationExamples

Example the spring integration

Integrating with Spring using Groovy-Restlet is straightforward.

## Using ofBean attribute

```
def comp = builder.component(){
    application(uri:""){
        router{
            resource(uri:"/users/{user}",
ofBean:"userResource")
        }
    }
}

comp.servers.add(protocol.HTTP, 8182)
```

The bean `userResource` is defined spring definition file as followed:

```
<bean id="userResource"
class="org.lpny.groovyrestlet.example.spring
.UserResource"
    scope="prototype" />
```

## Using ofClass attribute

```
def comp = builder.component() {
    application(uri: "") {
        router {
            resource(uri: "/orders",
ofClass: "org.lpny.groovyrestlet.example.spring.OrdersResource")
        }
    }
}

comp.servers.add(protocol.HTTP, 8182)
```

If spring context is specified when creating `GroovyRestlet` instance, Spring's [AutowireCapableBeanFactory](#) will be used to create that instance and autowire all necessary properties.

## Groovy Science

### Module Overview

Groovy Science is a symbolic manipulation library for Groovy that is intended to be easy to "glue" to existing scientific Java (and Groovy) libraries.

### Installing

There are no archive releases of Groovy Science yet, but the current source can be found at <http://svn.codehaus.org/groovy-contrib/science/>. To use it, you can do any of the following things:

- Build it in its own project, and have your project reference that project.
- Copy the source into your own project.
- Make a .jar file yourself, and use that.

### Pre-requisites

Groovy Science has been successfully built and used under Java 1.6.0 Update 7 and Groovy 1.5.1.

## Documentation

### Building SymbolicExpressions

The centerpiece of the library is the `SymbolicExpression` class. A `SymbolicExpression` is a representation of the "application" of an operator object to a list of other `SymbolicExpressions`. This makes for a simple tree structure, and it is not unlike the way Lisp code is represented in s-expressions.

```
import
org.codehaus.groovy.science.SymbolicExpression
import static
org.codehaus.groovy.science.SymbolicExpression.expr

Object plusOp = new Object();
Object leafOp = new Object();
SymbolicExpression leaf = expr( leafOp );
SymbolicExpression myExpression = expr(
plusOp, leaf, leaf );

assert myExpression.operator == plusOp;
assert myExpression.argumentList == [ leaf,
leaf ];

assert myExpression.argumentList[ 0
].operator == leafOp;
assert myExpression.argumentList[ 0
].argumentList == [];
```

The SymbolicExpression class overloads almost all of the operators that can be overloaded in Groovy. So, instead of building all expressions using `expr`, you can sometimes take advantage of Groovy's own syntax:

```
import
org.codehaus.groovy.science.SymbolicExpressi
on
import static
org.codehaus.groovy.science.SymbolicExpressi
on.expr
import
org.codehaus.groovy.science.OverloadableOper
ators

Object leafOp = new Object();
SymbolicExpression leaf = expr( leafOp );

assert leaf + leaf == expr(
OverloadableOperators.Plus, leaf, leaf );
assert leaf[ leaf ] == expr(
OverloadableOperators.GetAt, leaf, leaf );
```

If you wanted to represent an expression like "1 + 1", you could do so as follows:

```
import
org.codehaus.groovy.science.SymbolicExpressi
on
import static
org.codehaus.groovy.science.SymbolicExpressi
on.expr

SymbolicExpression one = expr( 1 );
SymbolicExpression onePlusOne = one + one;
```

If you do that, though, you might run the risk of confusing your constants with your other operators. To help keep your constants clearly identified, you can use the ConstantOperator class:

```
import
org.codehaus.groovy.science.SymbolicExpression
on
import static
org.codehaus.groovy.science.SymbolicExpression
on.expr
import
org.codehaus.groovy.science.ConstantOperator

import static
org.codehaus.groovy.science.ConstantOperator
.*
    // for con, unCon, and isCon

SymbolicExpression one = con( 1 );
SymbolicExpression onePlusOne = one + one;

assert one == expr( new ConstantOperator( 1
) );
assert one.operator.value == 1;
assert unCon( one ) == 1;
assert isCon( one );
assert !isCon( onePlusOne );
```

#### Developers

- Ross Angle [rokitna at hotmail]

#### Source Control

<http://svn.codehaus.org/groovy-contrib/science/>

## Contributing

Please contact the team members by e-mail.

Groovy Science can take a lot of different directions. Here are some of the bigger sub-projects it might encompass:

- A toolkit of wrappers that make it easier to use SymbolicExpressions with a particular scientific Java library.
- A comprehensive or near-comprehensive palette of operators for use with a particular domain, such as expressions involving real numbers, assertions in set theory, or chemical reactions.
- A new, expressive way to generate SymbolicExpressions "literally" in code, such as by parsing a string or by using AST macros to interpret actual Groovy source code.
- A new way of visualizing SymbolicExpressions, such as a plain text, TeX, or HTML prettyprinter---or even a way to more easily create such prettyprinters.

In case you don't have a big idea like that or a lot of time to do it in, here are some of the more menial things that could still really help:

- Write a build script.
- Generate the Javadocs. (Javadoc comments are already maintained, but actual HTML hasn't been generated yet.)
- Write more test cases.
- Identify code that is in severe need of commenting. If you can, comment it yourself.

## Groovy SOAP

### **Deprecated Module**

Before using GroovySOAP, make sure to check [GroovyWS](#)

## Introduction

[SOAP](#) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. Groovy has a SOAP implementation based on [Xfire](#) which allows you to create a SOAP server and/or make calls to remote SOAP servers using Groovy.

## Installation

You just need to download [this](#) jar file in your `$(user.home)/.groovy/lib` directory. This jar file embeds all the dependencies.

## Getting Started

### The Server

You can develop your web service using a groovy script and/or a groovy class. The following two groovy files are valid for building a web-service.

1. MathService.groovy

```
public class MathService {
    double add(double arg0, double arg1) {
        return (arg0 + arg1)
    }
    double square(double arg0) {
        return (arg0 * arg0)
    }
}
```

## 2. You can also using something more Groovy

```
double add(double arg0, double arg1) {
    return (arg0 + arg1)
}
double square(double arg0) {
    return (arg0 * arg0)
}
```

## 3. Then the easy part ... no need for comments

```
import groovy.net.soap.SoapServer

def server = new SoapServer("localhost", 6980)

server.setNode("MathService")

server.start()
```

That's all !

## The Client

### 1. Oh ... you want to test it ... two more lines.

```
import groovy.net.soap.SoapClient

def proxy = new
SoapClient("http://localhost:6980/MathServiceInterface?wsdl")

def result = proxy.add(1.0, 2.0)
assert (result == 3.0)

result = proxy.square(3.0)
assert (result == 9.0)
```

2. You're done!

## Custom Data Types

This example shows how to use custom data types with Groovy SOAP. The code can be downloaded from [here](#).

### The Server

The `PersonService.groovy` script contains the service implementation and the custom data type (`Person`).

#### PersonService.groovy

```
class Person {
    int id
    String firstname
    String lastname
}

Person findPerson(int id) {
    return new Person(id:id,
        firstname:'First', lastname:'Last')
```

`Server.groovy` is equivalent to the previous example.

#### Server.groovy

```
import groovy.net.soap.SoapServer;

def server = new SoapServer("localhost",
    6980);
server.setNode("PersonService");
server.start();
```

For each class compiled by the groovy compiler a `metaClass` property is added to the bytecode. This property

must be excluded from being mapped by XFire, otherwise an error will be reported when trying to obtain the WSDL document from <http://localhost:6980/PersonServiceInterface?wsdl>. The reason is that XFire cannot map `groovy.lang.MetaClass`. To ignore the `metaClass` property a custom type mapping must be defined (for details refer to [Aegis Binding](#)).

### Person.aegis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings
xmlns:sample="http://DefaultNamespace">
  <mapping name="sample:Person">
    <property name="metaClass"
ignore="true" />
  </mapping>
</mappings>
```

However, if you compile custom data types from Java the bytecode won't contain a `metaClass` property and, hence, there is no need to define a custom mapping.

### The Client

### Client.groovy

```
import groovy.net.soap.SoapClient

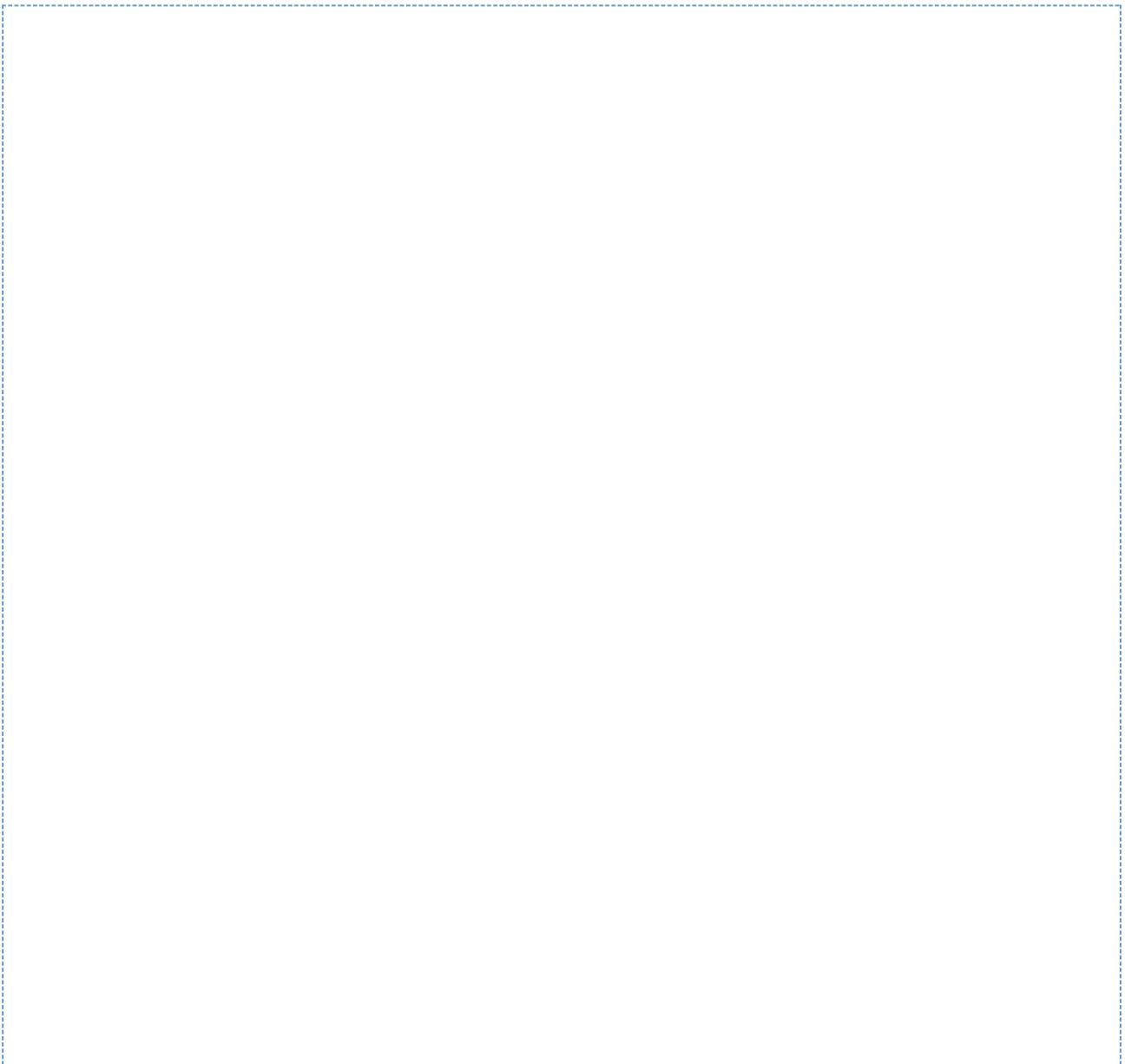
def proxy = new
SoapClient('http://localhost:6980/PersonServiceInterface?wsdl')
def person = proxy.findPerson(12)
println 'Person (' + person.id + ') = ' +
  person.firstname + ' ' + person.lastname
```

## More Information

### Current limitations (and workaround)

1. No authentication (see JIRA issue 1457)
2. No proxy support (see JIRA issue 1458)
3. Numeric values are represented as strings in custom data types and arrays.
4. Custom data types cannot be processed on client side when using the Groovy SOAP module with the current groovy-1.0 release.
5. It looks like the XFire dynamic client does not support complex datatypes. This may be a concern if you need for example to transfer an Image as a byte array. The workaround I use is to transform this in a String and transfer that String - As this is a bit painful I am investigating moving to the regular XFire client. Here is a little program demo-ing this (look at this "disco age" image - Is Groovy that old ?

The client (ImageClient.groovy)



```
import groovy.swing.SwingBuilder
import groovy.net.soap.SoapClient
import javax.swing.ImageIcon
import
org.apache.commons.codec.binary.Base64

proxy = new
SoapClient("http://localhost:6980/ImageServiceInterface?WSDL")

// get the string, transform it to a byte
array and decode this array
b64 = new Base64()
bbytes =
b64.decode(proxy.getImage().getBytes())

swing = new groovy.swing.SwingBuilder()

// this is regular SwingBuilder stuff
i1 = swing.label(icon:new ImageIcon(bbytes))
frame = swing.frame(title:'Groovy logo',
defaultCloseOperation:javax.swing.WindowConstants.DISPOSE_ON_CLOSE) {
    panel(){
        widget(i1)
    }
}
frame.pack()
frame.show()
```

The (ugly) server part embedding the image which is Base64 encoded (ImageServer.groovy):

```
import groovy.net.soap.SoapServer  
  
def server = new SoapServer("localhost",  
6980)  
server.setNode("ImageService")  
server.start()
```

and the missing and secret part is [here](#).

#### Demos with public web services

There exist a lot of web-services available for testing. One which is pretty easy to evaluate is the currency rate calculator from webservicex.net.

Here is a small swing sample that demonstrate the use of the service. Enjoy !

```
import groovy.swing.SwingBuilder  
import groovy.net.soap.SoapClient  
  
proxy = new  
SoapClient("http://www.webservicex.net/Curre  
ncyConvertor.asmx?WSDL")  
  
def currency = ['USD', 'EUR', 'CAD', 'GBP',  
'AUD']  
def rate = 0.0  
  
swing = new SwingBuilder()  
  
refresh = swing.action(  
name: 'Refresh',
```

```
closure:this.&refreshText,  
mnemonic:'R'  
)  
  
frame = swing.frame(title:'Currency Demo') {  
  panel {  
    label 'Currency rate from '  
    comboBox(id:'from', items:currency)  
    label ' to '  
    comboBox(id:'to', items:currency)  
    label ' is '  
    textField(id:'currency', columns:10,  
rate.toString())  
    button(text:'Go !', action:refresh)  
  }  
}  
frame.pack()  
frame.show()  
  
def refreshText(event) {  
  rate =  
proxy.ConversionRate(swing.from.getSelectedI
```

```
tem(), swing.to.getSelectedItem())
    swing.currency.text = rate
}
```

And here is the result:



## GroovySWT

GroovySWT is a wrapper around SWT, the eclipse Standard Widget Toolkit. It allows you to easily write Eclipse [SWT](#) applications by using Groovy's *builder* mechanism.

Here is some SWT code using native Groovy:

```
import org.eclipse.swt.SWT
import org.eclipse.swt.widgets.*
import org.eclipse.swt.layout.RowLayout as
Layout

def display = new Display()
def shell = new Shell(display)

shell.layout = new Layout(SWT.VERTICAL)

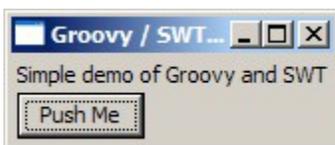
shell.text = 'Groovy / SWT Test'

def label = new Label(shell, SWT.NONE)
label.text = 'Simple demo of Groovy and SWT'
shell.defaultButton = new Button(shell,
SWT.PUSH)
shell.defaultButton.text = '  Push Me  '

shell.pack()
shell.open()

while (!shell.disposed) {
    if (!shell.display.readAndDispatch())
shell.display.sleep()
}
```

When you run this (see below for setup details), the result looks like:



Here is the same example using SwtBuilder - note that this example does more because it actually prints some text 'Hello' to standard output when you press the button:

```
import org.eclipse.swt.SWT
import org.eclipse.swt.widgets.*

def swt = new groovy.swt.SwtBuilder()
def shell = swt.shell(text: 'Groovy / SWT
Test') {
    rowLayout()
    label('Simple demo of Groovy and SWT')
    button(' Push Me ') {
        onEvent(type: 'Selection') { println
"Hello" }
    }
}

shell.doMainloop()
```

### *Examples*

For a short example of using the JFace builder for creating a standard application see the [TextEditor sample](#). You can find about 70 examples in the example [directory in subversion](#) or you can download them [here](#).

For a comparison of the groovy and java code see [here](#) or see the latest code in subversion.

### *Platform dependency.*

Groovy-swt is not platform dependent, but SWT is. So you need to make sure that you are running a SWT matching the platform you are running the application on. You can download the appropriate SWT version at [eclipse](#) (select the eclipse version, find the heading "SWT Binary and Source", and download the SWT for your platform)

### *Releases:*

See the [changelog](#) for changes.

0.5:

Updated to eclipse 3.5 (see the changelog for all the other changes)

- groovy-swt-0.5 with all eclipse libraries and the windows version of SWT: [groovy-swt-0.5-windows-all.jar](#)

- groovy-swt-0.5 with all eclipse libraries but no SWT: [groovy-swt-0.5-without-swt.jar](#)
- groovy-swt-0.5 examples: [groovy-swt-0.5-examples.zip](#)

Please note the 0.5 was completely re-written to use FactoryBuilderSupport (to be able to be included in griffon), so there may be a few inconsistencies with version 0.3, but if you find any then please let me know. You can see the [changelog](#) for a list of all the changes.

Another major new feature in version 0.5 is the binding framework (build on top of the jface databinding and following the same style as the swing builder). You can find a little documentation with examples [here](#).

0.3:

The sources (and README.txt telling you how to set up the libraries and dll's) can be found in subversion:

- [groovy-swt](#)

The jar file compiled against Eclipse SDK 3.3 jars can be found attached:

- [groovy-swt-0.3.jar](#)

To run groovy-swt application you also need some of the eclipse libraries (including SWT), so download [groovy-swt-0.3-including-Eclipse-libs.ZIP](#) and unzip all the jar files into the lib directory of your groovy installation. This zip file includes the windows version of SWT, so if you are on another platform you need to download the appropriate version of SWT at [eclipse](#) and include that instead.

## A sample application

I have added a couple of applications in the example directory that can be used as a template for building other applications.

Here is an extract from a basic TextEditor that shows how to use the JFace builder:

```
mainapp = jface.applicationWindow(
title:"Groovy Text Editor",
location:[100,100], size:[500, 300] ) {
    gridLayout(numColumns:1)
    menuManager( text:"File" ) {
        action ( text:"Exit",
closure:{ mainapp.close() } )
    }

    menuManager( text:"Edit" ) {
        action ( text:"Cut",
accelerator: SWT.MOD1 + (int) 'X', closure:{
handleCutCopy(); text.cut() } )
    }
}
```

```

        action ( text:"Copy",
accelerator: SWT.MOD1 + (int) 'C',closure:{
handleCutCopy(); text.copy() } )
        action ( text:"Paste",
accelerator: SWT.MOD1 + (int) 'P',closure:{
text.paste() } )
        separator()
        action ( text:"Set Font",
closure:{ setFont() } )
    }
    toolBar( style:"none" ) {
        boldButton =
toolItem(style:"check", toolTipText:"Bold")
{
        image(
src:"src/examples/groovy/swt/examples/TextEd
itor/bold.png" )

onEvent(type:"Selection",
closure:{setStyle(it.widget)})
        }
        italicButton =
toolItem(style:"check",
toolTipText:"Italic") {
        image(
src:"src/examples/groovy/swt/examples/TextEd
itor/italic.png" )

onEvent(type:"Selection",
closure:{setStyle(it.widget)})
        }
    }
}

```

```
        underlineButton =
toolItem(style:"check",
toolTipText:"Underline") {
        image(
src:"src/examples/groovy/swt/examples/TextEd
itor/underline.png" )

onEvent(type:"Selection",
closure:{setStyle(it.widget)})
        }
        strikethroughButton =
toolItem(style:"check",
toolTipText:"Strikethrough") {
        image(
src:"src/examples/groovy/swt/examples/TextEd
itor/strikethrough.png" )

onEvent(type:"Selection",
closure:{setStyle(it.widget)})
        }
        separator =
toolItem(style:"separator")
        toolItem(style:"push",
toolTipText:"Red text") {
        image(
src:"src/examples/groovy/swt/examples/TextEd
itor/red.png" )

onEvent(type:"Selection",
closure:{fgColor(RED)})
        }
        toolItem(style:"push",
```

```
toolTipText:"Blue text") {
            image(
src:"src/examples/groovy/swt/examples/TextEd
itor/blue.png" )

onEvent(type:"Selection",
closure:{fgColor(BLUE)})
        }
        toolItem(style:"push",
toolTipText:"Green text") {
            image(
src:"src/examples/groovy/swt/examples/TextEd
itor/green.png" )

onEvent(type:"Selection",
closure:{fgColor(GREEN)})
        }
        toolItem(style:"separator")
        toolItem(style:"push",
toolTipText:"Clear formatting") {
            image(
src:"src/examples/groovy/swt/examples/TextEd
itor/erase.png" )

onEvent(type:"Selection", closure:{clear()})
        }
    }

    text = styledText ( style:
"Border, Multi, V_Scroll, H_Scroll") {
```

```
gridData(horizontalAlignment: GridData.FILL,  
          verticalAlignment:  
GridData.FILL,  
  
grabExcessHorizontalSpace: true,  
  
grabExcessVerticalSpace: true  
          )  
    }  
  
text.addExtendedModifyListener(this)  
    }
```

Simple isn't it? 😊

For the complete code see [subversion](#) or the attached snapshot.

## Binding

The groovy-swt binding framework is inspired by the swing binding framework, but is based on the [jface databinding](#). It is not meant to be a replacement of the jface databinding, and does not include all the features of the jface databinding. Think of it as an easy shortcut covering most cases, so you may still have to revert to the jface databinding framework.

### *An Appetizer*

Here is just a simple example from the included [examples](#):

```
person = new Person01(firstName: 'John',  
lastName: 'Doo', gender: 'Male', age: 12,  
married: true)  
.....  
label('Firstname: ' )  
text(text: bind(model: person,  
modelProperty: 'firstName' ))  
label('Age: ' )  
text(text: bind(model: person,  
modelProperty: 'age' ))  
label('Married: ' )  
button(style: 'CHECK',  
selection: bind(model: person,  
modelProperty: 'married' ) )
```

This will give you automatic two-way binding:

- if you change the model (person) it will automatically be reflected in the GUI (the text and button widgets)
- if the user change the text widgets it will automatically be updated in the model.

### *The bind object*

Properties:

- model: the model object you are binding to.
- modelProperty: the model property you are binding to.
- childrenProperty: Used in TreeViewers to specify the children property of the model object.
- target: the widget you are binding to.
- targetProperty: the widget property you are binding to.
- model2target: a [UpdateValueStrategy](#) allowing you to specify validators, converter, etc.
- target2model: a [UpdateValueStrategy](#) allowing you to specify validators, converter, etc.
- closure: a simple way to create ComputedValues.

#### Builder styles

There are (at least) two ways you can write your bindings using the groovy-swt builder:

The basic using a bind node:

```
text(' ', id:firstNameTextWidget)
bind(target: firstNameTextWidget,
targetProperty: 'text', model: person,
modelProperty: 'firstName')
```

or the shortcut to avoid all the typing:

```
text(text:bind(model:person,
modelProperty:'firstName'))
```

If you find that the binding functionality provided in groovy-swt is not enough or doesn't fit your needs you can use create your own IObservable using the jface databinding framework directly and then provide that as the model object, like:

```
def observable =
BeansObservables.observeValue(person,
"firstName")
text(text:bind(model:observable)
```

#### Viewers

If you want to bind the data to jface viewers (lists, combos, tables, etc) you should bind the input attribute like:

```

def people = new WritableList(Realm.default)
people.add(new Person02(name:"Wile E.
Coyote", city:"Tucson"))
people.add(new Person02(name:"Road Runner",
city:"Lost Horse"))
.....
list() {
    listViewer(input:
bind(model:viewModel.people,
modelProperty:'name'))
}

```

If you want the viewer to reflect changes to the list it needs to be a list that send notification if the list changes, so use the jface-databinding class `WritableList`. Groovy-swt accepts ordinary groovy lists, but then it will be static data and the viewer will not reflect changes to the list:

```

class ViewModel {
def cities = ['NY', 'LA', 'NB']
}

....
combo(style:'READ_ONLY') {
    comboViewer(input: bind(model:viewModel,
modelProperty:'cities'))
}

```

If you use a jface viewer as a model it will create a master-detail observable of the selected item:

```
list() {
    listViewer(id: 'v1', input: bind(people,
modelProperty: 'name'))
}
// doing it manually:
text(text:
bind(BeansObservables.observeDetailValue(ViewerProperties.singleSelection().observe(v1),
'city', String.class)))
// shortcut
text(text: bind(model: v1,
modelProperty: 'city'))
```

### *ComputedValues*

To make simple one-way binding (possibly with some kind of calculations) you can use a bind closure:

```
text(text: bind {viewModel.text })
```

Groovy-swt will (like the Swing builder) automatically find out which properties are used in the closure and create a ComputedValue binding for the closure with bindings to the properties used in the closure.

Or with a more complex example:

```
text(text:bind(model:viewModel,
modelProperty:'text'))
// NOTE: Bind can not find the properties to
bind to if they are hidden inside a GString.
label(text:bind {"You have written
"+viewModel.text?.size()+ " characters"})
label('Characters left: ')
label(text:bind{(50-viewModel.text?.size()).
toString()}))
```

You can also use the `model2target.converter` to create a simple calculated value:

```
button('Remove', enabled: bind(model:
peopleViewer, modelProperty:'name',
'model2target.converter':
{t1.selectionCount>0}))
```

#### Threads

TODO: Realms, updating models in the realm,

#### Changelog

#### Changelog:

TODOS:

- Default table with headerVisible, showLines, etc.
- Support for NatTable
- Support for glazedLists

? In SwtBuilder eg. the button can have children (like onEvent, gridLayout, image).  
In SwingBuilder they can not: everything is attributes. Should this be changed?

? Any need for CustomSwingBuilder?

? TableTree, TableTreeItem are deprecated - should they be removed?

**0.5:**

- Updated to Eclipse 3.5 libraries.

**0.4.2:**

- bind with closures. If you bind with a closure it is like a ComputedValue which only gives you one-way binding.
- menu as a subnode to menuitem should automatically set the menuitem.menu
- Added a Bindable and WritableList that can be used from several threads but only fires events in the right realm.
- Using system colors (foreground: 'BLUE') for color attributes.
- Change the license to Apache as the rest of groovy.
- Upgraded to Eclipse 3.5M7

**0.4.1:**

- Added initial databinding support
- Shortened the onEvent(). Instead of

```
onEvent(type: 'Selection', closure: {...})
```

you can use

```
onEvent('Selection') {...}
```

- Added StackLayout
- Added InputDialog
- Added ErrorDialog
- Added MessageDialog
- Added MessageDialogWithToggle
- Added an id attribute
- Changed list() to use default styles: SWT.V\_SCROLL | SWT.BORDER
- Added a subclass of the Shell and added a method doMainloop() to do all the repetitive:

```
pack();
open();
while (!isDisposed()) {
    if (!getDisplay().readAndDispatch()) {
        getDisplay().sleep();
    }
}
```

**0.4.0:**

- Updated gmaven
- Updated to Eclipse 3.4
- Changed swt/jface builders to groovy
- Added the id attribute
- Change builders to support FactoryBuilderSupport to be able to use SWT in Griffon.
- Deleted the ApplicationGuiBuilder in groovy.swt.guibuilder. This was marked as "WORK IN PROGRESS, don't use this one yet" in 2004, so I don't suppose anyone was using it.
- Delete the get/setCurrent methods from the builders. Was only used by the GuiBuilders.
- Fixed a bug:

fillLayout ( type:"vertical" )  
never worked. Strings are now matched against SWT constants if relevant.

- Included the MigLayout in the SwtBuilder
- Make 'text' default attribute if a text is given without attribute. You can now use

label('label text')  
instead of  
label(text:'label text')

- Using SwingBuilder inside a SWT application is working again.

### 0.3

- updated to maven2
- added automatic download of eclipse libraries
- added support for RadioGroupFieldEditor (used in Preferences)
- added support for StyledText
- Fixed a bug in WizardDialog
- swt.Dialog is abstract
- Fixed a bug using cascading MenuManagers

## groovy-swt - Comparing the java and the groovy code

To show how to use the the JFace builder I converted a couple of applications from the standard SWT example set:

- TextEditor: A simple application showing how to use menubar, toolbar and a StyledText.
- AddressBook: A more complete application with a simple addressbook. Shows how to use toolbars, tables, sorting, Search dialogs, editing, copy/paste

I have on purpose changed the application code as little as possible (only to make work in Groovy), and with the main focus to change the creation of the GUI to show how to use the JFace builder. But it also shows how much easier it is to use the JFace builder.

Here is the extract from the groovy code (61 lines):

```
mainapp = jface.applicationWindow(  
title:'Groovy Text Editor',
```

```

location:[100,100], size:[500, 300] ) {
    gridLayout(numColumns:1)
    menuManager( 'File' ) {
        action ( 'Exit', closure:{
mainapp.close() } )
    }

    menuManager( 'Edit' ) {
        action ( 'Cut', accelerator:
'Ctrl+X', closure:{ handleCutCopy();
text.cut() } )
        action ( 'Copy', accelerator:
'Ctrl+C',closure:{ handleCutCopy();
text.copy() } )
        action ( 'Paste', accelerator:
'Ctrl+P',closure:{ text.paste() } )
        separator()
        action ( 'Set Font', closure:{
setFont() } )
    }
    toolBar( style:'none' ) {
        boldButton =
toolItem(style:'check', toolTipText:'Bold')
{
        image( src:'bold.png' )
        onEvent('Selection')
{setStyle(it.widget)}
        }
        italicButton =
toolItem(style:'check',
toolTipText:'Italic') {

```

```

        image( src:'italic.png' )
        onEvent( 'Selection' )
{setStyle(it.widget)}
    }
    underlineButton =
toolItem(style:'check',
toolTipText:'Underline' ) {
    image( src:'underline.png' )

onEvent( 'Selection' ){setStyle(it.widget)}
    }
    strikethroughButton =
toolItem(style:'check',
toolTipText:'Strikethrough' ) {
    image( src:'strikethrough.png' )
    onEvent( 'Selection' )
{setStyle(it.widget)}
    }
    separator =
toolItem(style:'separator')
    toolItem(style:'push',
toolTipText:'Red text' ) {
    image( src:'red.png' )
    onEvent( 'Selection' )
{fgColor(RED)}
    }
    toolItem(style:'push',
toolTipText:'Blue text' ) {
    image( src:'blue.png' )

onEvent( 'Selection' ){fgColor(BLUE)}
    }

```

```
        toItem(style:'push',
toolTipText:'Green text') {
            image( src:'green.png' )
onEvent('Selection'){fgColor(GREEN)}
        }
        toItem(style:'separator')
        toItem(style:'push',
toolTipText:'Clear formatting') {
            image( src:'erase.png' )
            onEvent('Selection') {clear()}
        }
    }

    text = styledText ( style: 'Border,
Multi, V_Scroll, H_Scroll') {
        gridData(horizontalAlignment:
GridData.FILL,
            verticalAlignment: GridData.FILL,
            grabExcessHorizontalSpace: true,
            grabExcessVerticalSpace: true
        )
    }

    text.addExtendedModifyListener(this)
}
```

```
mainapp.open()
```

And here is the extract from the java code (175 lines)

```
Menu createEditMenu() {
    Menu bar = shell.getMenuBar ();
    Menu menu = new Menu (bar);

    MenuItem item = new MenuItem (menu,
SWT.PUSH);
    item.setText
(resources.getString("Cut_menuitem"));
    item.setAccelerator(SWT.MOD1 + 'X');
    item.addListener(new
SelectionAdapter() {
        public void
widgetSelected(SelectionEvent event) {
            handleCutCopy();
            text.cut();
        }
    });
    item = new MenuItem (menu, SWT.PUSH);
    item.setText
(resources.getString("Copy_menuitem"));
    item.setAccelerator(SWT.MOD1 + 'C');
    item.addListener(new
SelectionAdapter() {
        public void
widgetSelected(SelectionEvent event) {
            handleCutCopy();
```

```

        text.copy();
    }
});
item = new MenuItem (menu, SWT.PUSH);
item.setText
(resources.getString("Paste_menuitem"));
item.setAccelerator(SWT.MOD1 + 'V');
item.addSelectionListener(new
SelectionAdapter() {
    public void
widgetSelected(SelectionEvent event) {
        text.paste();
    }
});
new MenuItem (menu, SWT.SEPARATOR);
item = new MenuItem (menu, SWT.PUSH);
item.setText
(resources.getString("Font_menuitem"));
item.addSelectionListener(new
SelectionAdapter() {
    public void
widgetSelected(SelectionEvent event) {
        setFont();
    }
});
return menu;
}

Menu createFileMenu() {
    Menu bar = shell.getMenuBar ();
    Menu menu = new Menu (bar);

```

```

        MenuItem item = new MenuItem (menu,
SWT.PUSH);
        item.setText
(resources.getString("Exit_menuitem"));
        item.addSelectionListener(new
SelectionAdapter() {
            public void
widgetSelected(SelectionEvent event) {
                shell.close ();
            }
        });

        return menu;
    }

void createMenuBar () {
    Menu bar = new Menu (shell, SWT.BAR);
    shell.setMenuBar (bar);

    MenuItem fileItem = new MenuItem (bar,
SWT.CASCADE);
    fileItem.setText
(resources.getString("File_menuitem"));
    fileItem.setMenu (createFileMenu ());

    MenuItem editItem = new MenuItem (bar,
SWT.CASCADE);
    editItem.setText
(resources.getString("Edit_menuitem"));
    editItem.setMenu (createEditMenu ());

```

```

}

void createShell (Display display) {
    shell = new Shell (display);
    shell.setText
(resources.getString("Window_title"));
    images.loadAll (display);
    GridLayout layout = new GridLayout();
    layout.numColumns = 1;
    shell.setLayout(layout);
    shell.addDisposeListener (new
DisposeListener () {
        public void widgetDisposed
(DisposeEvent e) {
            if (font != null)
font.dispose();
            images.freeAll ();
            RED.dispose();
            GREEN.dispose();
            BLUE.dispose();
        }
    });
}

void createStyledText() {
    initializeColors();
    text = new StyledText (shell, SWT.BORDER
| SWT.MULTI | SWT.V_SCROLL | SWT.H_SCROLL);
    GridData spec = new GridData();
    spec.horizontalAlignment =
GridData.FILL;
    spec.grabExcessHorizontalSpace = true;
}

```

```
spec.verticalAlignment = GridData.FILL;
spec.grabExcessVerticalSpace = true;
text.setLayoutData(spec);
text.addExtendedModifyListener(new
ExtendedModifyListener() {
    public void
modifyText(ExtendedModifyEvent e) {
        handleExtendedModify(e);
    }
});
}

void createToolBar() {
    toolBar = new ToolBar(shell, SWT.NONE);
    SelectionAdapter listener = new
SelectionAdapter() {
        public void
widgetSelected(SelectionEvent event) {
            setStyle (event.widget);
        }
    };
    boldButton = new ToolItem(toolBar,
SWT.CHECK);
    boldButton.setImage(images.Bold);

boldButton.setToolTipText(resources.getStrin
g("Bold"));

boldButton.addSelectionListener(listener);
    italicButton = new ToolItem(toolBar,
SWT.CHECK);
```

```
        italicButton.setImage(images.Italic);

italicButton.setToolTipText(resources.getString("Italic"));

italicButton.addSelectionListener(listener);
        underlineButton = new ToolItem(toolBar,
SWT.CHECK);

underlineButton.setImage(images.Underline);

underlineButton.setToolTipText(resources.getString("Underline"));

underlineButton.addSelectionListener(listener);
        strikeoutButton = new ToolItem(toolBar,
SWT.CHECK);

strikeoutButton.setImage(images.Strikeout);

strikeoutButton.setToolTipText(resources.getString("Strikeout"));

strikeoutButton.addSelectionListener(listener);

        ToolItem item = new ToolItem(toolBar,
SWT.SEPARATOR);
        item = new ToolItem(toolBar, SWT.PUSH);
        item.setImage(images.Red);
```

```
        item.addSelectionListener(new
SelectionAdapter() {
            public void
widgetSelected(SelectionEvent event) {
                fgColor(RED);
            }
        });
        item = new ToolItem(toolBar, SWT.PUSH);
        item.setImage(images.Green);
        item.addSelectionListener(new
SelectionAdapter() {
            public void
widgetSelected(SelectionEvent event) {
                fgColor(GREEN);
            }
        });
        item = new ToolItem(toolBar, SWT.PUSH);
        item.setImage(images.Blue);
        item.addSelectionListener(new
SelectionAdapter() {
            public void
widgetSelected(SelectionEvent event) {
                fgColor(BLUE);
            }
        });
        item = new ToolItem(toolBar,
SWT.SEPARATOR);
        item = new ToolItem(toolBar, SWT.PUSH);
        item.setImage(images.Erase);
        item.addSelectionListener(new
SelectionAdapter() {
```

```
        public void
widgetSelected(SelectionEvent event) {
            clear();
        }
    });
}

public static void main (String [] args) {
    Display display = new Display ();
    TextEditor example = new TextEditor ();
    Shell shell = example.open (display);
    while (!shell.isDisposed ())
        if (!display.readAndDispatch ())
display.sleep ();
    display.dispose ();
}

public Shell open (Display display) {
    createShell (display);
    createMenuBar ();
    createToolBar ();
    createStyledText ();
    shell.setSize(500, 300);
    shell.open ();
}
```

```
    return shell;
}
```

So basically the groovy code is about 1/3 of the java code and a lot more readable.

For the complete code see the attachment or [subversion](#).

## GroovyWS

### Be Careful

Due to low availability the project is currently dormant, you might consider [groovy-wslite](#) as a replacement module

## Module Overview

If you need to quickly consume and/or publish WS-I compliant [web services](#), GroovyWS can help you.

## Installation

If you are online and using the [latest groovy](#) version, here is the no-brainer way to use GroovyWS:

```
@Grab(group='org.codehaus.groovy.modules',
module='groovyws', version='0.5.2')
import groovyx.net.ws.WSClient

proxy = new
WSClient("http://www.w3schools.com/webservices/tempconvert.asmx?WSDL",
this.class.classLoader)
proxy.initialize()

result = proxy.CelsiusToFahrenheit(0)
println "You are probably freezing at
${result} degrees Fahrenheit"
```

 Make sure to have the correct [GroovyWS grape config file](#)

 Make sure that you have javac in your path - this is required for automatic generation of the classes on the client side

⚠ Groovy 1.6.3 relies on Ivy 2.0 which has problem computing some dependencies checksums - if you experience this problem, you have to either download a Groovy snapshot or upgrade Ivy to version 2.1.0 (or above) in your \$GROOVY\_HOME directory.

If you need more control, see the [GroovyWS installation](#) notes.

## Getting Started

GroovyWS comes with two sets of APIS that are briefly described below using a simple example.

- [Publishing a web-service](#)
- [Consuming a web service](#)

When your service is using groovy beans on the server side, you may want to control the fields that are serialized. This is done using a small xml file located next to your script. A small example is demonstrating this:

- [Using the Aegis mapping](#)

When consuming a web service, you may also be using some complex types. Those types are automatically generated from the WSDL, compiled and made available via your classloader. The client API is providing you a method to easily instantiate such a complex object from its name. Obviously, knowing the class name can be difficult when using a complex web service and may require to study the contract (WSDL). In order to help the user, GroovyWS is logging the names of the classes generated on the fly.

- [Using the client with complex objects](#)

The client side integrates seamlessly with [Grails](#) applications.

- [Using WSCClient in Grails](#)
- [Using the Grails XFire plugin and GroovyWS](#)
- [Using the Grails Axis2 plugin and GroovyWS](#)

You can also use secured web-services with GroovyWS. If you wish to do so check [Using WS-Security](#).

There exists a lot of public web services. We provide two examples that show how easy it is to use GroovyWS to tap on these resources.

- [Currency rate calculator](#)
- [TerraServer-USA by Microsoft](#)

More [WSCClient configuration](#) is available if you need to use proxies, basic authentication and security related features.

The [javadoc](#) is probably the ultimate place to the missing bits. If you are missing a feature email us [user@groovy.co.dehaus.org](mailto:user@groovy.co.dehaus.org).

CXF is generating tons of logging information. If you want to reduce the amount of information, you need to set the - **Djava.util.logging.config.file** property.

## Articles

1. A nice article from Geertjan's blog with several examples: [http://blogs.sun.com/geertjan/entry/groovy\\_web\\_service](http://blogs.sun.com/geertjan/entry/groovy_web_service)
2. An article explaining the difference between the different WSDL styles <http://www.ibm.com/developerworks/watson/services/library/ws-whichwsdl/>

## They use GroovyWS

If you use GroovyWS, please let us know and feel free too add a quote in this section.

## Community

- [Source repository](#)
- The build process is using [gradle](#). In order to build the sources from svn, just run:

```
./gradlew
```

in the directory containing the source tree.

- Feel free to contribute by testing, giving your feedback, [reporting bugs](#) and sending patches.

## Consuming a web service

### *The Client*

Oh ... you want to test the previously defined web service [Publishing a web service](#)

```
import groovyx.net.ws.WSClient  
  
def proxy = new  
WSClient("http://localhost:6980/MathService?  
wsdl", this.class.classLoader)  
proxy.initialize() // from 0.5.0  
def result = proxy.add(1.0 as double, 2.0 as  
double)  
assert (result == 3.0)  
  
result = proxy.square(3.0 as double)  
assert (result == 9.0)
```

You're done!

### **Publishing a web service**

#### **Currency rate calculator**

One which is pretty easy to evaluate is the currency rate calculator from [webservicex.net](#). Here is a small swing sample that demonstrate the use of the service. Enjoy !

```
import groovy.swing.SwingBuilder
import groovyx.net.ws.WSClient

proxy = new
WSClient("http://www.websvcicex.net/Currenc
yConvertor.asmx?WSDL",
this.class.classLoader)
proxy.initialize()

def currency = ['USD', 'EUR', 'CAD', 'GBP',
'AUD', 'SGD']
def rate = 0.0

swing = new SwingBuilder()

refresh = swing.action(
    name:'Refresh',
    closure:this.&refreshText,
    mnemonic:'R'
)

frame = swing.frame(title:'Currency Demo') {
    panel {
        label 'Currency rate from '
        comboBox(id:'from', items:currency)
        label ' to '
        comboBox(id:'to', items:currency)
        label ' is '
        textField(id:'currency', columns:10,
rate.toString())
```

```
        button(text: 'Go !', action: refresh)
    }
}
frame.pack()
frame.show()

def refreshText(event) {
    rate =
proxy.ConversionRate(swing.from.getSelectedI
```

```
tem(), swing.to.getSelectedItem())
    swing.currency.text = rate
}
```

GroovyWS grape config file

### ~/groovy/grapeConfig.xml

```
<ivysettings>
  <settings
defaultResolver="downloadGrapes" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/ivy-[revision].xml" />
        <artifact
pattern="${user.home}/.groovy/grapes/[organi
sation]/[module]/[types]/[artifact]-[revisio
n].[ext]" />
      </filesystem>
      <!-- todo add 'endorsed groovy
extensions' resolver here -->
      <ibiblio name="codehaus"
root="http://repository.codehaus.org/"
m2compatible="true" />
      <ibiblio name="snapshots.codehaus"
root="http://snapshots.repository.codehaus.o
rg/" m2compatible="true" />
      <ibiblio name="apache"
```

```
root="http://people.apache.org/repo/m2-ibiblio-rsync-repository/" m2compatible="true"/>
    <ibiblio name="apache-incubating"
root="http://people.apache.org/repo/m2-incubating-repository/" m2compatible="true"/>
    <ibiblio name="maven"
root="http://repo2.maven.org/maven2/"
m2compatible="true"/>
```

```
</chain>
</resolvers>
</ivysettings>
```

## GroovyWS installation

### *Distributions*

Please view the comments [here](#)

You can find a release of GroovyWS [here](#) or [here](#)

If you are you are using [maven](#), GroovyWS is available in the codehaus repositories

### *Snapshots*

If you want to live on the edge, you can find the snapshot jar files [here]<http://snapshots.repository.codehaus.org/org/codehaus/groovy/modules/groovyws/>].

## Publishing a web-service

### *The Server*

You can develop your web service using a groovy script and/or a groovy class. Imagine, we want to publish a web service performing simple arithmetics. First, we have to write the business class implementing the basic operations. In our example, we chose to just implement simple add and square operations. Note that this class is quite generic and could be use locally.

```
class MathService {
    double add(double arg0, double arg1) {
        return (arg0 + arg1)
    }
    double square(double arg0) {
        return (arg0 * arg0)
    }
}
```

Then we have to publish this service. In order to do so, we use [WSServer](#) and create one so called node linking our class with a URL.

```
import groovyx.net.ws.WSServer  
  
def server = new WSServer()  
  
server.setNode("MathService",  
"http://localhost:6980/MathService")  
  
server.start()
```

That's all !

### **TerraServer-USA by Microsoft**

TerraServer supports a Tiling Web Service that enables you to build applications that integrate with USGS imagery found on their site. Here is a sample of what you can achieve.

```
import groovyx.net.ws.WSClient;
def proxy = new
WSClient("http://terrasservice.net/TerraService.asmx?WSDL", this.class.classLoader)

proxy.initialize()

// Create the Place object
def place =
proxy.create("com.terrasserver_usa.terrasserver.Place")

// Initialize the Place object
place.city = "mountain view"
place.state = "ca"
place.country = "us"

// Geocode the place
def result =
proxy.ConvertPlaceToLonLatPt(place)

println "Longitude: ${result.lon}"
println "Latitude: ${result.lat}"
```

will give:

```
Longitude: -122.08000183105469
Latitude: 37.400001525878906
```

Using the Aegis mapping

Let say we have a server that manage a library in which you can add a book, find a book and get all the books. The server code will probably look like this:

## BookService.groovy

```
class BookService {

    private List allBooks = new ArrayList()

    Book findBook(String isbn) {
        for (book in allBooks) {
            if (book.isbn == isbn) return book
        }
        return null
    }

    void addBook(Book book) {
        allBooks.add(book)
    }

    Book[] getBooks() {
        return (Book[])allBooks.toArray(new
Book[allBooks.size()])
    }

}
```

with the class Book being something like that.

## Book.groovy

```
class Book {
    String author
    String title
    String isbn
}
```

To ignore the `metaClass` property a custom type mapping must be defined (for details refer to [Aegis Binding](#)).

## Book.aegis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings
xmlns:sample="http://DefaultNamespace">
    <mapping name="sample:Book">
        <property name="metaClass"
ignore="true" />
    </mapping>
</mappings>
```

However, if you compile custom data types from Java the bytecode won't contain a `metaClass` property and, hence, there is no need to define a custom mapping.

### Using the client with complex objects

The good thing here is that the client does not have to know about the `Book` class. It is automatically generated during the proxy creation time and can be used by your client. Here it is located in the `defaultnamespace` package since no package was used on the server side.

Here is how the client looks like now:

```
import groovyx.net.ws.WSClient

def proxy = new
WSClient("http://localhost:6981/BookService?
wsdl", this.class.classLoader)
proxy.initialize() // from 0.5.0

def books = proxy.getBooks()

for (book in books) println book

def book =
proxy.create("defaultnamespace.Book")

book.title = "Groovy in Action"
book.author = "Dierk"
book.isbn = "123"

proxy.addBook(book)

def bks = proxy.getBooks()

println bks.books[0].isbn
```

Iterating over the books is slightly more complicated since SOAP wrap the arrays in an element (in our case `ArrayOfBook`). Therefore you have to extract a field from that element. In our case:

```
def aob = proxy.getBooks()

for (book in aob.books) println book.name
```

### Using the Grails XFire plugin and GroovyWS

This usage is straightforward. Using grails let's create a simple XFire web service doing currency conversions. Let's assume, we can handle 3 currencies: AUD, USD and BGP. In order to keep things simple, we do assume a static exchange rate. Such a service would look like this:

```
class TestService {

    boolean transactional = false

    static expose=['xfire']

    static conversions = [
        'AUD': [ 'USD': 100.00D, 'GBP': 44.44D
    ],
        'USD': [ 'AUD': 1.00D, 'GBP': 88.88D
    ],
        'GBP': [ 'AUD': 22.22D, 'USD': 33.33D
    ]
    ]

    Double convert(String from, String to,
Double amount) {
        conversions[from][to] * amount
    }
}
```

The grails project creation is quite simple:

```
grails create-app xfire  
cd xfire  
grails install-plugin xfire  
grails create-service test  
vi grails-app/services/TestService.groovy  
(and paste the above code)  
grails run-app
```

The wsdl is available on

```
http://localhost:8080/xfire/services/test?wsdl
```

The following groovy script consumes our brand new web service

```
import groovyx.net.ws.WSClient

@Grab(group='org.codehaus.groovy.modules',
module='groovyws', version='0.5.0-SNAPSHOT')
def getProxy(wsdl, classLoader) {
    new WSClient(wsdl, classLoader)
}
proxy =
getProxy("http://localhost:8080/xfire/services/test?wsdl", this.class.classLoader)
proxy.initialize()

result = proxy.convert("AUD", "USD", 10.0)
println "10 AUD are worth ${result} USD"
```

### Using WSClient in Grails

Let's assume we want to use the temperature conversion service from grails. In order to make the project minimal, we will modify the default view of the project, add a controller proxying to the service and generate a minimal view to display the result. Here we go:

```
grails create-app hotstuff
cd hotstuff
vi grails-app/views/index.gsp
```

Modify the default view to

```
<html>
  <head>
    <title>Welcome to Grails</title>
    <meta name="layout" content="main" />
  </head>
  <body>
    <g:form name="temperature"
controller="temperature"
action="conversion">
      <p>
        <g:set var="units"
value="${'Fahrenheit', 'Celsius'}" />
        Convert <g:textField name="value"
value="10.0" /> <g:select name="from"
from="${units}" /> in <g:select name="to"
from="${units}" />
      </p>
      <g:submitButton name="submit"
value="Convert" />
    </g:form>
  </body>
</html>
```

```
grails create-controller temperature
```

Here is the controller code:

```
import groovyx.net.ws.WSClient
```

```
class TemperatureController {

    def index = { }

    def conversion = {
        def proxy = new
WSCClient("http://www.w3schools.com/webservices/temconvert.asmx?WSDL",
this.class.classLoader)
        proxy.initialize()

        def message

            if (!params?.value ||
params.from.equals(params.to)) {
                message = "Units are probably
the same. Don't aske me :)"
            } else {
                def result
                if
("Celsius".equals(params.from)) {
                    result =
proxy.CelsiusToFahrenheit(params.value)
                } else {
                    result =
proxy.FahrenheitToCelsius(params.value)
                }
                message = "${params.value}
degrees ${params.from} are ${result} degrees
${params.to}"
            }
        }
    }
}
```

```
    }  
  
    flash.message = message  
  }  
}
```

Then, we create the view to display the result

```
mkdir grails-app/views/temperature  
vi  
grails-app/views/temperature/conversion.gsp
```

Here is the simplistic view:

```
<html>
  <head>
    <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8" />
    <meta name="layout" content="main"
/>
    <title>Here we are</title>
  </head>
  <body>
    <div class="body">
      <g:if test="${flash.message}">
        <div
class="message">${flash.message}</div>
      </g:if>
    </div>
  </body>
</html>
```

 Currently, the Grab annotation cannot be used without interfering with native grails jars, so you have to add the following jars into your application lib directory

```
$ ls -lrt lib/
total 31960
-rw-r--r--  1 alleon  alleon    26072 Apr 15
11:01 groovyws-0.5.0-SNAPSHOT.jar
-rw-r--r--  1 alleon  alleon  4201229 Apr 15
11:03 cxf-bundle-2.1.4.jar
-rw-r--r--  1 alleon  alleon   520969 Apr 15
12:48 wstx-asl-3.2.7.jar
-rw-r--r--  1 alleon  alleon   60686 Apr 15
```

```
13:31 commons-logging-1.1.1.jar
-rw-r--r--  1 alleon  alleon  1485863 Apr 15
13:31 bcprov-jdk15-140.jar
-rw-r--r--  1 alleon  alleon   143806 Apr 15
13:31 XmlSchema-1.4.4.jar
-rw-r--r--  1 alleon  alleon   291779 Apr 15
13:31 FastInfoset-1.2.2.jar
-rw-r--r--  1 alleon  alleon   226915 Apr 15
13:31 jaxen-1.1.1.jar
-rw-r--r--  1 alleon  alleon  3127729 Apr 15
13:31 jaxb-xjc-2.1.9.jar
-rw-r--r--  1 alleon  alleon   856752 Apr 15
13:31 jaxb-impl-2.1.9.jar
-rw-r--r--  1 alleon  alleon    89967 Apr 15
13:31 jaxb-api-2.1.jar
-rw-r--r--  1 alleon  alleon    14611 Apr 15
13:31
geronimo-ws-metadata_2.0_spec-1.1.2.jar
-rw-r--r--  1 alleon  alleon    28804 Apr 15
13:31 geronimo-stax-api_1.0_spec-1.0.1.jar
-rw-r--r--  1 alleon  alleon    47817 Apr 15
13:31 geronimo-jaxws_2.1_spec-1.0.jar
-rw-r--r--  1 alleon  alleon   204851 Apr 15
13:31 geronimo-javamail_1.4_spec-1.5.jar
-rw-r--r--  1 alleon  alleon    12452 Apr 15
13:31 geronimo-annotation_1.0_spec-1.1.1.jar
-rw-r--r--  1 alleon  alleon    34126 Apr 15
13:31 geronimo-activation_1.1_spec-1.0.2.jar
-rw-r--r--  1 alleon  alleon   444503 Apr 15
13:31 xmlsec-1.4.2.jar
-rw-r--r--  1 alleon  alleon    84091 Apr 15
```

```
13:31 xml-resolver-1.2.jar
-rw-r--r--  1 alleon  alleon  3176148 Apr 15
13:31 xalan-2.7.1.jar
-rw-r--r--  1 alleon  alleon   313982 Apr 15
13:31 wss4j-1.5.6.jar
-rw-r--r--  1 alleon  alleon  148429 Apr 15
13:31 wsdl4j-1.6.2.jar
-rw-r--r--  1 alleon  alleon  278281 Apr 15
13:31 serializer-2.7.1.jar
-rw-r--r--  1 alleon  alleon  278145 Apr 15
13:31 saaj-impl-1.3.2.jar
-rw-r--r--  1 alleon  alleon   18817 Apr 15
13:31 saaj-api-1.3.jar
-rw-r--r--  1 alleon  alleon   32900 Apr 15
```

```
13:31 neethi-2.0.4.jar
-rw-r--r--  1 alleon  alleon   153115 Apr 15
13:31 jdom-1.1.jar
```

⚠ If you have been running our very first example all these jars are probably in your

```
~/groovy/grapes/
```

directory

BuildConfig.groovy

If you reference groovyws in your grails app then you will need a set of exclusions to prevent version conflicts ( from list [Nabble - Ronny Løvtingen](#) ):

```
compile("org.codehaus.groovy.modules:groovyws:0.5.2") {
    excludes 'geronimo-servlet_2.5_spec',
    'servlet-api', 'jaxb-xjc', 'jaxb-impl',
    'xml-apis', 'saaj-impl', 'junit',
    'slf4j-jdk14', 'xmlParserAPIs', 'jaxb-api',
    'saaj-api', 'xmlbeans', 'jaxen',
    'geronimo-stax-api_1.0_spec',
    'geronimo-activation_1.0.2_spec',
    'abdera-client',
    'geronimo-activation_1.1_spec'
}
```

## Using WS-Security

Before adding WS-Security to your web services, you must make sure to have:

1. Access to a certificate authority (CA) - either your own using OpenSSL for example or an external one like [CA Cert](#)
2. Access to keytool (usually through your Java SDK)

*Securing a server*

This is quite easy. You need first to create a keystore with a key pair. During that process you will be asked for passwords for protecting your keystore and private key. Let's choose 'groovyws' for both of them. This can be done for example with:

```
keytool -genkey -keyalg RSA -dname "C=FR,  
O=GroovyWS Inc, OU=GroovyWS Test Centre,  
CN=Server" -alias server \  
-keystore Server.jks
```

Then you need to generate the Certificate Signing Request like this:

```
keytool -certreq -alias server -file  
ServerCertificateRequest.pem -keystore  
Server.jks
```

You need to get the server certificate from your CA using the newly generated request. Let's assume you get back the file named ServerCertificate.pem. You need to include that certificate into your keystore. Usually this won't be possible unless your keystore contains the certificate of your CA. Let's add those two certificates:

```
keytool -import -alias TheCA -file  
TheCACert.pem -keystore Server.jks  
  
keytool -import -alias server -file  
ServerCertificate.pem -keystore Server.jks
```

You are now ready to start your server:

```
Map<String, String> mapServer = [  
  
    "https.keystore": "path/to/Server.jks",  
  
    "https.keystore.pass": "groovyws",  
  
    "https.truststore": "",  
  
    "https.truststore.pass": ""  
  
    ]  
  
    server = new WSServer(myServiceUrl)  
    server.setSSL(mapServer)  
  
    server.setClientAuthentication(false)  
    server.start()
```

In the above example, the client authentication is not required. If you turn the flag to true (or omit the line, it is true by default), the client must trust the server, you therefore have to provide a keystore containing the server certificate.

```
keytool -import -alias server -file  
ServerCertificate.pem -keystore  
TrustingTheServer.jks
```

```

Map<String, String> mapClient = [

    "https.keystore": "",

    "https.keystore.pass": "",

    "https.truststore": "path/to/TrustingTheServer.jks",

    "https.truststore.pass": "client"

]

    def proxy = new
WSClient(myServiceUrl+"?wsdl",
this.class.classLoader)
    proxy.setSSLProperties(mapClient)
    proxy.initialize()

    assert proxy.add(2.0 as double, 5.0
as double) == 7.0
    assert proxy.square(4.0 as double)
== 16.0

```

You may also setup more complex configurations where both the client & server need to trust each others ...

## WSClient configuration

### *Using proxies*

If you are using a proxy for accessing internet, you can use the following environment variables to get rid of it:

- http.proxyHost
- http.proxyPort
- proxy.user
- proxy.password

or directly use the following in your code:

```
proxy.setProxyProperties([:]) // map  
containing all or part of the above  
properties
```

The proxy will not be used when the WSCClient accesses the WSDL during initialization.

#### *Using basic authentication*

If your server use basic authentication, you need to set up the following properties:

- http.user
- http.password

or directly use

```
proxy.setBasicAuthentication(user, password)
```

#### *Setting a time out value*

If your want to set a time out value for the connection to the server just use:

```
proxy.setConnectionTimeout(value_in_millise  
conds)
```

#### *Using SOAP 1.2 (in test/review)*

If your server is proposing SOAP 1.2 endpoints, then you can use:

```
import groovyx.net.ws.cxf.SoapVersion  
  
proxy.setPreferredSoapVersion(SoapVersion.SO  
AP_1_2)
```

## **GSP**

GSP are not maintained as a standalone module. But it has been forked and reintegrated in [Grails](#).

---

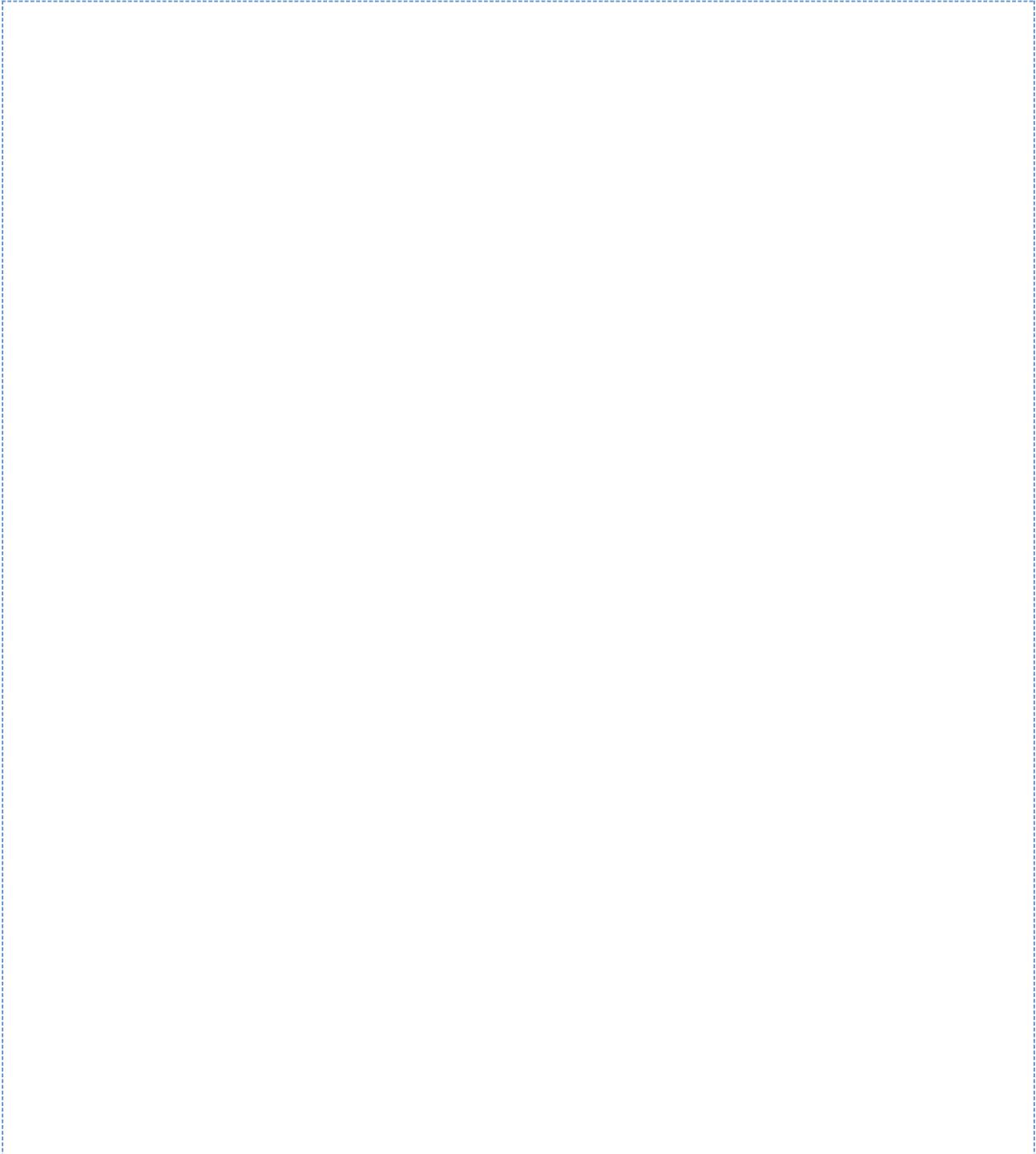
GSP means GroovyServer Pages, which is similar to JSP (JavaServer Pages).

GSP Module Project has started originally by Troy Heninger.

The original sources of GSP module 1.1 can be found at Groovy's [SVN](#) repository.

There is also a new GSP project page: <https://gsp.dev.java.net/>

***Sample GSP: AttributeTest.gsp***



```
<%
```

```
    if (session.counter == null)
        session.counter = 1
    else
        session.counter++

    session.setAttribute("id", "tmpID")
    session.setAttribute("uid", "userID")
```

```
    request.x = 123
    application.x = 500
```

```
    if (application.counter == null)
        application.counter = 1
    else
        application.counter++
```

```
%>
```

```
application.counter = ${application.counter}
<br>
session.counter = ${session.counter} <br>
session.id = ${session.id} <br>
session.uid = ${session.uid} <br>
session.getAttribute('id') =
${session.getAttribute('id')} <br>
request.x = ${request.x} <br>
(application.x == null ?) = ${application.x
== null} <br>
application.x = ${application.x} <br>
```

```
<servlet>
  <servlet-name>GSP</servlet-name>

  <servlet-class>groovy.modules.pages.GroovyPages</servlet-class>
  <init-param>
    <param-name>encoding</param-name>

    <param-value>ISO-8859-1</param-value>
  </init-param>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>GSP</servlet-name>
  <url-pattern>*.gsp</url-pattern>
</servlet-mapping>
</servlet>
```

## GSQL

GSQL supports easier access to databases using Groovy.

Groovy has [built-in](#) SQL support. If that doesn't meet your needs, consider this module.

The source code can be found here :

- [GSQL](#)

## Examples

## SqlGeneratorTest.groovy

```
/**
 * Test to verify valid construction of
 * default DDL
 *
 * @author <a
 * href="mailto:jeremy.rayner@bigfoot.com">Jere
 * my Rayner</a>
 * @version $Revision: 1.2 $
 */
package org.javanicus.gsql

import java.io.*

class SqlGeneratorTest extends
GroovyTestCase {
    def database
    def sqlGenerator

    void setUp() {
        def typeMap = new TypeMap()
        def build = new
RelationalBuilder(typeMap)
        def sqlGenerator = new
SqlGenerator(typeMap, System.getProperty(
"line.separator", "\n" ))

        def database =
build.database(name: 'genealogy') {
```

```

        table(name:'event') {
            column(name:'event_id',
type:'integer', size:10, primaryKey:true,
required:true)
            column(name:'description',
type:'varchar', size:30)
        }
        table(name:'individual') {
            column(name:'individual_id',
type:'integer', size:10, required:true,
primaryKey:true, autoIncrement:true)
            column(name:'surname',
type:'varchar', size:15, required:true)
            column(name:'event_id',
type:'integer', size:10)
            foreignKey(foreignTable:'event')
{
reference(local:'event_id',foreign:'event_id
')
        }
            index(name:'surname_index') {
                indexColumn(name:'surname')
            }
        }
    }

void testGenerateDDL() {
    def testWriter = new PrintWriter(new
FileOutputStream("SqlGeneratorTest.sql"))

```

```
sqlGenerator.writer = testWriter
```

```
sqlGenerator.createDatabase(database, true)
```

```
testWriter.flush()
```

```
    }  
  
}
```

## HTTP Builder

HTTPBuilder provides a convenient builder API for complex HTTP requests. It is built on top of [Apache HttpClient](#). The [project home page](#) includes full documentation and downloads.

Note that the example below works for HTTPBuilder version 0.5.0-RC2. In prior versions, the `uri` property was called `url`. Also note that `@Grab` dependency management requires Groovy 1.6 or later. More information may be found [here](#).

Example: HTTP GET, automatically parsed as a JSON response.

```
@Grab(group='org.codehaus.groovy.modules.http-builder', module='http-builder',  
version='0.5.0-RC2' )  
import groovyx.net.http.*  
import static groovyx.net.http.ContentType.*  
import static groovyx.net.http.Method.*  
  
def http = new HTTPBuilder(  
    'http://ajax.googleapis.com' )  
  
// perform a GET request, expecting JSON  
response data  
http.request( GET, JSON ) {  
    uri.path = '/ajax/services/search/web'  
    uri.query = [ v:'1.0', q: 'Calvin and  
Hobbes' ]  
  
    headers.'User-Agent' = 'Mozilla/5.0  
Ubuntu/8.10 Firefox/3.0.4'
```

```
// response handler for a success response
code:
response.success = { resp, json ->
    println resp.statusLine

    // parse the JSON response object:
    json.responseData.results.each {
        println "  ${it.titleNoFormatting} :
${it.visibleUrl}"
    }
}

// handler for any failure status code:
response.failure = { resp ->
    println "Unexpected error:
${resp.statusLine.statusCode} :
```

```
    ${resp.statusLine.reasonPhrase} "  
    }  
}
```

## JideBuilder

JideBuilder is a Groovy builder for the open source [JIDE Common Layer](#)

### Warning

JideBuilder has been relocated to the [Griffon](#) project, please update your links to <http://griffon.co.dehaus.org/JideBuilder>. The information on this page may be outdated.

## Contribution Overview

JideBuilder will let you create Jide components and standard Swing components as SwingBuilder does, because it doesn't override the default names of previously registered components in SwingBuilder, this means that you can mix both types of components with the same builder.

```

import griffon.builder.jide.JideBuilder

def jide = new JideBuilder()
jide.edt {
    frame( id: 'frame', title: 'JideBuilder',
pack: true, visible: true ){
        panel( layout: new BorderLayout(5,5)
){
            button( label: "Swing", constraints:
BorderLayout.WEST )
            jideButton( label: "Jide",
constraints: BorderLayout.EAST )
            textField( constraints:
BorderLayout.NORTH )
            calculator( constraints:
BorderLayout.CENTER )
        }
    }
}

```

There are 5 \*Searchable components that will add searching capabilities to a previously created widget or an internal one, exposing the searchable widget and the searchable wrapper if an 'id' attribute was provided, for example ComboBoxSearchable:

```

import griffon.builder.jide.JideBuilder

def jide = new JideBuilder()
jide.edt {
    frame( id: 'frame', title: 'JideBuilder',
pack: true, visible: true ){
        panel( layout: new BorderLayout(5,5)
){
            comboBox( id: "mycombo", items:
[1,2,3] )
            comboBoxSearchable( id: "c1",
comboBox: mycombo )
            comboBoxSearchable( id: "c2", items:
[4,5,6] )
        }
    }
}
assertNotNull jide.ci_searchable

```

JIDE Common Layer defines a basic structure for dialogs with StandardDialog, which is an abstract class, you may create your own dialogs based on that template by calling 'standardDialog()', by default all 3 panels will be empty, but you can provide your own content by calling 'dialogBannerPanel()', 'dialogContentPanel()' and 'dialogButtonPanel()' inside its closure.

All JTextComponents (even those created with regular SwingBuilder methods) will accept a 'selectAll' property, which will call com.jidesoft.swing.SelectAllUtils.install() on the component.

SearchableBar has an additional property 'install' which will register a SearchableBar.Installer automatically into the immediate container. This property must be a Map and may have the following properties:

- constraints: must be a valid Object accepted by the container's layout (required).
- keyStroke: must be a javax.swing.KeyStroke, defaults to CTRL + F (optional).

#### ***Team Members***

Andres Almiray [aalmiray at users dot sourceforge dot net]

#### **Download**

[jidebuilder-2.0](#)

## Installing

Drop jidebuilder-2.0 into \$GROOVY\_HOME/lib along with its dependencies

- [jide-oss-2.3.0](#)
- [svg-salamander-1.0](#)

## Pre-requisites

Groovy 1.6-beta-2 is the required minimum version to run JideBuilder 2.0

Previous versions of JideBuilder will run with earlier versions of Groovy, it is recommended that you upgrade to the 1.6.x series in order to take advantage of

- @Bindable and ASTTransformations
- short binding syntax
- numerous enhancements made to SwingBuilder and FactoryBuilderSupport

## Documentation

### Alphabetical list of components

Element	Class	Notes
animator	Animator	
autoCompletion	AutoCompletion	Wraps an existing JComboBox with 'comboBox' attribute Wraps an existing JTextComponent with 'textComponent' attribute Will create an internal JTextField if 'textComponent' and 'comboBox' are omitted If 'textComponent' is specified (or internal JTextField is created), 'searchable' or 'list' must also be specified Additional attributes may be set on the wrapped component
autoCompletionComboBox	AutoCompletionComboBox	
autoResizingTextArea	AutoResizingTextArea	
bannerPanel	BannerPanel	
bottom		JideSwingUtilities.createBottomPanel
buttonPanel	ButtonPanel	
calculator	Calculator	

center		JideSwingUtilities.createCenterPanel
checkboxList	CheckBoxList	
checkboxListWithSelectable	CheckBoxListWithSelectable	
checkboxTree	CheckBoxTree	
clickThroughLabel	ClickThroughLabel	
comboBoxSearchable	ComboBoxSearchable	Wraps an existing JComboBox with 'comboBox' attribute Will create an internal JComboBox if 'comboBox' is omitted Additional attributes may be set on the wrapped component Obeys 'items' when creating an internal JComboBox.
contentContainer	ContentContainer	
dateSpinner	DateSpinner	
dialogBannerPanel	DialogBannerPanel (from JideBuilder)	Use it inside standardDialog
dialogButtonPanel	DialogButtonPanel (from JideBuilder)	Use it inside standardDialog
dialogContentPanel	DialogContentPanel (from JideBuilder)	Use it inside standardDialog
dialogPage	DefaultDialogPage (from JideBuilder)	Use it inside a multiplePageDialog Add a 'closure' attribute as the body of lazyInitialize() default does nothing
fileIntelliHints	FileIntelliHints	Needs 'textComponent' attribute Additional attributes may be set on the wrapped component
folderChooser	FolderChooser	
gripper	Gripper	
headerBox	HeaderBox	

jideButton	JideButton	
jideBorderLayout	JideBorderLayout	
jideBoxLayout	JideBoxLayout	
jideMenu	JideMenu	Use a closure with 'customize' attribute to create a PopupMenuConfigurer
jideOptionPane	JideOptionPane	
jidePopup	JidePopup	
jidePopupMenu	JidePopupMenu	
jideScrollPane	JideScrollPane	
jideSplitButton	JideSplitButton	Use a closure with 'customize' attribute to create a PopupMenuConfigurer
jideSplitPane	JideSplitPane	
jideTabbedPane	JideTabbedPane	
jideToggleButton	JideToggleButton	
jideToggleSplitButton	JideToggleSplitButton	Use a closure with 'customize' attribute to create a PopupMenuConfigurer
labeledTextField	LabeledTextField	
left		JideSwingUtilities.createLeftPanel
listDataIntelliHints	ListDataIntelliHints	Needs 'textComponent' and 'completionList' attributes, Additional attributes may be set on the wrapped component
listSearchable	ListSearchable	Wraps an existing JList with 'list' attribute Will create an internal JList if 'list' is omitted Additional attributes may be set on the wrapped component

multilineLabel	MultilineLabel	
multiplePageDialog	MultiplePageDialog	
multiplePageDialogPane	MultiplePageDialogPane	
nullButton	NullButton	
nullCheckBox	NullCheckBox	
nullJideButton	NullJideButton	
nullLabel	NullLabel	
nullPanel	NullPanel	
nullRadioButton	NullRadioButton	
nullTristateCheckBox	NullTristateCheckBox	
paintPanel	PaintPanel	
pointSpinner	PointSpinner	
popupMenuCustomizer	DefaultPopupMenuCustomizer (from JideBuilder)	Use 'closure' attribute to define the body of customize()
rangeSlider	RangeSlider	
resizableDialog	ResizableDialog	
resizableFrame	ResizableFrame	
resizablePanel	ResizablePanel	
resizableWindow	ResizableWindow	
right		JideSwingUtilities.createRightPanel
searchableBar	SearchableBar	Wraps an existing Searchable with 'searchable' attribute Additional attributes may be set on the wrapped component
simpleScrollPane	SimpleScrollPane	
splitButtonGroup	SplitButtonGroup	

standardDialog	DefaultStandardDialog (from JideBuilder)	Use 'dialogBannerPanel', 'dialogContentPanel' and 'dialogButtonPanel' to provide additional configuration for each panel
styledLabel	StyledLabel	
tableSearchable	TableSearchable	Wraps an existing JTable with 'table' attribute Will create an internal JTable if 'table' is omitted Additional attributes may be set on the wrapped component
textComponentSearchable	TextComponentSearchable	Wraps an existing JTextComponent with 'textComponent' attribute Will create an internal JTextField if 'textComponent' is omitted Additional attributes may be set on the wrapped component
top		JideSwingUtilities.createTopPanel
treeSearchable	TreeSearchable	Wraps an existing JTree with 'tree' attribute Will create an internal JTree if 'tree' is omitted Additional attributes may be set on the wrapped component
tristateCheckBox	TristateCheckBox	
svgIcon	ResizableSVGIcon	<ul style="list-style-type: none"> <li>• path: same as svgResourcePath but allows aliases</li> <li>• trackSize: enables/disables resizing (default: false)</li> <li>• resizePercentage: controls how much the icon will change size, value must be in range 10..100 (default: 100)</li> <li>• retainAspectRatio: controls how resizing will take effect (default: true)</li> <li>• size: alias for 'preferredSize'</li> </ul>

## Developers

Andres Almiray

**Source Control**

<http://svn.codehaus.org/griffon/builders/jidebuilder>

### ***Building***

JideBuilder uses Ant as its build tool, which means that if you want to build your own version of JideBuilder from source you'll need to have it installed. Follow the instructions at <http://ant.apache.org>

### ***Contributing***

Please contact the team members by e-mail.

### **Community**

#### ***Mailing List(s)***

<http://griffon.codehaus.org/Mailing+Lists>

#### ***Issue tracker***

<http://jira.codehaus.org/browse/GRIFFON>

## **MetaBuilder**

### **MetaBuilder**

As if there weren't enough options already for constructing your own builders in Groovy, along comes another: MetaBuilder. Quite literally, MetaBuilder is a builder that builds builders. Through some simple examples, this article will show you how you can put MetaBuilder to work for you in just three easy steps. To follow along, simply visit [SourceForge](#) to get the MetaBuilder distribution and include **groovytools-builder-x.x.x.jar** in your classpath.

Oh, and you'll also need Groovy 1.5 and Java 1.5, too 😊

### **MetaBuilder in Three Easy Steps**

1. Create an instance of MetaBuilder
2. Define your domain specific language (DSL)
3. Build your objects

#### ***Create an instance of MetaBuilder***

There's almost nothing to this step but just to call a constructor:

## Step 1

```
import groovytools.builder.*
```

```
MetaBuilder mb = new
```

```
MetaBuilder(getClass().getClassLoader())
```

### ✓ Tip

You don't have to pass MetaBuilder a class loader, but in some cases, especially in scripts that declare new classes such as this one, it's necessary. This can be done by setting the `classLoader` property or by passing the class loader in MetaBuilder's constructor, as shown above.

### *Define Your Domain Specific Language (DSL)*

MetaBuilder provides a DSL, implemented as a Groovy builder, for defining your own builders. If you are already familiar with builders in Groovy, it takes just one simple example to get you started. Continuing where we left off above, let's start defining a customer class and customer builder:

## Step 2

```
class Customer {
    def name
    def dateOfBirth
    def ssn
    def phone
}

mb.define {
    customer(factory: Customer) {
        properties {
            name()
            dateOfBirth()
            ssn()
        }
    }
}
```

In the previous snippet, `define` is used to tell MetaBuilder that we are going to create some new definitions, or schema, for the objects that our new builder can create. MetaBuilder keeps track of our definitions and it's even possible to reuse and extend these definitions, as we'll see later.

`customer` is the name of the schema and the `factory` attribute tells MetaBuilder what object to create whenever a `customer` is to be built. `properties` contains a list of property names.

### ✓ Tip

MetaBuilder will throw an exception if a build script attempts to use unspecified or misspelled properties. So, for example, even though `phone` is a member of `Customer`, MetaBuilder won't allow you to use it unless you add it to your schema.

### ***Build Your Objects***

Building objects is now just a matter of telling MetaBuilder that is what you want to do:

## Step 3

```
def aCustomer = mb.build {
  customer {
    name = 'J. Doe'
    dateOfBirth = '1/1/1900'
    ssn = '555-55-5555'
  }
}

// this is equivalent
aCustomer = mb.build {
  customer ( name: 'J. Doe', dateOfBirth:
'1/1/1900', ssn: '555-55-5555' )
}

// you can even mix up the styles:
aCustomer = mb.build {
  customer ( name: 'J. Doe', dateOfBirth:
'1/1/1900' ) {
    ssn = '555-55-5555'
  }
}
```

Great! If you've been following along in your own IDE, you hopefully now have the basics down and are ready to take a look at some advanced techniques.

### ✓ Tip

`MetaBuilder.build()` returns the last object constructed. If your build script creates multiple objects, use the `buildList` method instead to return all of them.

## Catching Errors

What would happen if you used a property that was not in the schema or mistyped a legitimate property name? For example:

## Catching Errors

```
aCustomer = mb.build {
    customer {
        name = 'J. Doe'
        dob = '1/1/1900' // should
have been 'dataOfBirth'
        ssn = '555-55-5555'
        phone = '1-555-555-5555' // not
allowed
    }
}
```

If you executed this, MetaBuilder will throw the following exception:

```
groovytools.builder.PropertyException: Property 'dob': property unknwn
```

Assuming you fix dob only and retry, MetaBuilder will throw another exception:

```
groovytools.builder.PropertyException: Property 'phone': property unknwn
```

Despite the fact that phone is an actual property of your class, MetaBuilder only goes by what's in the schema. That kind of checking can protect private or sensitive properties. Read on to see how MetaBuilder can do even more with some additional information.

### Controlling the Build

The purpose of this section is to go a bit deeper into MetaBuilder's capabilities.

#### *More on factory*

Let's take another look at the `factory` attribute used earlier. By simply specifying the `factory` attribute in your schemas, you tell MetaBuilder how to build the right object every time. MetaBuilder was designed to accept as wide a variety of values as possible. For example, you can specify the `factory` attribute as any of the following:

- Class, i.e. `Customer`
- String, i.e. `'Customer'`
- instance of `groovy.util.Factory`
- Closure

Feel free to consult [MetaBuilder Meta-Schema](#) for all the gory details on each of the attribute values MetaBuilder

accepts.

This next example demonstrates how one might use closure to create Customer objects:

## Specifying a Factory using a Closure

```
mb.define {
  customer2(factory: { new Customer() } )
{
  properties {
    name()
    dateOfBirth()
    ssn()
  }
}

def aCustomer2 = mb.build {
  customer2 {
    name = 'J. Doe'
    dateOfBirth = '1/1/1900'
    ssn = '555-55-5555'
  }
}
```

### *Property Attributes*

MetaBuilder supports a number of useful attributes on properties. These include the following:

- **def**: the default value or a Closure to produce the default value, as needed
- **req**: if true, a value must be specified (it could be null though)
- **property**: an alternative property name or Closure to use to set the value
- **min**: the minimum length for a property value
- **max**: the maximum length for a property value
- **check**: causes an exception if the value fails the following test:

```
switch(value) {
  case check: return true
}
return false
```

✓ **Tip**

The **check** attribute accepts the following types of objects which makes enforcing constraints easy:

- **Closures**
- **Patterns**
- **Classes**
- **Numbers**
- **Strings**
- **Ranges**
- **Collections**

Here's an example showing combinations of some of the above:

#### MetaBuilder Property Attributes in Action

```
mb.define {
  customer3(factory: Customer) {
    properties {
      name(req: true, min: 1)
      dob(property: 'dateOfBirth')
      ssn(check: ~/\d{3}-\d{2}-\d{4}/)
    }
  }
}

def aCustomer3 = mb.build {
  customer4 {
    name = 'J. Doe'
    dob = '1/1/1900'
    ssn = '555-55-5555'
  }
}
```

#### **Reusing and Extending Schema: the *schema* Attribute**

Use the `schema` attribute to tell MetaBuilder that you want to reuse a schema. In the next example, we'll create a new `Phone` class and update our schema to use it:

## Adding a Phone Class and Schema

```
class Phone {
  def type
  def number
}

mb.define {
  phone (factory: Phone) {
    properties {
      type(check:
['home', 'cell', 'work'], def: 'home')
      number(req: true, check:
~/\d{3}-\d{3}-\d{4}/)
    }
  }
}
```

Now, let's make phone a required property on our customer schema:

## Using the Phone Schema

```
mb.define {
  customer4(factory: Customer) {
    properties {
      name(req: true, min: 1)
      dob(property: 'dateOfBirth')
      ssn(check: ~/\d{3}-\d{2}-\d{4}/)
      phone(schema: 'phone', req:
true)
    }
  }
}

def aCustomer4 = mb.build {
  customer4 {
    name = 'J. Doe'
    dob  = '1/1/1900'
    ssn  = '555-55-5555'
    phone {
      type = 'home'
      number = '123-456-7890'
    }
  }
}
```

### *Collections*

So far, we've only looked at examples of properties, but MetaBuilder also supports collections. You define collections like you define properties, just provide a list of them and set attributes as needed.

## Defining Schema with Indexed and Non-Indexed Collections

```
class Customer2 extends Customer {
  def phoneList = []
  def addresses = [:]
}

mb.define {
  customer5(factory: Customer2) {
    properties {
      name(req: true, min: 1)
      dob(property: 'dateOfBirth')
      ssn(check: ~/\d{3}-\d{2}-\d{4}/)
      phone(schema: 'phone', req:
true)
    }
    collections {
      phoneList(min: 1) {
        phone(schema: phone)
      }
      addresses (key: 'type', min: 1,
max:2) {
        address() {
          properties {
            type(check:
['billto', 'shipto'], def: 'billto')
            street()
            city()
            state()

```

```
zip()
```

```
}
```

```
}
```

```
}
```

```
}  
  }  
}
```

The above definition extends `Customer` and adds a list of phone numbers and a map of addresses. Like properties, collections are mapped to properties of an object by the name.

Note how `address` is defined directly within the collection. Nesting definitions can make the definition a bit more brief, but comes at the risk of creating definitions that aren't as easily reused.

Another thing to note is the use of the `key` attribute in the `addresses` collection. Presence of the `key` attribute tells MetaBuilder that the parent-child relationship is indexed. In the following example, you can see that `customer5` has both a list of `phone` and map of `addresses` using the `address`'s `type` and the `key`.

## Building Objects with Indexed and Non-Indexed Collections

```
def aCustomer5 = mb.build {
  customer5 {
    name = 'J. Doe'
    dob  = '1/1/1900'
    ssn  = '555-55-5555'
    phone {
      type = 'home'
      number = '123-456-7890'
    }
    phoneList {
      phone(type: 'work', number:
'111-222-3333')
      phone(type: 'cell', number:
'444-555-6666')
    }
    address {
      type    = 'billto'
      street  = '1234 Some St.'
      city    = 'Some City'
      zip     = '12345'
    }
    address {
      type    = 'shipto'
      street  = '1234 Some Other St.'
      city    = 'Some Other City'
      zip     = '12345'
    }
  }
}
```

Collections support a number of useful properties:

- **collection**: specifies an alternate name for the collection or a Closure used to get the collection from the parent.
- **key**: specifies a property name or a Closure used to get the key from a child. Also indicates the collection is a map.
- **add**: specifies a method name or Closure used to add the child directly to the parent, supercedes `collection`.
- **min**: the minimum size of the collection
- **max**: the maximum size of the collection
- **size**: alternate method or Closure that may be used to get the collection size.

## Where to Go From Here

This whirlwind tour of MetaBuilder really only just scratched the surface of its features and capabilities. If you get stuck, be sure to check out the [MetaBuilder Meta-Schema](#), which describes the entire MetaBuilder feature set.

Also included in the release are a number of tests and examples that are also useful to look at.

## Mailing List

- [user@groovy.codehaus.com](mailto:user@groovy.codehaus.com)

## Developer(s)

- [Didge](#)

## Native Launcher

The groovy native launcher is a native program for launching groovy scripts. It compiles to an executable binary file, e.g. `groovy.exe` on Windows. Note that you still need to have groovy and a JRE or JDK installed, i.e. the native launcher is a native executable replacement for the startup scripts (`groovy.bat`, `groovy`).

The native launcher is included in the Groovy Windows installer. For other platforms, if your package management system does not have it, you will have to compile it yourself. This is not hard, you just need to have SCons (and Python) installed.

## How it works

Essentially the launcher does the same thing that the normal Java launcher (`java` executable) - it dynamically loads the dynamic library containing the JVM and hands the execution over to it. It does not start a separate process (i.e. it does not call the `java` executable).

## Status

The native launcher aims to compile and run on any os and any JDK/JRE  $\geq 1.4$ . If you are using a combination of `os+jdk/jre` that is not supported, please post a JIRA enhancement request and support will be added.

At the moment, the following platforms have been tested:

- Windows (XP, Vista)
- Linux (SuSE, Ubuntu) on x86
- Solaris on sparc
- Mac OS X on x86

At the moment, the following JDKs/JREs have been tested

- several versions of Sun JRE/JDK (from 1.4, 1.5 and 1.6 series)
- JRockit (on Windows)

The current version of the native launcher works with any version of Groovy.

### Pre-compiled binaries

Here are precompiled binaries for Windows:

[groovy.exe](#)

[groovyw.exe](#)

They are not guaranteed to be completely up to date with the sources of HEAD in the Subversion repository, but they should work.

Hopefully we will have precompiled binaries for all supported platforms in the future.

### What about the other Groovy executables?

The same binaries work for all the Groovy executables (groovy, groovyc, groovysh...). Just copy / (soft)link to the executable with the name of the executable you want, e.g. on Windows

```
copy groovy.exe groovyc.exe
```

and on Linux, Mac OS X, Solaris, etc.:

```
ln -s groovy groovyc
```

In addition the build produces launchers for gant and grails.

A note about gant: the gant launcher is meant for standalone gant installation. To use gant installed into your groovy installation (e.g. by groovy windows installer) use the renamed groovy executable as described above.

### Compiling

To get the sources please see [the native launcher git repository](#).

The executables are compiled using [SCons](#). For Windows there is a SCons installer that can be used after having installed [Python](#) - The SCons build framework is written in Python. SCons is part of Cygwin and can be installed using the usual installer. The same goes for MacPorts on Mac OS X, though there is a disk image installer as well. For Ubuntu, Debian, Fedora, SuSE, etc. SCons is packages and so the usual package management can be used to install. The build has only been tested with SCons 0.98 and greater, it may not work with earlier versions of SCons.

Once you have SCons installed then simply typing:

```
scons
```

will compile things for the platform you are compiling on. Type:

```
scons -h
```

for a short help message about the native launcher build and

```
scons -H
```

for a message about using scons.

#### Setting "well known" locations at compile time

The native launcher, if compiled with default options, will look up groovy and java installation locations as described below. However, on some platforms (linux and other \*nix variants) it may be desirable for performance and security reasons to "hardwire" these locations, i.e. set them to fixed values at compile time.

Currently, native launcher supports setting three things at compile time (together or separately): GROOVY\_HOME, GROOVY\_STARTUP\_JAR and JAVA\_HOME by passing in the values to scons build via "extramacros" option.

Example:

```
scons  
extramacros="GROOVY_HOME=/usr/share/groovy  
GROOVY_STARTUP_JAR=/usr/share/groovy/lib/groovy-1.5.1.jar"
```

To test that the binary you compiled uses preset locations, run groovy w/ environment variable \_\_JLAUNCHER\_DEBUG set, e.g.

```
__JLAUNCHER_DEBUG=true build/groovy -v
```

The launcher will print debug info to stderr, among other things how it obtained the locations of groovy and java installations.

## ***Compiling on windows***

On Windows you can either compile with the Microsoft cl compiler and linker or you can use GCC, either the [MinGW](#) version of the [Cygwin](#) version.

If you are not already using Cygwin, then you may want to investigate using MSYS and the MinGW toolchain.

- Download and install Python.  
Download and install SCons.
- Download the 'Automated MinGW Installer' from [http://sourceforge.net/project/showfiles.php?group\\_id=2435&package\\_id=240780](http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=240780)
- Run it (it automatically installs the needed compiler, you don't have to choose additional packages)
- Download MSYS from [http://sourceforge.net/project/showfiles.php?group\\_id=2435&package\\_id=24963](http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=24963) (don't choose a technology preview, use the executable offered as current release)
- Run it (it install the bash and sets up the path, adding the MinGW executables to the path inherited from Windows)
- You get an icon that starts a bash
- Start the bash and navigate to where you downloaded the source code for the native launcher
- Enter the command 'scons' and watch it compile the sources.

Compiling with the Cygwin or MinGW GCC produces executables that depend only dlls that are found on Windows by default. If you compile with Visual Studio, you will need an extra dll that may or may not be found on a particular windows system. The dll you need depends on the Visual Studio version, see [here](#) for details.

Try running the generated executable - if there's no complaint about a missing dll, you're fine.

## **Usage**

To use the native launcher, you need to either place the executable in the bin directory of groovy installation OR set the GROOVY\_HOME environment variable to point to your groovy installation.

The launcher primarily tries to find the groovy installation by seeing whether it is sitting in the bin directory of one. If not, it resorts to using GROOVY\_HOME environment variable. Note that this means that GROOVY\_HOME environment variable does not need to be set to be able to run groovy.

## ***Finding java installation***

The native launcher uses the following order to look up java installation to use:

1. user provided java home (using the -jh / --javahome parameter)
2. java installation pointed to by JAVA\_HOME environment variable
3. java installation found by seeing where java executable can be found on PATH (symlinks are followed to find the actual executable)
4. java installation marked as the current version in windows registry (value of "CurrentVersion" in keys
  - \\HKEY\_LOCAL\_MACHINE\SOFTWARE\JavaSoft\Java Development Kit
  - \\HKEY\_LOCAL\_MACHINE\SOFTWARE\JRockit\Java Development Kit
  - \\HKEY\_LOCAL\_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment
  - \\HKEY\_LOCAL\_MACHINE\SOFTWARE\JRockit\Java Runtime Environment
5. hard coded "/System/Library/Frameworks/JavaVM.framework" (os-x only)

To put it another way - JAVA\_HOME does not need to be set.

## ***Parameters***

The native launcher accepts all the same parameters as the .bat / shell script launchers, and a few others

on top of that. For details, type

```
groovy -h
```

#### JVM parameters

Any options not recognized as options to groovy are passed on to the jvm, so you can e.g. do

```
groovy -Xmx250m myscript.groovy
```

The -client (default) and -server options to designate the type of jvm to use are also supported, so you can do

```
groovy -Xmx250m -server myscript.groovy
```

Note that no aliases like -hotspot, -jrockit etc. are accepted - it's either -client or -server

You can freely mix jvm parameters and groovy parameters. E.g. in the following -d is param to groovy and -Dmy.prop=foo / -Xmx200m are params to the jvm:

```
groovy -Dmy.prop=foo -d -Xmx200m myscript.groovy
```

#### JAVA\_OPTS

The environment variable JAVA\_OPTS can be used to set jvm options you want to be in effect every time you run groovy, e.g. (win example)

```
set JAVA_OPTS=-Xms100m -Xmx200m
```

You can achieve the same effect by using environment variable JAVA\_TOOL\_OPTIONS, see <http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/jvmti.html#tooloptions> and [http://java.sun.com/j2se/1.5/pdf/jdk50\\_ts\\_guide.pdf](http://java.sun.com/j2se/1.5/pdf/jdk50_ts_guide.pdf)

Note that if you set the same option from the command line that is already set in JAVA\_OPTS, the one given on the command line overrides the one given in JAVA\_OPTS.

#### Paths on Cygwin

By default, the Windows version of the native launcher only understands Windows style paths if compiled using the Microsoft compiler or the MinGW GCC. If you compile using Cygwin GCC then by default, Cygwin and Windows style paths are understood. The variable cygwinsupport controls the behaviour. If you need to explicitly set whether the Cygwin path code is included in the build then you can set an option on the command line:

```
scons cygwinsupport=False
```

allowed values are True and False. Alternatively if you want to set the value explicitly for every build you can add a line like:

```
cygwinsupport = False
```

to the file local.build.options in the same directory as the SConstruct file.

Cygwin path support is a little experimental, but there are no known problem at the moment. If you use it, could you please report back success or any problems via the Groovy user mailing list.

#### *groovy.exe and groovyw.exe on Windows*

Similarly to java.exe and javaw.exe on a jdk, the build process produces groovy.exe and groovyw.exe on windows. The difference is the same as w/ java.exe and javaw.exe - groovy.exe requires a console and will launch one if it is not started in a console, whereas groovyw.exe has no console (and is usually used to start apps w/ their own gui or that run on the background).

## Windows file association

If you want to run your groovy scripts on windows so that they seem like any other commands (i.e. if you have myscript.groovy on your PATH, you can just type myscript), you have to associate groovy script files with the groovy executable. If you use the groovy windows installer it will do this for you. Otherwise, do as follows:

- add .groovy to PATHEXT environment variable
- make changes in windows registry as follows
- run regedit.exe
- create a new key HKEY\_CLASSES\_ROOT\.groovy and give it the value groovyFile
- create HKEY\_CLASSES\_ROOT\groovyFile
- under that, create HKEY\_CLASSES\_ROOT\groovyFile\Shell and give it value open
- under that, create HKEY\_CLASSES\_ROOT\groovyFile\Shell\open\command and give it value (adjust according to your groovy location) "c:\programs\groovy-1.0\bin\groovy.exe" "%1" %\*

## Why?

Why have a native launcher, why aren't the startup scripts (groovy.bat, groovy.sh) sufficient? Here are some reasons:

- it solves an [open bug](#) : return value of groovy (on windows) is always 0 no matter what happens in the executed script ( even if you call System.exit(1) ). Granted, this could be solved by editing the launch scripts also.
- it is slightly faster than the corresponding .bat / shell script
- you can mix jvm params and groovy params, thus making it easier and more natural to e.g. reserve more memory for the started jvm.
- the process will be called "groovy", not "java". Cosmetic, yes, but still nice. = )
- fixes the problems there have been w/ the .bat launcher and paths w/ whitespace
- works better on cygwin than the posix launcher script
- on Linux, you can't use an interpreted script as a #! interpreter, because of a kernel bug

Also, the launcher has been written so that the source can be used to easily create a native launcher for any Java program.

## Known issues

- Using -server option on Solaris crashes the jvm. This is due to native library incompatibilities. A workaround is to modify LD\_LIBRARY\_PATH to contain the libs needed by server jvm first, e.g.

```
LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/sparc/server:$JAVA_HOME/jre/lib/sparc:$LD_LIBRARY_PATH groovy
-server myscript.groovy
```

## Help wanted

If you have expertise with any of the following and want to help, please email me at antti dot karanta (at) hornankuusi dot fi:

- If you are running on an OS that is not yet supported, please contact me and we'll make it work. You do not need C expertise, I'll just ask you some questions about the environment, then you compile after I've made the changes and make sure it works. Examples of environments I'd like someone who has them to help me

out with: Linux on non-x86 hardware, Solaris on x86, HP-UX, FreeBSD...

- Try running on Windows 2000 and tell me how it goes.
- If you are on an IBM system and have IBM JVM (<http://www.ibm.com/developerworks/java/jdk/>) installed (or are willing and able to install it), try running the native launcher with it (e.g. using --javahome parameter). Tell me how it goes. Also, I'd very much like to know where the said jvm's location is stored in windows registry (if you don't know how to look this up, ask me for help)

## Proxy-o-Matic

Proxy-o-Matic lets you create dynamic proxies fast and in an homogeneous way

### Contribution Overview

Proxy-o-Matic lets you create dynamic proxies fast and in an homogeneous way. Groovy has the option to create proxy implementations of interfaces, abstract classes and even concrete classes based on closures and maps. You can see them as a poor man's version of anonymous inner classes, but they are **not an exact replacement** as you can't qualify method calls with **this** nor **super**.

Proxies created with Proxy-o-Matic also suffer from the this/super problem but they add a couple of features that the standard proxy creation mechanism doesn't offer:

- ability to define overloaded methods
- ability to call its own methods from within
- ability to proxy more than 1 interface at a time
- ability to proxy from Expandos as well

Here are a couple of examples that show how to use Proxy-o-Matic

```
import static
org.kordamp.groovy.util.ProxyOMatic.proxy

interface Foo { String foo() }
interface Bar { String bar() }
interface FooBar extends Foo, Bar {
    String foobar()
}

def f = proxy( Foo ) {
    foo { -> "Foo" }
}
assert f instanceof Foo
assert f.foo() == "Foo"
```

```
def fb = proxy( FooBar ) {  
    foo { -> "Foo" }  
    bar { -> "Bar" }  
    foobar { -> foo() + bar() }  
}
```

```
assert fb instanceof FooBar  
assert fb.foo() == "Foo"  
assert fb.bar() == "Bar"  
assert fb.foobar() == "FooBar"
```

```
interface Fooz extends Foo {  
    String foo( String n )  
}
```

```
def fz = proxy( Fooz ) {  
    foo { -> "Foo" }  
    foo { String n -> "Foo$n".toString() }  
}
```

```
assert fz instanceof Fooz  
assert fz.foo() == "Foo"  
assert fz.foo("Groovy") == "FooGroovy"
```

```
def bf = proxy( Bar, [Foo] ) {  
    foo { -> "Foo" }  
    bar { -> "Bar" }  
}
```

```
assert bf instanceof Bar
```

```
assert bf instanceof Foo  
assert bf.foo() == "Foo"  
assert bf.bar() == "Bar"
```

Credit must be given when credit is due, in this case Proxy-o-Matic emerged from an idea Alex Tkachman pitched at the Groovy-dev mailing list, thanks Alex for the marvelous idea of a Proxy builder DSL.

#### ***Team Members***

Andres Almiray [aalmiray at users dot sourceforge dot net]

#### **Download**

[proxyomatic-0.5.jar](#)

[source](#)

#### ***Installing***

Just drop proxyomatic-<version>.jar into \$GROOVY\_HOME/lib or ~/.groovy/lib and your done.

#### ***Pre-requisites***

Have the latest stable version of Groovy installed, that's all baby!

#### **Documentation**

Using Proxy-o-Matic is pretty straight forward: call any of ProxyOMatic's proxy() methods. To achieve a DSL like usage remember to import statically `ProxyOMatic.proxy`. Proxy-o-Matic can only create proxies from interfaces for the time being, abstract/concrete classes will be supported in a following version. These are the method signatures you would need to work with

- proxy( Class type, source )
- proxy( Class type, List<Class>, source )
- proxy( Class type, Class[], source )

where source can be any of [Closure, Map, Expando]

Another thing to consider is that given the nature of closures in Groovy the following would be treated as equivalent definitions:

```
interface Baz {
    String baz( Object b )
}
def b = proxy( Baz ) {
    baz { "BAZ" }
    baz { String n -> n }
}

assert b.baz("gotcha") == "gotcha"
```

So please avoid using the default parameter and always qualify the number of parameters a closure must have.

### ***Contributing***

Please contact the team members by e-mail.

### **Community**

#### ***Mailing List(s)***

<http://groovy.codehaus.org/Mailing+Lists>

#### ***Issue tracker***

<http://jira.codehaus.org/secure/BrowseProject.jspx?id=10242>

### **Windows NSIS-Installer**

#### **A Windows-specific installation script**

that allows to create installers for the different groovy versions. You can examine the results on the [download page](#).



The installer first copies the groovy files to where the user wants,



then asks for permission to create a variable GROOVY\_HOME and to add the respective bin directory (%GROOVY\_HOME%/bin) to the path, if no reference to groovy already exists in it.



The user can decide whether to put these into the user or system environment (on Win 95 and Win 98 this information is appended to the autoexec.bat instead) i.e., whether the installation will be for the current user or for all users. Then the installer checks for the existence of JAVA\_HOME. If it doesn't exist, a message box points out the potential problem.

Next the user can decide whether or not to install the native launcher,



can decide to install additional packages,



and finally whether to associate .groovy files with the native launcher, and whether to add the extension .groovy and .gy to PATHEXT. This allows to start Groovy files from the command line directly, without having to call the groovy

explicitly.



As you can see in the screen shots (they are clickable thumbnails), currently english, german, french and spanish are supported as installation languages.

Oh, and whatever you do, do not use the command line option -russel.

The french translation has been contributed by Xavier Mehaut and Tugdual Grall, the spanish translation by Andres Almiray, and the translation to brazilian portuguese has been provided by Marcos Silva Pereira.

Thank you very much for your help.

### Silent installation

The installer supports silent installation with the command line option /S. When using this option you can influence which parts are installed using additional options:

Option	Function
/A	Install Additional Packages
/ALL	Install All
/D	Set Installation Directory (mandatory)
/F	Create File Associations
/N	Install Native Launcher
/S	Silent Install
/V	Set Variables

The option /D is mandatory and points to the installation directory. It is important to note that the /D option **must** be the last parameter on the command line and must not contain any quotes, even if the path contains spaces. Only absolute paths are supported. Use an equal sign to separate the option and the path.

Example: `groovy-1.5.6-installer-2.exe /S /A /N /D=d:\program files\groovy`

### Working with the installer source

To use the installer source to create your own binary installers, simply check it out into an Eclipse workspace.

Additionally you might want to install the [NSIS-Eclipse-Plugin](#), it helps when developing NSIS stuff.

If you have worked with NSIS-scripts already there should be no big problem understanding what is done.

The main script is the file setup.nsi, where you can find the installer program. Three additional pages are defined, Variables, NativeLauncher and FileAssociation, and for each you can find the respective .ini file containing the screen placement. Simply open them in the design editor provided by the Eclipse-Plugin, and you can see what they look like.

Two bmp-images contain the Groovy logo in different resolutions, one for the welcome page and one as header image for the other pages. The name is the game ...

Finally you have the launch scripts. These actually start the compiler with different settings for the groovy version, the groovy location (on your hard disk) and the location of the native launcher. The groovy version influences the final name of the installer.

The launch scripts are setting the following variables:

- **SOURCE\_VERSION** defines the version of the release
- **SOURCE\_DIR** is the full path to the groovy install directory
- **NATIVE\_DIR** is the full path to the native launcher
- **SCRIPTOM\_DIR** is the full path to the scriptom module
- **GANT\_DIR** is the full path to the gant module
- **GRAPHICS\_B** is the full path to the graphicsbuilder module
- **SWINGX\_B** is the full path to the swingxbuilder module
- **VERSION\_TXT** is the full path to the installed\_versions.txt (this file describes the used versions)
- **DOC\_DIR** is the full path to the doc directory

Most probably you only have to set these variables and you are happy to go.

Have fun.

If there is any problem whatsoever, either ask on the mailing lists or contact the author at

joachim.baumann\_at\_xinaris.de.

Excerpt:a Windows-specific installer for Groovy

## Windows Services

### Introduction

**Gosh** provides an alternate way to run command-line **Groovy** scripts, as well as a simplified way to write **Windows Services (WinNT Services)** using **Groovy**. **Gosh** treats scripts the same, whether they are run from the command-line or as a service (including bindings). You can debug a service by running it from the command-line, and you can also run standard command-line **Groovy** scripts. There is very little difference between a **Groovy** script that is intended to be run as a **Windows Service** and any other **Groovy** script.

**Gosh** uses [JavaService](#) to do the "heavy lifting" of running a **Windows Service**. **Gosh** provides the framework that integrates command-line and service calls with **Groovy**, and a utility to configure **JavaService**. The end result is something that is a lot easier to develop with, and ultimately a lot easier to maintain, than **Windows Services** based on pure **Java** using **JavaService**.

This project has been under development since the middle of 2007, but it still should be considered beta software. Some of the details documented here are likely to change over time. We will, of course, attempt to

maintain backward-compatibility as much as possible going forward. As of this writing, the following items still need to be addressed (these will be documented in Jira eventually):

- Need an installer. Manual setup is still a little more complex than I would like.
- Should use a native runner for command-line. Currently uses a batch file.
- Setting up a new **Windows Service** requires a batch file. Should support either a GUI or **Groovy** script.
- Classloader structure needs peer review (has proven to work well in testing).
- Dynamic-loading design (including folder structure conventions) needs peer review. Folder structure may still change.
- Dynamic loading of **DLLs** is still experimental at this point - has not been proven in non-sun JVMs.
- Better support for advanced logging. Currently only supports simple *stdout* and *stderr* capture to file.
- **Gosh** uses an older version of **JavaService.exe** to run **Windows Services**. Needs to be updated and tested. Proven stable through **Java 6u3**.
- Need to take a look at **JavaServiceWrapper** from TanukiSoftware as a possible replacement for **JavaService**, since it seems to have more features and also supports **Unix** daemons.
- Need to find a new name for the project. Can anyone do better than "**Gosh**?"

⊖ When installed, **Gosh** takes over as the default command-line runner for **Groovy**. This may break existing code.

## Requirements

The following is required to run **Gosh**:

- 32-bit **Windows/Java (x86 architecture)** - **x64** and **IA-64** architectures are not supported
- **Java 1.5** or higher
- **Groovy 1.5** or higher (included in download)

## Installation

Download the project archive and extract the files.

- 🍌 [gosh-0.1.zip](#) Build 3 (1/3/2008) - Complete build including binaries, source, documentation, and examples.

Follow the installation instructions in the `.\docs\Groovy Gosh Users Guide.pdf` file.

## Building from Source

Everything you need (except for ant) to build **Gosh** is included in the download. Navigate to `.\gosh\build` and run `make.bat`.

## Project Structure

The **Gosh** project structure is designed to accommodate complex projects. This may include projects containing multiple **JAR** libraries and **JNI DLLs**, where different versions of the same library may exist on one machine. **Gosh** also supports convenient programmable runtime configuration using - drumroll please - a **Groovy** script.

One of the key differences between **Gosh** and the **Groovy** command-line tools is the emphasis on local, dynamic (runtime) project configuration. The **Gosh** project model follows a convention similar to what you would see in an application server, like a **Tomcat** web application. The standard **Groovy** command-line tools use a global configuration file that applies to all scripts. Although these tools support dynamically changing the classpath, the support is not obvious and somewhat awkward.

Conversely, in **Gosh**, there is no global configuration at all. **JAR** files (and also **DLLs**) are local to the project, and are loaded dynamically. **Gosh** also supports a local configuration script, written in **Groovy**, that lets you conditionally define external dependencies. This makes it easy to use different library versions in different projects on the same machine, something which **Groovy** isn't particularly good at at them moment. It also makes it almost foolproof to distribute an entire **Groovy**-based application to different machines (**xcopy** or **zip** the project).

Note that you can run multiple scripts from a single **Gosh** project. That includes **Windows Services**. So you can define, for example, 5 **Windows Services** and 20 utility scripts in the same project, all using the same set of **JAR** files and **DLLs**.

A **Gosh** project consists of a root folder containing the following:

- Uncompiled **Groovy** scripts, class files, properties files, and other resources.
- (optional) A **.GROOVY-INF** folder, containing any of the following:
  - (optional) A **.lib** folder containing **JAR** files. These will automatically be added to the classloader.
  - (optional) A **.bin** folder containing additional Windows **\*.DLL** files. This path is added to the beginning of the **java.library.path**, so **DLLs** in this folder override other **DLLs** on the local machine.
  - (optional) A **.config.groovy** script. This lets you do any additional configuration, like loading external **JAR** files or setting up logging.

When you run a **Groovy** script using **Gosh**, this is what happens:

1. **Gosh** "discovers" the project root folder. **Gosh** searches upwards from the current folder, looking for a **.GROOVY-INF** folder. The project root is handled by a single instance of **GroovyClassLoader**.
  - a. If it is found, the folder containing it becomes the project root. This affects the assumed package hierarchy of the project.
  - b. If it is not found, the folder containing the script is assumed to be the project root. Note that this means you can't reliably run **Groovy** scripts from within packages unless you define **.GROOVY-INF!**
2. If it exists, **Gosh** loads all the **JAR** files in **.GROOVY-INF\lib** into the system classloader.
3. If it exists, **Gosh** adds the **.GROOVY-INF\bin** folder to the **java.library.path**.
4. If it exists, **Gosh** runs the **.GROOVY-INF\config.groovy** script (**config.groovy** is compiled and cached, so this step actually costs very little).
5. Finally, **Gosh** runs your script.

Although it looks like a lot is going on here, the logic is actually very streamlined and fast. We've taken great pains to minimize the startup time, and the performance is generally acceptable.

## Configuring a Windows Service

You can initially configure a **Windows Service** using a simple **DOS** batch file. Additional configuration is accomplished using Windows administration tools and by directly editing the **Windows Registry**. The tools are a little crude still, but none of this is very difficult. The documentation here should give you a clear understanding of what is necessary to configure and maintain a **Gosh**-based **Windows Service**.

### 1. Initial Configuration: DOS Batch File

To create (or recreate) a Windows Service, you will need to define a batch file. The batch file will call another batch file, **goshservice.bat**, with a set of parameters, as in the code example below. **Goshservice.bat** wraps the call to **JavaService.exe**, which requires relatively complex command line syntax.

```

SET
JAVASERVICE_STUB="C:\myapp\bin\MyService.exe
"
SET SERVICE_NAME="My Service"
SET
GROOVY_SCRIPT="C:\myapp\MyService.groovy"
SET LOGS_FOLDER="C:\myapp\logs\myservice"

C:\apps\gosh\javaservice\goshservice.bat
%JAVASERVICE_STUB% %SERVICE_NAME%
%GROOVY_SCRIPT% %LOGS_FOLDER%

```

The parameters you pass to **goshservice.bat** are defined as follows:

Parameter	Description
JAVASERVICE_STUB	Each service needs a renamed copy of JavaService.exe to function. The batch file makes a copy for you, in the location you specify, and uses it to run the service.
SERVICE_NAME	The name of the service. Must be unique.
GROOVY_SCRIPT	The groovy script to run as a service.
LOGS_FOLDER	Standard out and standard err streams will be written to this folder, as files. Note that each service needs a unique folder!

After you run your batch file, the new service will be accessible for additional configuration from the Windows **Services** tool. You can rerun the batch file with different parameters, but remember that you will have to redo the manual configuration steps each time.

## 2. Additional Configuration: Windows Tools

Most **Windows Services** require some additional setup. Open the **Services** tool to finish configuring the service.

- **start Settings Control Panel Administrative Tools Services**

At a very minimum, you should verify the configuration settings for the following items:

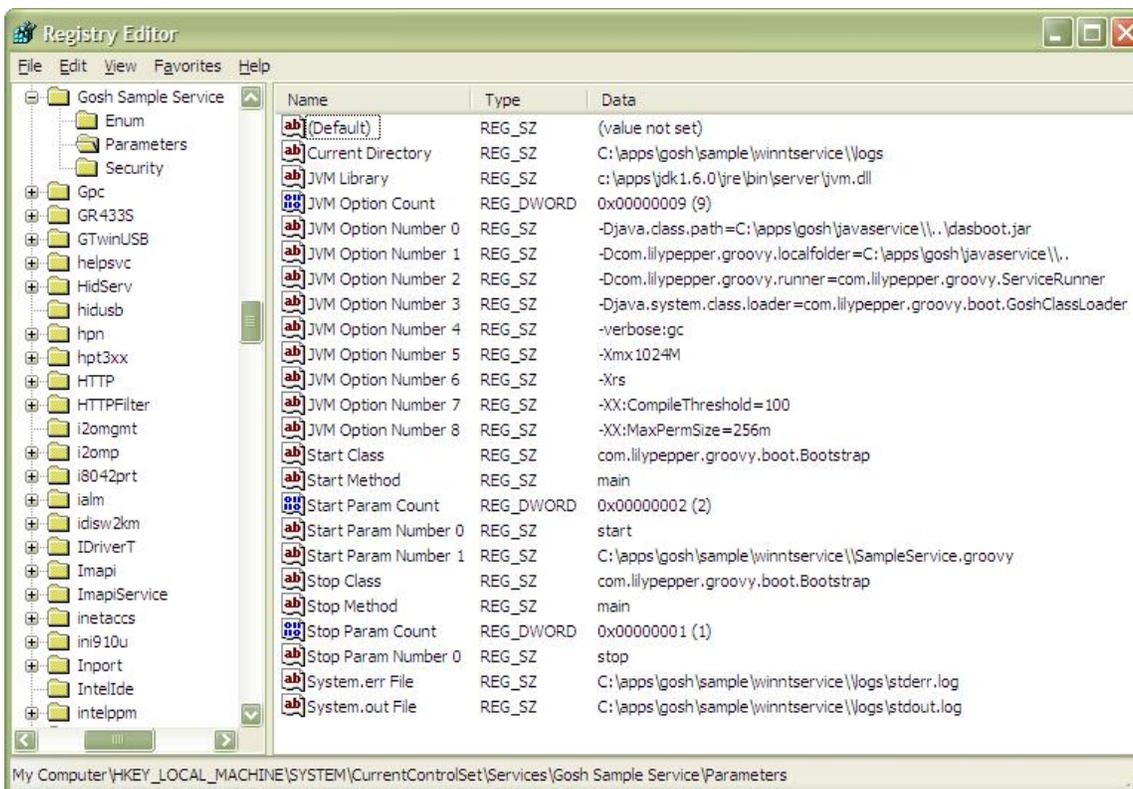
- **Description** - A description of the service.
- **Startup Type** - *Manual* or *Automatic*?
- **Log On** - The account determines what rights the service has.

Once you are done, you can start the service.

### 3. Advanced Configuration: The Windows Registry

Windows stores information about how to run services in the registry. If you want to play around with some of the service settings, and you are comfortable working directly with **RegEdit.exe**, you can save yourself some time and effort over recreating and reconfiguring the service each time. Note that when you change a registry setting, you will have to restart the service for it to become effective.

The following figure shows typical registry settings for the **Gosh Sample Service**, included in the sample project. You may notice a number of familiar parameters, such as the classpath and the maximum heap size. These can all be edited, and it is pretty obvious what each one means.



### Writing a Service

At this point, you are probably wondering what the difference is between a regular **Groovy** script and **Groovy** script that runs as a **Windows Service**.

There is a difference. A **Groovy** script does a job, and when it is finished, it ends. With a **Windows Service**, the program doesn't end until Windows asks it to. You can implement this - very simply - by creating a loop that checks a flag to exit.

```
while(!service.shutdownRequested)
{
    doSomethingServicey()
    service.sleep 5000 //Wait 5 seconds.
}
```

That is really all there is to it. The **service** instance is one of the bindings. **service.shutdownRequested** returns **true** when Windows has requested that the service be shutdown. **service.sleep** is a variation on **Thread.sleep()** that returns immediately if the service has been requested to shut down.

## Examples

There are several examples in the `.\gosh\sample` project, included in the build.

The **winntservice** example implements a complete **Windows Service**, ready to install and run. To use:

- Run **installsampleservice.bat** from a command-line.
- Using the **Services** tool, start the service.
- Check the log files to verify that the service is running.

There are also a few command-line examples in the same project.

## Bindings

**Gosh** automatically creates a number of bindings. All of these are available to your script as variables, both when run as a service and when run from the command-line.

Name	Type	Description
args	List	When run from the command-line, this contains the command-line arguments. When run as a service, this is an empty list. <b>Gosh</b> does not support passing parameters to a service.
service	ServiceRunner	Provides state information about the service. When running from a command line, the state information never changes, so you have to shut down the process manually.

systemClassLoader	GoshClassLoader	The <b>Java</b> system classloader. This classloader allows you add add <b>JAR</b> files and new classpaths at runtime.
groovyClassLoader	GroovyClassLoader	The <b>Groovy</b> classloader. Note that the parent of the <b>groovyClassLoader</b> is the <b>systemClassLoader</b> .
fs	FileSystem	Supports finding the root path of your project, creating files from the root of your project (rather than the current folder) using relative paths, and changing the <b>java.library.path</b> , among other things.
scriptFile	File	The canonical path to the initial <b>Groovy</b> script.

## ClassLoader Architecture

The classloader architecture in **Gosh** differs somewhat from the architecture found in other the other command-line runners for **Groovy**. **Gosh** replaces the **RootClassLoader** with the **GoshClassLoader**, which is a very simple override of **UrlClassLoader**.

**Gosh** is broken into two **JAR** files. The first, **dasboot.jar**, contains the minimal number of classes to boot **Gosh**. This includes the **main** entry class, the **GoshClassLoader**, and a small number of supporting classes. Once **dasboot.jar** is loaded, the system classloader gets set to an instance of **GoshClassLoader**, and all the other class libraries, including **gosh.jar**, are added to it.

Once that is done, **Gosh** creates a **GroovyClassLoader**, which is a child of the system classloader. This classloader is pointed to the root folder of the project, where it interprets uncompiled **Groovy** scripts.

In all cases, we have attempted to use the simplest, tightest classloader structure that works. This results in optimal loading and execution times.

## Limitations

The following are known limitations of **Gosh**:

- You can't typically use this to bootstrap complex application servers, like **Tomcat** and **JBoss**, due to the modified classloader structure and potential conflicts with **Groovy JAR** library dependencies. If anyone has success doing this, please contact me.

## WingSBuilder

WingsBuilder is a Groovy builder for the [wingS Framework](#)

## Contribution Overview

You can use WingSBuilder in the same fashion as [Swing Builder](#). Most of the constructs provided by the original SwingBuilder are supported in WingSBuilder, thanks to the efforts of the wingS team in following the Swing model very closely. WingSBuilder also supports wingS specific components and the ability to attach your own components through `widget()` and `container()`.

The following is an example of WingSBuilder in action

```
class GuessingGame {
  GuessingGame(){
    def randomNr = (new
Random().nextInt(10) + 1) as String
    def builder = new WingSBuilder()

    /* STitledBorder was removed in
wings-3.1
    def border =
SBorderFactory.createSTitledBorder(
SBorderFactory.createSLineBorder([0,0,255]
as Color, 2),
                                "Guessing Game")
    */
    def border =
SBorderFactory.createSLineBorder([0,0,255]
as Color, 2)
    def font = new SFont( null,
SFont.BOLD, 14 )

    def frame = builder.frame( title:
"Guessing Game" ) {
      panel( border: border){
        gridLayout( columns: 1, rows: 5,
vgap: 10)
```

```

        label("Hello World - this is
wingS (+WingSBuilder&Groovy)!",
            font: font)
        label("We want fun, so let's
play a game!" +
            "Try to guess a number
between 1 and 10.")
            textField( id: "answer" )
            button( text: "Guess!",
actionPerformed: { event ->
                    def value =
builder.answer.text
                    if( value == randomNr ){
                        builder.message.text =
"Congratulations! You guessed my number!"
                    }else{
                        builder.message.text = "No
- '${value}' is not the right number. Try
again!"
                    }
                })
            label( id: "message" )
        }
    }
}

```

```
    frame.visible = true
  }
}
```

Which is rendered in FireFox as



### ***Team Members***

Andres Almiray [aalmiray at users dot sourceforge dot net]

### **Download**

[wingsbuilder-1.0](#)

### ***Installing***

Just drop the wingsbuilder jar file along with its dependencies into your weabpp's lib directory.

### ***Pre-requisites***

WingS 3.1 and Groovy 1.5 are required to run WingSBuilder.

### **Documentation**

Alphabetical list of SwingBuilder compatible components

Element	Class	Notes
action	javax.swing.Action	same as SwingBuilder.action()
actions	Collection	same as SwingBuilder.actions()
bean	Object	same as SwingBuilder.bean()

borderLayout	SBorderLayout	
boundedRangeModel	SDefaultBoundedRangeModel	
boxLayout	SBoxLayout	
button	SButton	
buttonGroup	SButtonGroup	
cardLayout	SCardLayout	
checkbox	SCheckbox	
closureColumn	DefaultTableColumn	same as SwingBuilder.closureColumn()
container	SComponent	placeholder for any SComponent
comboBox	SCombox	Obeys 'items' to populate model
dashedLineBorder	SBorder	
desktopPane	SDesktopPane	
dialog	SDialog	
dottedLineBorder	SBorder	
emptyBorder	SBorder	
etchedBorder	SBorder	
flowLayout	SFlowLayout	
formattedTextField	SSormattedTextField	
frame	SFrame	
gbc	GridBagConstraints	same as SwingBuilder.gbc()
gridBagConstraints	GridBagConstraints	same as SwingBuilder.gridBagConstraints()
gridBagLayout	SGridBagLayout	
gridLayout	SGridLayout	

internalFrame	SInternalFrame	
label	SLabel	
lineBorder	SBorder	
list	SList	
loweredBevelBorder	SBorder	
loweredEtchedBorder	SBorder	
menu	SMenu	
menuBar	SMenuBar	
menuItem	SMenuItem	
map	java.util.Map	same as SwingBuilder.map()
optionPane	SOptionPane	
panel	SPanel	
passwordField	SPasswordField	
popupMenu	SPopupMenu	
progressBar	SProgressBar	
propertyColumn	DefaultTableColumn	same as SwingBuilder.propertyColumn()
radioButton	SRadioButton	
raisedBevelBorder	SBorder	
raisedEtchedBorder	SBorder	
scrollBar	SScrollBar	
scrollPane	SScrollPane	
separator	SSeparator	
spinnerDateModel	SSpinnerDateModel	

spinnerListModel	SSpinnerListModel	
spinnerNumberModel	SSpinnerNumberModel	
tabbedPane	STabbedPane	
table	STable	
tableModel	DefaultTableModel	same as SwingBuilder.tableModel()
tableColumn	STableColumn	
textArea	STextArea	
textField	STextField	
toggleButton	SToggleButton	
toolBar	SToolBar	
tree	STree	
widget	SComponent	placeholder for any SComponent

**Alphabetical list of WingS specific components**

<b>Element</b>	<b>Class</b>	<b>Notes</b>
anchor	SAnchor	
byteArrayIcon	SByteArrayIcon	
downloadButton	SDownloadButton	Must define 'resource' property at least other construction properties are 'text' & 'icon'
fileChooser	SFileChooser	
fileIcon	SFileIcon	Must define 'file' or 'filename' property at least other construction properties are 'mimetype' & 'extension'
flowDownLayout	SFlowDownLayout	
form	SForm	

imageIcon	SImageIcon	
nullLayout	SNullLayout	
pageScroller	SPageScroller	
pagingBoundedRangeModel	SPagingBoundedRangeModel	
popup	SPopup	
rawText	SRawTextComponent	
resourceIcon	SResourceIcon	
rootLayout	SRootLayout	
spacer	SSpacer	Must define 'width' & 'height' properties
templateLayout	STemplateLayout	

**Alphabetical list of WingX specific components**

<b>Element</b>	<b>Class</b>	<b>Notes</b>
calendar	XCalendar	
division	XDivision	
scrollablePanel	XScrollablePanel	
suggest	XSuggest	
treeTable	XTreeTable	
xpageScroller	XPageScroller	
xscrollPane	XScrollPane	
xtable	XTable	

**Developers**

Andres Almiray

**Source Control**

<http://svn.codehaus.org/groovy-contrib/wingsbuilder>

## ***Building***

WingSBuilder uses Maven2 as its build tool, which means that if you want to build your own version of WingSBuilder from source you'll need to have it installed. Follow the instructions at <http://maven.apache.org> Once Maven2 is installed you will also need to install 2 files from the wingS distribution (but it wouldn't hurt to check at <http://mvnrepository.org> if they are already there). The files are:

```
wings-3.1.jar  
wingx-3.1.jar
```

If the files are not found in any Maven2 repository the you'll need to download the source from <http://wingsframework.org>, follow the compile & install instructions, the jars should be at build/web/WEB-INF/lib Install them on your local Maven2 repo with the following command

```
mvn install:installFile -DgroupId=org.wings -Dversion=3.1 -Dpackaging=jar \  
-DartifactId=<depname> -Dfile=<jarfile>
```

where 'depname' is any of [wings, wingx] with its corresponding jarfile

## ***Contributing***

Please contact the team members by e-mail.

## **Community**

### ***Mailing List(s)***

<http://groovy.codehaus.org/Mailing+Lists>

### ***Issue tracker***

<http://jira.codehaus.org/secure/BrowseProject.jspx?id=10242>

## **XMLRPC**

### **What is the XMLRPC module?**

This is a module which allows you to create a local XML-RPC server and/or to make calls on remote XML-RPC servers.

### **What is XML-RPC?**

[XML-RPC](#) is a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It uses HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible while allowing complex data structures to be transmitted, processed and returned.

### **Using XMLRPC**

Here is an example:

#### ***The Server***

It's really easy to set up a server which provides a set of remotely callable functions.

1. Create a server object

```
import groovy.net.xmlrpc.*
import java.net.ServerSocket

def server = new XMLRPCServer()
```

2. Add some methods

```
server.echo = {return it} // the closure is now named "echo" and is
remotely callable
```

3. Start the server

```
def serverSocket = new ServerSocket( 0 ) // Open a server socket on a
free port
server.startServer(serverSocket) // Start the XML-RPC server
listening on the server socket
```

4. You're done!

### ***The Client***

It's pretty easy to make the remote calls too

1. Create a proxy object to represent the remote server

```
def serverProxy = new
XMLRPCServerProxy("http://localhost:${serverSocket.getLocalPort()}")
```

2. Call the remote method via the proxy

```
println serverProxy.echo("Hello World!")
```

3. That's all you need

### **More information**

The sources can be found here : [XML-RPC](#).

For a binary download, go to the [repository](#).

If you are using maven to download your dependencies, you won't find all the dependencies in the Maven 2 Repo yet.

The missing dependency (smack) can be manually downloaded from [here](#).

If you are using POMs, the following workaround can be used, to use smack 3.1.0 instead of 3.0.1:

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-xmlrpc</artifactId>
  <version>0.7</version>
  <!-- The version of smack is NOT in Maven
central -->
  <exclusions>
    <exclusion>
      <groupId>jivesoftware</groupId>
      <artifactId>smack</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<!-- So, we define our own smack dependency
-->
<dependency>
  <groupId>org.igniterealtime.smack</groupId>
  <artifactId>smack</artifactId>
  <version>3.1.0</version>
</dependency>
```

#### Sample scripts

- [Confluence Example](#) showing how to download a secured Confluence page.
- Another example inspired by Glen's Confluence example:

```

import groovy.net.xmlrpc.*

def c = new XMLRPCServerProxy("http://docs.codehaus.org/rpc/xmlrpc")
def token = c.confluence1.login("your_username","your_password")
// print all the code snippets from the Groovy Home page
def page = c.confluence1.getPage(token, "Groovy", "Home")
def incode = false
def separator = '////////////////////////////////////'
page.content.split('\n').each{
    if (it =~ /\{code\}/) {
        incode = !incode
        if (incode) println separator
        return
    }
    if (incode) print it
}
println separator

```

Which results in (at least around December 2007) the following:

```

////////////////////////////////////
def name='World'; println "Hello $name!"
////////////////////////////////////
class Greet {
    def name
    Greet(who) { name = who[0].toUpperCase() +
                who[1..-1] }
    def salute() { println "Hello $name!" }
}

g = new Greet('world') // create object
g.salute()             // Output "Hello World!"
////////////////////////////////////
import static org.apache.commons.lang.WordUtils.*

class Greeter extends Greet {
    Greeter(who) { name = capitalize(who) }
}

new Greeter('world').salute()
////////////////////////////////////
groovy -e "println 'Hello ' + args[0]" World
////////////////////////////////////

```

## GroovyFX

### Update: The GroovyFX Project has moved!

Please join us at our new [site](#). This page still serves as a source of documentation, but will be taken down once all of the information has been migrated.

## Module Overview

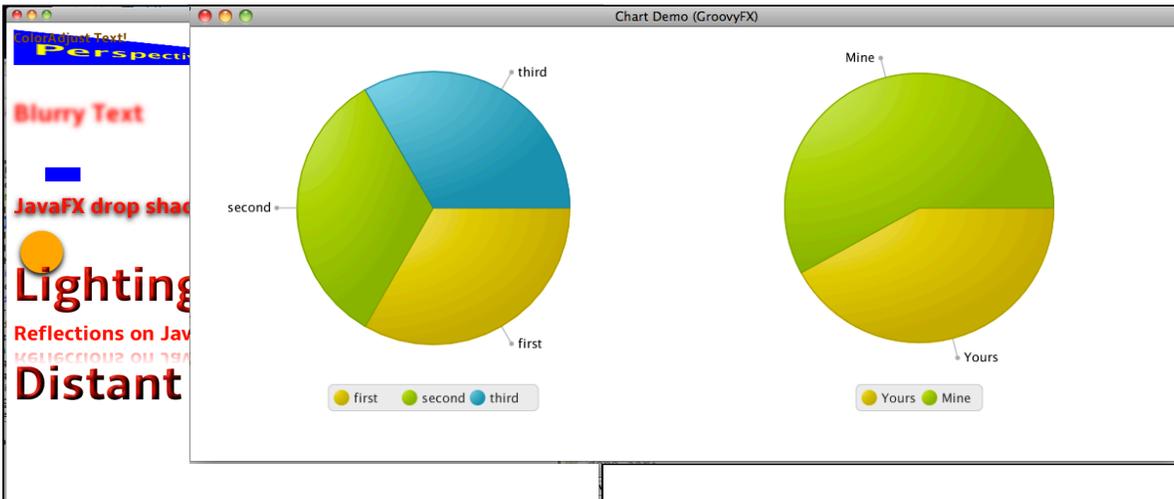
# GroovyFX provides a Groovy binding for JavaFX 2.0.

GroovyFX is an API that makes working with JavaFX in Groovy much simpler and more natural to those well-versed in Groovy. GroovyFX is focused on leveraging the Builder pattern for JavaFX applications, but more than that, it leverages some of the DSL capabilities that Groovy provides and makes the Groovy based JavaFX code easier to write and, just as important, easier to read. For example, JavaFX colors can be set using pseudo variables, like *red* or *blue*.

The primary goal of GroovyFX is to make JavaFX development simpler and more concise than what it takes in Java. This is done via numerous built-in features that Groovy provides, including the Tree Structured Language supported through Groovy's Builder framework that makes declaring a JavaFX SceneGraph more closely resemble the actual SceneGraph itself. This is done through GroovyFX's SceneGraphBuilder object, that supports all the Controls, Shapes, Effects, and other JavaFX objects, as well as support for using Groovy closures for event handling.

Features include a SceneGraphBuilder, a TimelineBuilder and Groovy DSL features to handle JavaFX objects, enumerations, binding, and paints/colors.

The following screenshots show two GroovyFX applications, one demonstrating the visual effect features available with JavaFX and the other demonstrating the JavaFX Charts. The ChartDemo.groovy source is listed below the screen snapshots.



## ChartDemo.groovy

```
package demo

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder
import javafx.collections.FXCollections
import javafx.scene.chart.PieChart

def pieData =
FXCollections.observableArrayList([new
PieChart.Data("Yours", 42), new
PieChart.Data("Mine", 58)])

GroovyFX.start {
    new SceneGraphBuilder().stage(title:
'Chart Demo (GroovyFX)', width: 1024,
height: 700, visible: true) {
        scene {
            tilePane {
                pieChart(data: [first:
0.25f, second: 0.25f, third: 0.25f])
                pieChart(data: pieData)
            }
        }
    }
}
```

- [Layout in GroovyFX](#)
- [SceneGraphBuilder](#)
- [SceneGraphBuilder - containers](#)
  - [SceneGraphBuilder.anchorPane](#)
  - [SceneGraphBuilder.borderPane](#)
  - [SceneGraphBuilder.borderPane.bottom](#)
  - [SceneGraphBuilder.borderPane.center](#)
  - [SceneGraphBuilder.borderPane.left](#)
  - [SceneGraphBuilder.borderPane.right](#)
  - [SceneGraphBuilder.borderPane.top](#)
  - [SceneGraphBuilder.flowPane](#)
  - [SceneGraphBuilder.gridPane](#)
  - [SceneGraphBuilder.gridPane.column](#)
  - [SceneGraphBuilder.gridPane.row](#)
  - [SceneGraphBuilder.group](#)
  - [SceneGraphBuilder.hbox](#)
  - [SceneGraphBuilder.scene](#)
  - [SceneGraphBuilder.stackPane](#)
  - [SceneGraphBuilder.tilePane](#)
  - [SceneGraphBuilder.vbox](#)
- [SceneGraphBuilder - controls](#)
  - [SceneGraphBuilder.accordion](#)
  - [SceneGraphBuilder.button](#)
  - [SceneGraphBuilder.checkBox](#)
  - [SceneGraphBuilder.choiceBox](#)
  - [SceneGraphBuilder.graphic](#)
  - [SceneGraphBuilder.hyperlink](#)
  - [SceneGraphBuilder.label](#)
  - [SceneGraphBuilder.listView](#)
  - [SceneGraphBuilder.passwordBox](#)
  - [SceneGraphBuilder.progressIndicator](#)
  - [SceneGraphBuilder.progressBar](#)
  - [SceneGraphBuilder.radioButton](#)
  - [SceneGraphBuilder.scrollBar](#)
  - [SceneGraphBuilder.scrollPane](#)
  - [SceneGraphBuilder.separator](#)
  - [SceneGraphBuilder.slider](#)
  - [SceneGraphBuilder.splitPane](#)
  - [SceneGraphBuilder.splitPane.dividerPosition](#)
  - [SceneGraphBuilder.tableView](#)
  - [SceneGraphBuilder.tableView.tableColumn](#)
  - [SceneGraphBuilder.tabPane](#)
  - [SceneGraphBuilder.tabPane.tab](#)
  - [SceneGraphBuilder.textArea](#)
  - [SceneGraphBuilder.textBox](#)
  - [SceneGraphBuilder.textField](#)
  - [SceneGraphBuilder.titledPane](#)
  - [SceneGraphBuilder.titledPane.content](#)
  - [SceneGraphBuilder.titledPane.title](#)
  - [SceneGraphBuilder.toggleButton](#)
  - [SceneGraphBuilder.toolBar](#)
  - [SceneGraphBuilder.tooltip](#)
- [SceneGraphBuilder - effects](#)

- [SceneGraphBuilder.blend](#)
- [SceneGraphBuilder.blend.bottomInput](#)
- [SceneGraphBuilder.blend.topInput](#)
- [SceneGraphBuilder.bloom](#)
- [SceneGraphBuilder.boxBlur](#)
- [SceneGraphBuilder.colorAdjust](#)
- [SceneGraphBuilder.colorInput](#)
- [SceneGraphBuilder.displacementMap](#)
- [SceneGraphBuilder.dropShadow](#)
- [SceneGraphBuilder.gaussianBlur](#)
- [SceneGraphBuilder.glow](#)
- [SceneGraphBuilder.imageInput](#)
- [SceneGraphBuilder.innerShadow](#)
- [SceneGraphBuilder.lighting](#)
- [SceneGraphBuilder.lighting.bumpInput](#)
- [SceneGraphBuilder.lighting.contentInput](#)
- [SceneGraphBuilder.lighting.distant](#)
- [SceneGraphBuilder.lighting.point](#)
- [SceneGraphBuilder.lighting.spot](#)
- [SceneGraphBuilder.motionBlur](#)
- [SceneGraphBuilder.perspectiveTransform](#)
- [SceneGraphBuilder.reflection](#)
- [SceneGraphBuilder.sepiaTone](#)
- [SceneGraphBuilder.shadow](#)
- [SceneGraphBuilder - input](#)
  - [empty](#)
  - [SceneGraphBuilder.onKeyPressed](#)
  - [SceneGraphBuilder.onKeyReleased](#)
  - [SceneGraphBuilder.onKeyTyped](#)
  - [SceneGraphBuilder.onMouseClicked](#)
  - [SceneGraphBuilder.onMouseDragged](#)
  - [SceneGraphBuilder.onMouseEntered](#)
  - [SceneGraphBuilder.onMouseExited](#)
  - [SceneGraphBuilder.onMouseMoved](#)
  - [SceneGraphBuilder.onMousePressed](#)
  - [SceneGraphBuilder.onMouseReleased](#)
- [SceneGraphBuilder - nodes](#)
  - [SceneGraph.text](#)
  - [SceneGraphBuilder.imageView](#)
  - [SceneGraphBuilder.imageView.image](#)
  - [SceneGraph.node](#)
  - [SceneGraph.container](#)
- [SceneGraphBuilder - paint](#)
  - [Scenegraph.hsb](#)
  - [Scenegraph.linearGradient](#)
  - [Scenegraph.radialGradient](#)
  - [Scenegraph.rgb](#)
  - [Scenegraph.stop](#)
- [SceneGraphBuilder - shapes](#)
  - [SceneGraphBuilder.arc](#)
  - [SceneGraphBuilder.circle](#)
  - [SceneGraphBuilder.cubicCurve](#)
  - [SceneGraphBuilder.ellipse](#)

- [SceneGraphBuilder.line](#)
- [SceneGraphBuilder.path](#)
- [SceneGraphBuilder.path.arcTo](#)
- [SceneGraphBuilder.path.closePath](#)
- [SceneGraphBuilder.path.cubicCurveTo](#)
- [SceneGraphBuilder.path.hLineTo](#)
- [SceneGraphBuilder.path.lineTo](#)
- [SceneGraphBuilder.path.moveTo](#)
- [SceneGraphBuilder.path.quadCurveTo](#)
- [SceneGraphBuilder.path.vLineTo](#)
- [SceneGraphBuilder.polygon](#)
- [SceneGraphBuilder.polyline](#)
- [SceneGraphBuilder.quadCurve](#)
- [SceneGraphBuilder.rectangle](#)
- [SceneGraphBuilder.SVGPath](#)
- [SceneGraphBuilder - stage](#)
  - [SceneGraphBuilder.popup](#)
  - [SceneGraphBuilder.stage](#)
- [SceneGraphBuilder - transforms](#)
  - [SceneGraphBuilder.affine](#)
  - [SceneGraphBuilder.rotate](#)
  - [SceneGraphBuilder.scale](#)
  - [SceneGraphBuilder.shear](#)
  - [SceneGraphBuilder.translate](#)
- [SceneGraphBuilder - web](#)
  - [SceneGraph.htmlEditor](#)
  - [SceneGraph.webEngine](#)
  - [SceneGraph.webView](#)
- [TimelineBuilder](#)
  - [TimelineBuilder.timeline](#)
  - [TimelineBuilder.timeline.at](#)
  - [TimelineBuilder.timeline.at.change](#)

## Layout in GroovyFX

### *Layout Constraints*

The following layout constraint properties can be applied to any Node in the SceneGraphBuilder. GroovyFX will take care of setting them up as appropriate depending on the layout pane in which the Node is placed.

Property	Type	Applies To	Implemented
topAnchor	Double	AnchorPane	✓
bottomAnchor	Double	AnchorPane	✓
leftAnchor	Double	AnchorPane	✓
rightAnchor	Double	AnchorPane	✓

margin	Insets	HBox, VBox, GridPane, TilePane, StackPane, FlowPane	✓
row (or rowIndex)	Integer	GridPane	✓
column (or columnIndex)	Integer	GridPane	✓
rowSpan	Integer	GridPane	✓
columnSpan	Integer	GridPane	✓
hgrow	Priority	GridPane, HBox	✓
vgrow	Priority	GridPane, VBox	✓
halignment	HPos	GridPane	✓
valignment	VPos	GridPane	✓
alignment	Pos	TilePane, StackPane, BorderPane	✓

## SceneGraphBuilder

### SceneGraphBuilder

The SceneGraphBuilder is the main helper class for doing Groovy Markup for JavaFX 2.0.

The following sections describe some of the generalized behavior supported in the SceneGraphBuilder on all nodes and attributes defined in the MarkUP.

### Fonts

All font attributes may of course be set with a JavaFX Font object ([javafx.scene.text.Font](#)). In addition, fonts may be defined using the JavaFX CSS styles for "-fx-font" or "-fx-font-size". If a font size string is presented, then the default Font will be used with that size. (see [CSS Reference Guide](#).)

```
text( font: "32pt", text: "This is Text")
button( font: "16pt Courier" text: "Push
Here" )
```

### Paints/Colors

All color and paint attributes may be set with one of the JavaFX Paint or Color objects ([javafx.scene.paint.LinearGradient](#), [javafx.scene.paint.RadialGradient](#) and [javafx.scene.paint.Color](#)). In addition, colors may be set using the pseudo color variables, such as *red*, *green*, *blue*, *etc*. Colors may also be defined as a web string such as "#333",

"cyan", etc. Colors may also be defined using the JavaFX CSS styles for colors, linear, and radial gradients. (see [JavaFX CSS Reference Guide](#)) Also see [SceneGraphBuilder - paint](#) for more details.

```
scene(fill: hsb(128, 0.5, 0.5, 0.5))
circle(centerX: 50, centerY: 50, radius: 25,
fill: rgb(0, 0, 255))
rectangle(x: 100, y: 50, width: 50, height:
50, fill: red)
rectangle(x: 100, y: 50, width: 50, height:
50, fill: "#333")
rectangle(x: 100, y: 50, width: 50, height:
50, fill: "linear (0%,0%) to (0%,100%) stops
(0%,gray) (100%,black)")
```

#### Enumerated Values

All Java Enumerations may be set using their string equivalents. For example, if an attribute field is of type `javafx.geometry.Pos`, then the value may be set with one of the string values, "center", "bottom\_left", etc.

There are some special cases where Object instances are used rather than enumerations. For example the `javafx.scene.Cursor` class contains numerous singleton instances for the various cursors that may be set on the scene. Also, there are special pseudo variables for the Orientation enumeration, *horizontal* and *vertical*.

```
scene(cursor: "OPEN_HAND") //
javafx.scene.Cursor static member.
sg.stage(style: "transparent") //
javafx.stage.StageStyle enumeration
scrollBar(min: 0, max: 100, value: 50,
orientation: horizontal) //
javafx.geometry.Orientation enumeration
```

#### Geometry

The Geometry objects in the `javafx.geometry` package, `Point2D`, `Point3D`, `Dimension2D`, `Rectangle2D`, and `BoundingBox`, may be set using numbered lists. For the 2D classes, 2 numbers must be provided, and for the 3D classes, 3 numbers must be provided. `BoundingBox` may be either 2D, requiring 4 values, or 3D, requiring 6 values. A `Rectangle2D` may also take an empty list or the "EMPTY" string as shown in the previous section on Insets.

```
rotate(angle: 90, axis: [1,1,0])  
imageView(viewport: [0,0, 400, 400])
```

### *Insets*

The `javafx.geometry.Insets` class may be instantiated by using the `javafx` class, or by using a Groovy number list. If a number list is used, it may contain, one, two, or four numbers. If one number is presented then all sides of the `Insets` will be set to that value. If two numbers are presented then the first number will be used to set the top and bottom side of the `Inset`, and the second number will be used to set the left and right side of the `Inset`. If four numbers are present in the list, the numbers will be assigned to the top, right, bottom and left in order. An empty list or the String "empty" will map to the `Insets.EMPTY` object.

```
stackPane(style: "-fx-background-color:  
burlywood", padding: 20)  
stackPane(style: "-fx-background-color:  
burlywood", padding: [10])  
stackPane(style: "-fx-background-color:  
burlywood", padding: [50, 25])  
stackPane(style: "-fx-background-color:  
burlywood", padding: [0, 50, 0, 0])  
stackPane(style: "-fx-background-color:  
burlywood", padding: [])  
stackPane(style: "-fx-background-color:  
burlywood", padding: "EMPTY")
```

### **ToggleGroups**

`ToggleGroups` are identified with a unique string across the `SceneGraphBuilder` context. To tie a toggle type button to a specific `ToggleGroup`, use the special attribute `toggleGroup` on the `Toggle` type button. Buttons with the same `ToggleGroup` names will be assigned the same `ToggleGroup` instance.

```
scene(fill: hsb(128, 0.5, 0.5, 0.5), root:
group()) {
    toggleButton (
        layoutX: 25,
        layoutY: 300,
        font: "16pt Courier",
        text: "One",
        selected: true,
        toggleGroup: "Group1"
    )
    toggleButton (
        layoutX: 125,
        layoutY: 300,
        font: "16pt Courier",
        text: "Two",
        selected: true,
        toggleGroup: "Group1"
    )
}
```

## SceneGraphBuilder - containers

Various containers supported by SceneGraphBuilder:

- [SceneGraphBuilder.anchorPane](#)
- [SceneGraphBuilder.borderPane](#)
- [SceneGraphBuilder.borderPane.bottom](#)
- [SceneGraphBuilder.borderPane.center](#)
- [SceneGraphBuilder.borderPane.left](#)
- [SceneGraphBuilder.borderPane.right](#)
- [SceneGraphBuilder.borderPane.top](#)
- [SceneGraphBuilder.flowPane](#)
- [SceneGraphBuilder.gridPane](#)
- [SceneGraphBuilder.gridPane.column](#)
- [SceneGraphBuilder.gridPane.row](#)
- [SceneGraphBuilder.group](#)

- [SceneGraphBuilder.hbox](#)
- [SceneGraphBuilder.scene](#)
- [SceneGraphBuilder.stackPane](#)
- [SceneGraphBuilder.tilePane](#)
- [SceneGraphBuilder.vbox](#)

## SceneGraphBuilder.anchorPane

### *Generated Object*

javafx.scene.layout.AnchorPane

### *Attributes*

This node accepts the attributes for a `javafx.scene.layout.AnchorPane` class. See the JavaDoc for [javafx.scene.layout.AnchorPane](#).

### *Content*

May contain `javafx` Nodes. Each node may have attributes for **topAnchor**, **bottomAnchor**, **leftAnchor** and **rightAnchor** that contains a value representing the offset to the AnchorPane's edge.

### *Usage*

Defines a container that lays out its children anchoring each node to the AnchorPane's edges.

### *Examples*

```

import groovyx.javafx.GroovyFX
import javafx.scene.paint.Color;
import groovyx.javafx.SceneGraphBuilder
import javafx.scene.control.*;

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "AnchorPane Example
(Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            anchorPane {
                button(text: "ONE",
topAnchor: 10, bottomAnchor: 10,
                    rightAnchor: 110,
leftAnchor: 10)
                button(text: "TWO",
rightAnchor: 10, topAnchor: 10)
            }
        }
    }
});

```

SceneGraphBuilder.borderPane

[be sure to tile the page <BuilderName>.<node> i.e., SwingBuilder.frame]

*Generated Object*

javafx.scene.layout.BorderPane

#### *Attributes*

This node accepts the attributes for a `javafx.scene.layout.BorderPane` class. See the JavaDoc for [javafx.scene.layout.BorderPane](#).

#### *Content*

May contain a **top**, **bottom**, **right**, **left**, or **center** node. It may also contain a `javafx` Node that will be placed in the center region of the `BorderPane`.

#### *Usage*

Defines a container that lays out its children along specified borders.

#### *Examples*

```
import groovyx.javafx.GroovyFX
import javafx.scene.paint.Color;
import groovyx.javafx.SceneGraphBuilder
import javafx.scene.control.*;

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "BorderPane Example
(Groovy)",
        width: 650, height: 450,
        visible: true,
    ) {
        scene(fill: lightgreen) {
            borderPane() {
                top(align: "center", margin:
[10, 0, 10, 0]) {
                    button(text: "top")
                }
            }
        }
    }
})
```

```
        right(align: "center",
margin: [0, 10, 0, 1]) {
            button(text: "right")
        }
        left(align: "center",
margin: [0, 10]) {
            button(text: "left")
        }
        bottom(align: "center",
margin: [10, 0]) {
            button(text: "bottom")
        }
        /** center example, either
construct, center() or standalone node is
treated the same.  */
        //center(align: "center") {
        label(text: "center")
        //}
    }
```

```
    }  
  }  
});
```

## SceneGraphBuilder.borderPane.bottom

### *Generated Object*

Specifies a node and properties for assigning a node to the bottom border region in a borderPane.

### *Attributes*

**align** the alignment within the region, values can either be a `javafx.geometry.Pos` object, or a string that maps the the enumerations defined for the `Pos` object. see [javafx.geometry.Pos](#)

**margin** the margin to surround the node within the borderPane region. This contains a `javafx.geometry.Insets` object, or a list of numbers that contain the top, right, bottom, and left margins. If only one number is provided then all sides of the margin will be set to that number. If 2 numbers are provided the top and bottom will use the first number and left and right will use the second number. see [javafx.geometry.Insets](#)

### *Content*

This node contains one `javafx.scene.Node`.

### *Usage*

Sets the node at the bottom region of the borderPane.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "BorderPane Example  
(Groovy)",  
        width: 650, height: 450,
```

```
        visible: true,
    ) {
        scene(fill: lightgreen) {
            borderPane() {
                top(align: "center", margin:
[10, 0, 10, 0]) {
                    button(text: "top")
                }
                right(align: "center",
margin: 10) {
                    button(text: "right")
                }
                left(align: "center",
margin: [0, 10]) {
                    button(text: "left")
                }
                bottom(align: "center",
margin: [10, 0]) {
                    button(text: "bottom")
                }
                center(align: "center") {
                    label(text: "center")
                }
            }
        }
    }
```

```
    }  
  }  
});
```

## SceneGraphBuilder.borderPane.center

### *Generated Object*

Specifies a node and properties for assigning a node to the center border region in a borderPane.

### *Attributes*

**align** the alignment within the region, values can either be a `javafx.geometry.Pos` object, or a string that maps the the enumerations defined for the `Pos` object. see [javafx.geometry.Pos](#)

**margin** the margin to surround the node within the borderPane region. This contains a `javafx.geometry.Insets` object, or a list of numbers that contain the top, right, bottom, and left margins. If only one number is provided then all sides of the margin will be set to that number. If 2 numbers are provided the top and bottom will use the first number and left and right will use the second number. see [javafx.geometry.Insets](#)

### *Content*

This node contains one `javafx.scene.Node`.

### *Usage*

Sets the node at the center region of the borderPane.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "BorderPane Example  
(Groovy)",  
        width: 650, height: 450,
```

```
        visible: true,
    ) {
        scene(fill: lightgreen) {
            borderPane() {
                top(align: "center", margin:
[10, 0, 10, 0]) {
                    button(text: "top")
                }
                right(align: "center",
margin: 10) {
                    button(text: "right")
                }
                left(align: "center",
margin: [0, 10]) {
                    button(text: "left")
                }
                bottom(align: "center",
margin: [10, 0]) {
                    button(text: "bottom")
                }
                center(align: "center") {
                    label(text: "center")
                }
            }
        }
    }
```

```
    }  
  }  
});
```

## SceneGraphBuilder.borderPane.left

### *Generated Object*

Specifies a node and properties for assigning a node to the left border region in a borderPane.

### *Attributes*

**align** the alignment within the region, values can either be a `javafx.geometry.Pos` object, or a string that maps the the enumerations defined for the `Pos` object. see [javafx.geometry.Pos](#)

**margin** the margin to surround the node within the borderPane region. This contains a `javafx.geometry.Insets` object, or a list of numbers that contain the top, right, bottom, and left margins. If only one number is provided then all sides of the margin will be set to that number. If 2 numbers are provided the top and bottom will use the first number and left and right will use the second number. see [javafx.geometry.Insets](#)

### *Content*

This node contains one `javafx.scene.Node`.

### *Usage*

Sets the node at the left region of the borderPane.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "BorderPane Example  
(Groovy)",  
        width: 650, height: 450,
```

```
        visible: true,
    ) {
        scene(fill: lightgreen) {
            borderPane() {
                top(align: "center", margin:
[10, 0, 10, 0]) {
                    button(text: "top")
                }
                right(align: "center",
margin: 10) {
                    button(text: "right")
                }
                left(align: "center",
margin: [0, 10]) {
                    button(text: "left")
                }
                bottom(align: "center",
margin: [10, 0]) {
                    button(text: "bottom")
                }
                center(align: "center") {
                    label(text: "center")
                }
            }
        }
    }
```

```
    }  
  }  
});
```

## SceneGraphBuilder.borderPane.right

### Generated Object

Specifies a node and properties for assigning a node to the right border region in a borderPane.

### Attributes

**align** the alignment within the region, values can either be a `javafx.geometry.Pos` object, or a string that maps the the enumerations defined for the `Pos` object. see [javafx.geometry.Pos](#)

**margin** the margin to surround the node within the borderPane region. This contains a `javafx.geometry.Insets` object, or a list of numbers that contain the top, right, bottom, and left margins. If only one number is provided then all sides of the margin will be set to that number. If 2 numbers are provided the top and bottom will use the first number and left and right will use the second number. see [javafx.geometry.Insets](#)

### Content

This node contains one `javafx.scene.Node`.

### Usage

Sets the node at the right region of the borderPane.

### Examples

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "BorderPane Example  
(Groovy)",  
        width: 650, height: 450,
```

```
        visible: true,
    ) {
        scene(fill: lightgreen) {
            borderPane() {
                top(align: "center", margin:
[10, 0, 10, 0]) {
                    button(text: "top")
                }
                right(align: "center",
margin: 10) {
                    button(text: "right")
                }
                left(align: "center",
margin: [0, 10]) {
                    button(text: "left")
                }
                bottom(align: "center",
margin: [10, 0]) {
                    button(text: "bottom")
                }
                center(align: "center") {
                    label(text: "center")
                }
            }
        }
    }
```

```
    }  
  }  
});
```

## SceneGraphBuilder.borderPane.top

### *Generated Object*

Specifies a node and properties for assigning a node to the top border region in a borderPane.

### *Attributes*

**align** the alignment within the region, values can either be a `javafx.geometry.Pos` object, or a string that maps the the enumerations defined for the `Pos` object. see [javafx.geometry.Pos](#)

**margin** the margin to surround the node within the borderPane region. This contains a `javafx.geometry.Insets` object, or a list of numbers that contain the top, right, bottom, and left margins. If only one number is provided then all sides of the margin will be set to that number. If 2 numbers are provided the top and bottom will use the first number and left and right will use the second number. see [javafx.geometry.Insets](#)

### *Content*

This node contains one `javafx.scene.Node`.

### *Usage*

Sets the node at the top region of the borderPane.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "BorderPane Example  
(Groovy)",  
        width: 650, height: 450,
```

```
        visible: true,
    ) {
        scene(fill: lightgreen) {
            borderPane() {
                top(align: "center", margin:
[10, 0, 10, 0]) {
                    button(text: "top")
                }
                right(align: "center",
margin: 10) {
                    button(text: "right")
                }
                left(align: "center",
margin: [0, 10]) {
                    button(text: "left")
                }
                bottom(align: "center",
margin: [10, 0]) {
                    button(text: "bottom")
                }
                center(align: "center") {
                    label(text: "center")
                }
            }
        }
    }
```

```
}  
}  
});
```

## SceneGraphBuilder.flowPane

### *Generated Object*

javafx.scene.layout.FlowPane

### *Attributes*

This node accepts the attributes for a javafx.scene.layout.FlowPane class. See the JavaDoc for [javafx.scene.layout.FlowPane](#).

### *Content*

May contain javafx Nodes.

### *Usage*

Defines a container that lays out its children in a flow that wraps at the flowPane's boundary.

### *Examples*

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "FlowPane Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            flowPane(hgap: 10, alignment:
"center" ) {
                button(text: "ONE")
                button(text: "TWO")
                button(text: "THREE")
                button(text: "FOUR")
            }
        }
    }
});

```

SceneGraphBuilder.gridPane

Generated Object

javafx.scene.layout.GridPane

*Attributes*

This node accepts the attributes for a javafx.scene.layout.GridPane class. See the JavaDoc for [javafx.scene.layout.GridPane](#).

*Content*

May contain javafx Nodes. Each node may have a grid constraint. Grid Pane may also contain row and column nodes.

*Usage*

Defines a container that lays out its children within a flexible grid of rows and columns.

*Examples*

```
import groovyx.javafx.GroovyFX
import javafx.scene.paint.Color;
import groovyx.javafx.SceneGraphBuilder
import javafx.scene.control.*;

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "GridPane Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            gridPane(hgap: 4, vgap: 4,
padding: [18,18,18,18], alignment: "center",
gridLinesVisible: true) {
                label (text: "Name: ") {
                    constraint(row: 0,
column: 0, halignment: "right")
                }
                label (text: "Jim
Clarke") {
                    constraint(row: 0,
column:1, columnSpan: 5,halignment: "left")
```

```

        }
        label (text: "Address:")
    {
        constraint(row: 1,
column:0, halignment: "right")
    }
    label (text: "123 Main
St") {
        constraint(row:
1,column:1, columnSpan: 5,halignment:
"left")
    }
    label (text: "City:") {
        constraint(row:
2,column:0, halignment: "right")
    }
    label (text: "Orlando")
    {
        constraint(row:
2,column:1, halignment: "left")
    }
    label (text: "State:") {
        constraint(row:
2,column:2, halignment: "right")
    }
    label (text: "FL") {
        constraint(row:
2,column:3, halignment: "left", hgrow:
"never")
    }
    label (text: "Zipcode:")

```

```

{
    constraint(row:
2,column:4, halignment: "right")
    }
    label (text: "32817") {
        constraint(row:
2,column:5, halignment: "left", hgrow:
"never")
    }
    button(text: "Register")
}
    constraint(row:
3,0,columnSpan: 6, halignment: "center")
    }
    row(index: 4) {
        label (text: "one")
        label (text: "two")
        label (text:
"three")
    }
    column(index: 6) {
        label (text: "one")
        label (text: "two")
        label (text:
"three")
    }
}

```

```
    }  
  }  
});
```

## SceneGraphBuilder.gridPane.column

### *Generated Object*

Specifies nodes and properties for assigning nodes to a column in the GridPane.

### *Attributes*

**index** the column index

### *Content*

This node contains one or more javafx Nodes.

### *Usage*

Sets the nodes for a column in the GridPane.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "GridPane Row Example  
(Groovy)",  
        width: 650, height:450,  
        visible: true,  
    ) {  
        scene(fill: lightgreen ) {
```

```
        gridPane(hgap: 4, vgap: 4,
padding: [18,18,18,18], alignment: "center",
gridLinesVisible: true) {
            column(index: 0) {
                label (text: "one")
                label (text: "two")
                label (text:
"three")
            }
            column(index: 1) {
                label (text: "four")
                label (text: "five")
                label (text: "six")

                label (text:
"seven")
            }
        }
```

```
    }  
  }  
});
```

## SceneGraphBuilder.gridPane.row

### *Generated Object*

Specifies nodes and properties for assigning nodes to a row in the GridPane.

### *Attributes*

**index** the row index

### *Content*

This node contains one or more javafx Nodes.

### *Usage*

Sets the nodes for a row in the GridPane.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
  
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
  
    sg.stage(  
        title: "GridPane Row Example  
(Groovy)",  
        width: 650, height:450,  
        visible: true,  
    ) {  
        scene(fill: lightgreen ) {
```

```
        gridPane(hgap: 4, vgap: 4,
padding: [18,18,18,18], alignment: "center",
gridLinesVisible: true) {
            row(index: 0) {
                label (text: "one")
                label (text: "two")
                label (text:
"three")
            }
            row(index: 1) {
                label (text: "four")
                label (text: "five")
                label (text: "six")

                label (text:
"seven")
            }
        }
```

```
}  
}  
});
```

## SceneGraphBuilder.group

### *Generated Object*

javafx.scene.Group

### *Attributes*

This node accepts the attributes for a javafx.scene.Group class. See the JavaDoc for [javafx.scene.Group](#).

### *Content*

May contain javafx Nodes.

### *Usage*

Defines a container that children that are rendered in order whenever this node is rendered.

### *Examples*

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Group Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            group() {
                button(text: "ONE",
layoutX: 10, layoutY: 300)
                button(text: "TWO",
layoutX: 110, layoutY: 300)
                button(text: "THREE",
layoutX: 210, layoutY: 300)
                button(text: "FOUR",
layoutX: 310, layoutY: 300)
            }
        }
    }
});

```

SceneGraphBuilder.hbox

*Generated Object*

javafx.scene.layout.HBox

*Attributes*

This node accepts the attributes for a `javafx.scene.layout.HBox` class. See the JavaDoc for [javafx.scene.layout.HBox](#).

*Content*

May contain javafx Nodes.

*Usage*

Defines a container that lays out its children horizontally.

*Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "HBox Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            hbox(spacing: 10 ) {
                button(text: "ONE")
                button(text: "TWO")
                button(text: "THREE")

                button(text: "FOUR")
            }
        }
    }
});
```

## **SceneGraphBuilder.scene**

[be sure to tile the page <BuilderName>.<node> i.e., SwingBuilder.frame]

### ***Generated Object***

javafx.scene.Scene

### ***Attributes***

This node accepts the attributes for a javafx.scene.Scene class. See the JavaDoc for [javafx.scene.Scene](#)

### ***Content***

May include stylesheets, mouse events, key events, javafx container Panes, or javafx Nodes.

### ***Usage***

Each stage must have a Scene.

### ***Examples***

```

GroovyFX.start({ primaryStage ->
    def sg = new
SceneGraphBuilder(primaryStage);

    sg.stage(
        title: "CircleT3D (Groovy)",
        visible: true,
        resizable: true
    ) {
        scene(fill: lightgreen, camera: new
PerspectiveCamera(), width: 400, height:300)
        {
            stylesheets( urls:
"file://mystyle.css" )
            onMousePressed( onEvent: {
event -> println "scene mouse press"})
            onKeyReleased( onEvent: { event
-> println "scene key" + event.text})

            circle(centerX: 200, centerY:
150, radius: 75, fill: blue,
                rotate: 30, rotationAxis:
Rotate.Y_AXIS)
        }
    }
})

```

SceneGraphBuilder.stackPane

*Generated Object*

javafx.scene.layout.StackPane

***Attributes***

This node accepts the attributes for a javafx.scene.layout.StackPane class. See the JavaDoc for [javafx.scene.layout.StackPane](#).

***Content***

May contain javafx Nodes.

***Usage***

Defines a container that lays out its children in a back-to-front stack.

***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "StackPane Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            stackPane() {
                button(text: "ONE")
                button(text: "TWO")
                button(text: "THREE")
                button(text: "FOUR")
            }
        }
    }
});

```

### SceneGraphBuilder.tilePane

#### *Generated Object*

javafx.scene.layout.TilePane

#### *Attributes*

This node accepts the attributes for a javafx.scene.layout.TilePane class. See the JavaDoc for [javafx.scene.layout.TilePane](#).

#### *Content*

May contain javafx Nodes.

### Usage

Defines a container that lays out its children in a grid of uniformly sized "tiles".

### Examples

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder()

    sg.stage(
        title: "TilePane Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            tilePane(hgap: 8, vgap: 8,
prefColumns: 2) {
                button(text: "ONE")
                button(text: "TWO")
                button(text: "THREE")
                button(text: "FOUR")
            }
        }
    }
});
```

### SceneGraphBuilder.vbox

#### Generated Object

javafx.scene.layout.VBox

#### Attributes

This node accepts the attributes for a `javafx.scene.layout.VBox` class. See the JavaDoc for [javafx.scene.layout.VBox](#).

**Content**

May contain javafx Nodes.

**Usage**

Defines a container that lays out its children vertically.

**Examples**

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "HBox Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            vbox(spacing: 10 ) {
                button(text: "ONE")
                button(text: "TWO")
                button(text: "THREE")
                button(text: "FOUR")
            }
        }
    }
});
```

SceneGraphBuilder - controls

JavaFX Controls

## SceneGraphBuilder.accordion

### *Generated Object*

javafx.scene.control.Accordion

### *Attributes*

This node takes the attributes of the `javafx.scene.control.Accordion` class, see [javafx.scene.control.Accordion](#).

### *Content*

If the content is a tooltip, the Accordion's tooltip will be set to the Tooltip.

The content is one or more JavaFX nodes that make up the panes of the Accordion.

### *Usage*

Used to create an Accordion

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "Accordion Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      accordion ( ) {  
        titledPane(id: "t1", ) {  
          title() {  
            label(text: "Label 1")  
          }  
          content() {  
            label(text: "This is
```

```
Label 1\n\nAnd there were a few empty lines  
just there!")
```

```
}
```

```
}
```

```
titledPane(id: "t2") {  
    title() {  
        label(text: "Label 2")
```

```
}
```

```
content() {
```

```
    label(text: "This is
```

```
Label 2\n\nAnd there were a few empty lines  
just there!")
```

```
}
```

```
}
```

```
titledPane(id: "t3") {  
    title() {  
        label(text: "Label 3")
```

```
}
```

```
// this is content
```

```
    label(text: "This is Label
```

```
3\n\nAnd there were a few empty lines just  
there!")
```

```
}
```

```
}
```

```
}
```

```
}
```

```
});
```

## SceneGraphBuilder.button

### *Generated Object*

javafx.scene.control.Button

### *Attributes*

This node takes the attributes of the `javafx.scene.control.Button`. see [javafx.scene.control.Button](#).

### *Content*

If the content is a `javafx.scene.control.Tooltip` (see [javafx.scene.control.Tooltip](#)) object, then the Button's tooltip will be set to this Tooltip object.

If the content is a JavaFX Node, then the Button's graphic will be set to the JavaFX Node.

Other content is processed per the common `javafx Node` content including input events, effects and transforms.

### *Usage*

This is a JavaFX Node and accepts the common Node attributes.

### *Examples*

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            button (
                layoutX: 25,
                layoutY: 300,
                font: "16pt Courier",
                text: "This is a Button",
                onAction: { println "button
pressed"}
            )
        }
    }
});

```

**SceneGraphBuilder.checkBox**

*Generated Object*

javafx.scene.control.CheckBox

*Attributes*

This node takes the attributes of the `javafx.scene.control.CheckBox`. see [javafx.scene.control.CheckBox](#).

#### ***Content***

If the content is a `javafx.scene.control.Tooltip` (see [javafx.scene.control.Tooltip](#)) object, then the `CheckBox`'s tooltip will be set to this `Tooltip` object.

If the content is a `JavaFX Node`, then the `CheckBox`'s graphic will be set to the `JavaFX Node`.

Other content is processed per the common `javafx Node` content including input events, effects and transforms.

#### ***Usage***

This is a `JavaFX Node` and accepts the common `Node` attributes.

#### ***Examples***

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            checkBox (
                layoutX: 25,
                layoutY: 300,
                font: "16pt Courier",
                text: "Check",
                selected: true
            )
        }
    }
});

```

### SceneGraphBuilder.choiceBox

*Generated Object*

javafx.scene.control.ChoiceBox

*Attributes*

This node takes the attributes of the [javafx.scene.control.ChoiceBox](#). see [javafx.scene.control.ChoiceBox](#).

Also it takes an attribute **items** that contains a list of the ChoiceBox items.

#### ***Content***

If the content is a `javafx.scene.control.Tooltip` (see [javafx.scene.control.Tooltip](#)) object, then the ChoiceBox's tooltip will be set to this Tooltip object.

If the content is a JavaFX Node, then the ChoiceBox's graphic will be set to the JavaFX Node.

Other content is processed per the common `javafx Node` content including input events, effects and transforms.

#### ***Usage***

This is a JavaFX Node and accepts the common Node attributes.

#### ***Examples***

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            choiceBox (
                layoutX: 25,
                layoutY: 300,
                items: [ "one", "two",
"three" ]
            )
        }
    }
});

```

## SceneGraphBuilder.graphic

### *Generated Object*

javafx.scene.Node

### *Attributes*

This node does not take any attributes.

### *Content*

This node takes one JavaFX node that is used for the graphic for the following nodes:

button, checkBox, label, choiceBox, hyperlink, tooltip, radioButton, toggleButton, and tab.

```
registerFactory( 'button', lf)
  registerFactory( 'checkBox', lf)
  registerFactory( 'label', lf)
  registerFactory( 'choiceBox', lf)
  registerFactory( 'hyperlink', lf)
  registerFactory( 'tooltip', lf)
  registerFactory( 'radioButton', lf)
  registerFactory( 'toggleButton', lf)
```

#### *Usage*

Used to create a Graphic that is displayed for a Node.

#### *Examples*

```
GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
  title: "TabPane Example",
  x: 100, y: 100, width: 400, height:400,
  visible: true,
  style: "decorated",
) {

  scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
    tabPane ( ) {
      tab(text: 'Tab 1') {
        label(text: "This is Label
1\n\nAnd there were a few empty lines just
```

```
there!")
        graphic() {
            rectangle(width: 20,
height: 20, fill: red)
        }
    }
    tab(text: 'Tab 2') {
        label(text: "This is Label
2\n\nAnd there were a few empty lines just
there!")

        graphic() {
            rectangle(width: 20,
height: 20, fill: blue)
        }
    }
    tab(text: 'Tab 3') {
        label(text: "This is Label
3\n\nAnd there were a few empty lines just
there!")

        graphic() {
            rectangle(width: 20,
height: 20, fill: green)
        }
    }
}
```

```
}
```

```
});
```

## **SceneGraphBuilder.hyperlink**

### *Generated Object*

javafx.scene.control.Hyperlink

### *Attributes*

This node takes the attributes of the `javafx.scene.control.Hyperlink`. see [javafx.scene.control.Hyperlink](#).

### *Content*

If the content is a `javafx.scene.control.Tooltip` (see [javafx.scene.control.Tooltip](#)) object, then the `CheckBox`'s tooltip will be set to this `Tooltip` object.

If the content is a JavaFX Node, then the `Hyperlink`'s graphic will be set to the JavaFX Node.

Other content is processed per the common `javafx` Node content including input events, effects and transforms.

### *Usage*

This is a JavaFX Node and accepts the common Node attributes.

### *Examples*

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            hyperlink (
                layoutX: 25,
                layoutY: 300,
                text:
                "http://www.google.com"
                onAction: { println 'Go to
http://www.google.com' }
            )
        }
    }
});

```

## SceneGraphBuilder.label

*Generated Object*

javafx.scene.control.Label

*Attributes*

This node takes the attributes of the `javafx.scene.control.Label`. see [javafx.scene.control.Label](#).

***Content***

If the content is a `javafx.scene.control.Tooltip` (see [javafx.scene.control.Tooltip](#)) object, then the `CheckBox`'s tooltip will be set to this `Tooltip` object.

If the content is a `JavaFX Node`, then the `Label`'s graphic will be set to the `JavaFX Node`.

Other content is processed per the common `javafx Node` content including input events, effects and transforms.

***Usage***

This is a `JavaFX Node` and accepts the common `Node` attributes.

***Examples***

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            label (
                layoutX: 25,
                layoutY: 300,
                font: "16pt Courier",
                text: "This is a label"
            )
        }
    }
});

```

## SceneGraphBuilder.listView

*Generated Object*

javafx.scene.control.ListView

*Attributes*

This node takes the attributes of the `javafx.scene.control.ListView` see [javafx.scene.control.ListView](#).

*Content*

If the content is a tooltip, the the ListView's tooltip will be set to the Tooltip.

#### *Usage*

Used to create a vertical or horizontal list of items.

#### *Examples*

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            listView(items:
["one", "two", "three"], prefWidth: 200,
prefHeight: 400)
        }
    }
});
```

#### **SceneGraphBuilder.passwordBox**

##### *Generated Object*

javafx.scene.control.PasswordBox

##### *Attributes*

This node takes the attributes of the [javafx.scene.control.PasswordBox](#) see [javafx.scene.control.PasswordBox](#).

##### *Content*

If the content is a tooltip, the the PasswordBox's tooltip will be set to the Tooltip.

*Usage*

Used to create a password text box

*Examples*

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {
        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            passwordBox(prefWidth: 200)
        }
    }
});
```

**SceneGraphBuilder.progressIndicator**

*Generated Object*

javafx.scene.control.ProgressIndicator

*Attributes*

This node takes the attributes of the javafx.scene.control.ProgressIndicator see [javafx.scene.control.ProgressIndicator](#).

*Content*

If the content is a tooltip, the the ProgressIndicator's tooltip will be set to the Tooltip.

### Usage

Used to create a Progress Indicator

### Examples

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            pi = progressIndicator(
prefWidth: 200 )
        }
    }
});
```

### SceneGraphBuilder.progressBar

#### Generated Object

javafx.scene.control.ProgressBar

#### Attributes

This node takes the attributes of the [javafx.scene.control.ProgressBar](#) see [javafx.scene.control.ProgressBar](#).

#### Content

If the content is a tooltip, the the ProgressBar's tooltip will be set to the Tooltip.

### Usage

Used to create a Progress Bar

*Examples*

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            pb = progressBar( prefWidth: 200
)
        }
    }
});
```

**SceneGraphBuilder.radioButton**

*Generated Object*

javafx.scene.control.RadioButton

*Attributes*

This node takes the attributes of the [javafx.scene.control.RadioButton](#). see [javafx.scene.control.RadioButton](#).

*Content*

If the content is a [javafx.scene.control.Tooltip](#) (see [javafx.scene.control.Tooltip](#)) object, then the CheckBox's tooltip will be set to this Tooltip object.

If the content is a JavaFX Node, then the RadioButton's graphic will be set to the JavaFX Node.

Other content is processed per the common javafx Node content including input events, effects and transforms.

#### *Usage*

This is a JavaFX Node and accepts the common Node attributes.

#### *Examples*

```
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
    sg.stage(  
        title: "Stage Frame",  
        x: 100, y: 100, width: 400,  
height:400,  
        visible: true,  
        style: "decorated",  
        onHide: { println "Close"}  
    ) {  
  
        scene(fill: hsb(128, 0.5, 0.5, 0.5),  
parent: group()) {  
            radioButton (  
                layoutX: 25,  
                layoutY: 300,  
                font: "16pt Courier",  
                text: "Radio Button",  
                selected: true  
            )  
        }  
    }  
});
```

SceneGraphBuilder.scrollBar

*Generated Object*

javafx.scene.control.ScrollBar

*Attributes*

This node takes the attributes of the `javafx.scene.control.ScrollBar` see [javafx.scene.control.ScrollBar](#).

*Content*

If the content is a tooltip, the the Scrollbar's tooltip will be set to the Tooltip.

*Usage*

Used to create a vertical or horizontal Scrollbar

*Examples*

```
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
    sg.stage(  
        title: "Stage Frame",  
        x: 100, y: 100, width: 400,  
height:400,  
        visible: true,  
        style: "decorated",  
        onHide: { println "Close"}  
    ) {  
  
        scene(fill: hsb(128, 0.5, 0.5, 0.5),  
parent: group()) {  
            scrollbar(min: 0, max: 100,  
value: 50, orientation: "horizontal",  
prefWidth: 200 )  
        }  
    }  
});
```

SceneGraphBuilder.scrollPane

*Generated Object*

javafx.scene.control.ScrollPane

**Attributes**

This node takes the attributes of the javafx.scene.control.ScrollPane see [javafx.scene.control.ScrollPane](#).

**Content**

If the content is a tooltip, the the ScrollPane's tooltip will be set to the Tooltip.

If the content is a JavaFX node, then the ScrollPane's node attribute will be set to this node.

**Usage**

Used to create a Scroll Pane

**Examples**

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            scrollPane( prefWidth: 200,
prefHeight: 400) {
                rectangle(width: 600,
height: 600, fill: "linear (0%,0%) to
(100%,100%) stops (0.0,red) (1.0,black)")
            }
        }
    }
});

```

## SceneGraphBuilder.separator

*Generated Object*

javafx.scene.control.Separator

*Attributes*

This node takes the attributes of the [javafx.scene.control.Separator](#) see [javafx.scene.control.Separator](#).

*Content*

If the content is a tooltip, the the Separator's tooltip will be set to the Tooltip.

#### *Usage*

Used to create a vertical or horizontal Separator line.

#### *Examples*

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {
        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            separator(orientation:
"horizontal", prefWidth: 200 )
        }
    }
});
```

### **SceneGraphBuilder.slider**

#### *Generated Object*

javafx.scene.control.Slider

#### *Attributes*

This node takes the attributes of the [javafx.scene.control.Slider](#) see [javafx.scene.control.Slider](#).

#### *Content*

If the content is a tooltip, the the Slider's tooltip will be set to the Tooltip.

### Usage

Used to create a vertical or horizontal Slider

### Examples

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            slider(min: 0, max: 100, value:
50, orientation: "horizontal",
showTickMarks: true, prefWidth: 200 )
        }
    }
});
```

### SceneGraphBuilder.splitPane

#### Generated Object

javafx.scene.control.SplitPane

#### Attributes

This node takes the attributes of the [javafx.scene.control.SplitPane](#) see [javafx.scene.control.SplitPane](#).

#### Content

If the content is a tooltip, the theSplitPane's tooltip will be set to the Tooltip.

If the content is a JavaFX node, then it will be added to the SplitPane's items.

If the content is a "dividerPosition", then the values of the dividerPosition will be used to set the SplitPane's divider positions. see [SceneGraphBuilder.splitPane.dividerPosition](#)

#### *Usage*

Used to create a SplitPane

#### *Examples*

```
GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "SplitPane Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        splitPane ( orientation:
"horizontal") {
            label(text: "This is Label
1\n\nAnd there were a few empty lines just
there!") // left or top
            label(text: "This is Label
2\n\nAnd there were a few empty lines just
there!") // right or bottom
            label(text: "This is Label
3\n\nAnd there were a few empty lines just
there!")

                dividerPosition(index: 0,
position: 0.25)
```

```
        dividerPosition(index: 1,  
position: 0.50)  
        dividerPosition(index: 2,  
position: 1.0)  
    }  
}
```

```
}  
  
});
```

## SceneGraphBuilder.splitPane.dividerPosition

### *Generated Object*

Sets the divider position for the indexed divider in a SplitPane.

### *Attributes*

**index** the index integer specifying which divider to set the position.

**position** the value of the position in a range of 0.0 to 1.0.

### *Content*

There is no content for this node.

### *Usage*

Used to set the divider position for the indexed divider.

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "SplitPane Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      splitPane ( orientation:  
"horizontal") {  
        label(text: "This is Label
```

```
1\n\nAnd there were a few empty lines just  
there!") // left or top  
        label(text: "This is Label  
2\n\nAnd there were a few empty lines just  
there!") // right or bottom  
        label(text: "This is Label  
3\n\nAnd there were a few empty lines just  
there!")  
  
        dividerPosition(index: 0,  
position: 0.25)  
        dividerPosition(index: 1,  
position: 0.50)  
        dividerPosition(index: 2,  
position: 1.0)  
    }  
}
```

```
}  
  
});
```

## SceneGraphBuilder.tableView

### *Generated Object*

javafx.scene.control.TableView

### *Attributes*

This node takes the attributes of the javafx.scene.control.TableView see [javafx.scene.control.TableView](#).

### *Content*

If the content is a tooltip, the TableView's tooltip will be set to the Tooltip.

TableView takes a series of TableColumn objects to define each column of the table.

### *Usage*

Used to create a TableView

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
import javafx.scene.control.table.*;  
  
import javafx.beans.property.StringProperty;  
  
class Person {  
  
    public Person(String first, String last,  
String city, String state) {  
        setFirstName(first);  
        setLastName(last);  
    }  
}
```

```
        setCity(city);
        setState(state);
    }

    private StringProperty firstName;
    public void setFirstName(String value)
    { firstNameProperty().set(value); }
    public String getFirstName() { return
firstName == null? "" : firstName.get(); }
    public StringProperty
firstNameProperty() {
        if (firstName == null) firstName =
new StringProperty();
        return firstName;
    }

    private StringProperty lastName;
    public void setLastName(String value) {
lastNameProperty().set(value); }
    public String getLastName() { return
lastName == null? "" : lastName.get(); }
    public StringProperty
lastNameProperty() {
        if (lastName == null) lastName =
new StringProperty();
        return lastName;
    }

    private StringProperty city;
    public void setCity(String value) {
cityProperty().set(value); }
```

```
    public String getCity() { return city
== null? "" : city.get(); }
    public StringProperty cityProperty() {
        if (city == null) city = new
StringProperty();
        return city;
    }

    private StringProperty state;
    public void setState(String value) {
stateProperty().set(value); }
    public String getState() { return state
== null? "" : state.get(); }
    public StringProperty stateProperty()
{
        if (state == null) state = new
StringProperty();
        return state;
    }
}
def data = [
    new Person('Jim', 'Clarke', 'Orlando',
'Fl'),
    new Person('Jim', 'Connors', 'Long
Island', 'NY'),
    new Person('Eric', 'Bruno', 'Long
Island', 'NY'),
]

//todo cell factory
```

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Table Example (Groovy)",
        width: 650, height:450,
        visible: true,
    ) {
        scene(fill: lightgreen ) {
            tableView(items: data) {
                tableColumn(text: "First
Name", property: 'firstName')
                tableColumn(text: "Last
Name", property: 'lastName')
                tableColumn(text: "City",
property: 'city')
                tableColumn(text: "State",
property: 'state')
            }
        }
    }
}
```

```
        }  
    }  
});
```

## SceneGraphBuilder.tableView.tableColumn

### *Generated Object*

javafx.scene.control.TableColumn

### *Attributes*

This node takes the attributes of the javafx.scene.control.TableColumn see [javafx.scene.control.TableColumn](#).

### *Content*

There is no content for a TableColumn

### *Usage*

Used to create a TableColumn for use in a TableView. A TableColumn ties a property from a data set to a specific column in a TableView and defines the column header.

### *Examples*

```
import groovyx.javafx.GroovyFX  
import javafx.scene.paint.Color;  
import groovyx.javafx.SceneGraphBuilder  
import javafx.scene.control.*;  
import javafx.scene.control.table.*;  
  
import javafx.beans.property.StringProperty;  
  
class Person {  
  
    public Person(String first, String last,  
String city, String state) {  
        setFirstName(first);  
        setLastName(last);  
        setCity(city);  
    }  
}
```

```
        setState(state);
    }

    private StringProperty firstName;
    public void setFirstName(String value)
    { firstNameProperty().set(value); }
    public String getFirstName() { return
firstName == null? "" : firstName.get(); }
    public StringProperty
firstNameProperty() {
        if (firstName == null) firstName =
new StringProperty();
        return firstName;
    }

    private StringProperty lastName;
    public void setLastName(String value) {
lastNameProperty().set(value); }
    public String getLastName() { return
lastName == null? "" : lastName.get(); }
    public StringProperty
lastNameProperty() {
        if (lastName == null) lastName =
new StringProperty();
        return lastName;
    }

    private StringProperty city;
    public void setCity(String value) {
cityProperty().set(value); }
    public String getCity() { return city
```

```

== null? "" : city.get(); }
    public StringProperty cityProperty() {
        if (city == null) city = new
StringProperty();
        return city;
    }

    private StringProperty state;
    public void setState(String value) {
stateProperty().set(value); }
    public String getState() { return state
== null? "" : state.get(); }
    public StringProperty stateProperty()
{
        if (state == null) state = new
StringProperty();
        return state;
    }
}
def data = [
    new Person('Jim', 'Clarke', 'Orlando',
'Fl'),
    new Person('Jim', 'Connors', 'Long
Island', 'NY'),
    new Person('Eric', 'Bruno', 'Long
Island', 'NY'),
]

//todo cell factory

GroovyFX.start({

```

```
def sg = new SceneGraphBuilder();

sg.stage(
    title: "Table Example (Groovy)",
    width: 650, height:450,
    visible: true,
) {
    scene(fill: lightgreen ) {
        tableView(items: data) {
            tableColumn(text: "First
Name", property: 'firstName')
            tableColumn(text: "Last
Name", property: 'lastName')
            tableColumn(text: "City",
property: 'city')
            tableColumn(text: "State",
property: 'state')
        }
    }
}
```

```
    }  
  }  
});
```

## SceneGraphBuilder.tabPane

### *Generated Object*

javafx.scene.control.TabPane

### *Attributes*

This node takes the attributes of the `javafx.scene.control.TabPane` see [javafx.scene.control.TabPane](#).

### *Content*

If the content is a tooltip, the the TabPane's tooltip will be set to the Tooltip.

If the content is Tab, then the tab will be added to this TabPane.

### *Usage*

Used to create a TabPane

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "TabPane Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      tabPane ( ) {  
        tab(text: 'Tab 1') {  
          label(text: "This is Label
```

1\n\nAnd there were a few empty lines just there!")

```
        graphic() {  
            rectangle(width: 20,  
height: 20, fill: red)  
        }  
    }  
    tab(text: 'Tab 2') {  
        label(text: "This is Label
```

2\n\nAnd there were a few empty lines just there!")

```
        graphic() {  
            rectangle(width: 20,  
height: 20, fill: blue)  
        }  
    }  
    tab(text: 'Tab 3') {  
        label(text: "This is Label
```

3\n\nAnd there were a few empty lines just there!")

```
        graphic() {  
            rectangle(width: 20,  
height: 20, fill: green)  
        }  
    }  
}
```

```
}  
  
});
```

## SceneGraphBuilder.tabPane.tab

### *Generated Object*

javafx.scene.control.Tab

### *Attributes*

This node takes the attributes of the `javafx.scene.control.Tab` see [javafx.scene.control.Tab](#).

### *Content*

If the content is a tooltip, the the Tab's tooltip will be set to the Tooltip.

If the content is a graphic, then this Tab's graphic is set to the graphic node.

If the content is a JavaFX node, this Tab's content is set to that node.

### *Usage*

Used to create a Tab that is added to a tabPane.

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "TabPane Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      tabPane ( ) {  
        tab(text: 'Tab 1') {
```

```
        label(text: "This is Label  
1\n\nAnd there were a few empty lines just  
there!")
```

```
        graphic() {  
            rectangle(width: 20,  
height: 20, fill: red)  
        }  
    }  
    tab(text: 'Tab 2') {  
        label(text: "This is Label  
2\n\nAnd there were a few empty lines just  
there!")
```

```
        graphic() {  
            rectangle(width: 20,  
height: 20, fill: blue)  
        }  
    }  
    tab(text: 'Tab 3') {  
        label(text: "This is Label  
3\n\nAnd there were a few empty lines just  
there!")
```

```
        graphic() {  
            rectangle(width: 20,  
height: 20, fill: green)  
        }  
    }  
}
```

```
}  
  
});
```

## **SceneGraphBuilder.textArea**

### *Generated Object*

javafx.scene.control.TextArea

### *Attributes*

This node takes the attributes of the `javafx.scene.control.TextArea` see [javafx.scene.control.TextArea](#).

### *Content*

If the content is a tooltip, the the TextArea's tooltip will be set to the Tooltip.

### *Usage*

Used to create a text area that supports multiple lines of text.

### *Examples*

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            text = textArea(prefRowCount:
10, prefColumnCount:80)
        }
    }
});

```

## SceneGraphBuilder.textBox

### *Generated Object*

javafx.scene.control.TextBox

### *Attributes*

This node takes the attributes of the javafx.scene.control.TextBox see [javafx.scene.control.TextBox](#).

### *Content*

If the content is a tooltip, the the TextBox's tooltip will be set to the Tooltip.

### *Usage*

Used to create a text box

### *Examples*

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            tf = textBox(prefWidth: 200,
action: { println tf.text} )
        }
    }
});

```

## SceneGraphBuilder.textField

### *Generated Object*

javafx.scene.control.TextField

### *Attributes*

This node takes the attributes of the javafx.scene.control.TextField see [javafx.scene.control.TextField](#).

### *Content*

If the content is a tooltip, the the TextField's tooltip will be set to the Tooltip.

### *Usage*

Used to create a text area that support a single line of text.

### *Examples*

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            text =
textField(prefColumnCount:80)
        }
    }
});

```

## SceneGraphBuilder.titledPane

### *Generated Object*

javafx.scene.control.TitledPane

### *Attributes*

This node takes the attributes of the `javafx.scene.control.TitledPane` see [javafx.scene.control.TitledPane](#).

### *Content*

If the content is a tooltip, the the TitledPane's tooltip will be set to the Tooltip.

The content may be a "title", "content" node or a JavaFX node. If the content is a JavaFX node, then the TitledPane's content attribute will be set to this node.

### *Usage*

Used to create a Titled Pane

### *Examples*

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "TitledPane Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated"
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        vbox ( spacing: 10) {
            titledPane(id: "t1", ) {
                title() {
                    label(text: "Label 1")
                }
                content() {
                    label(text: "This is
Label 1\n\nAnd there were a few empty lines
just there!")
                }
            }
            titledPane(id: "t2") {
                title() {
                    label(text: "Label 2")
                }
                content() {
                    label(text: "This is
Label 2\n\nAnd there were a few empty lines
just there!")
                }
            }
        }
    }
}

```

```
    }
    titledPane(id: "t3") {
        title() {
            label(text: "Label 3")
        }
        // this is content
        label(text: "This is Label
3\n\nAnd there were a few empty lines just
there!")
    }
}
```

```
}  
  
});
```

## SceneGraphBuilder.titledPane.content

### *Generated Object*

javafx.scene.Node

### *Attributes*

This node takes no attributes

### *Content*

The content is a JavaFX node, that is used for the TitledPane's content.

### *Usage*

Used to create the TitledPane's content node.

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "TitledPane Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated"  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      vbox ( spacing: 10) {  
        titledPane(id: "t1", ) {  
          title() {  
            label(text: "Label 1")  
          }  
        }  
      }  
    }  
  }  
})
```

```
        }
        content() {
            label(text: "This is
Label 1\n\nAnd there were a few empty lines
just there!")
        }
    }
    titledPane(id: "t2") {
        title() {
            label(text: "Label 2")
        }
        content() {
            label(text: "This is
Label 2\n\nAnd there were a few empty lines
just there!")
        }
    }
    titledPane(id: "t3") {
        title() {
            label(text: "Label 3")
        }
        // this is content
        label(text: "This is Label
3\n\nAnd there were a few empty lines just
there!")
    }
}
}
```

```
}  
  
});
```

## SceneGraphBuilder.titledPane.title

### *Generated Object*

javafx.scene.Node

### *Attributes*

This node takes no attributes

### *Content*

The content is a JavaFX node, that is used for the TitledPane's title.

### *Usage*

Used to create a TitledPane's title.

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "TitledPane Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated"  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      vbox ( spacing: 10) {  
        titledPane(id: "t1", ) {  
          title() {  
            label(text: "Label 1")
```

```

        }
        content() {
            label(text: "This is
Label 1\n\nAnd there were a few empty lines
just there!")
        }
    }
    titledPane(id: "t2") {
        title() {
            label(text: "Label 2")
        }
        content() {
            label(text: "This is
Label 2\n\nAnd there were a few empty lines
just there!")
        }
    }
    titledPane(id: "t3") {
        title() {
            label(text: "Label 3")
        }
        // this is content
        label(text: "This is Label
3\n\nAnd there were a few empty lines just
there!")
    }
}
}
}

```

```
}  
  
});
```

## SceneGraphBuilder.toggleButton

### *Generated Object*

javafx.scene.control.ToggleButton

### *Attributes*

This node takes the attributes of the javafx.scene.control.ToggleButton. see [javafx.scene.control.ToggleButton](#).

### *Content*

If the content is a javafx.scene.control.Tooltip (see [javafx.scene.control.Tooltip](#)) object, then the CheckBox's tooltip will be set to this Tooltip object.

If the content is a JavaFX Node, then the ToggleButton's graphic will be set to the JavaFX Node.

Other content is processed per the common javafx Node content including input events, effects and transforms.

### *Usage*

This is a JavaFX Node and accepts the common Node attributes.

### *Examples*

```
GroovyFX.start({  
    def sg = new SceneGraphBuilder();  
    sg.stage(  
        title: "Stage Frame",  
        x: 100, y: 100, width: 400,  
height:400,  
        visible: true,  
        style: "decorated",  
        onHide: { println "Close"}  
    ) {  
  
        scene(fill: hsb(128, 0.5, 0.5, 0.5),
```

```
parent: group()) {
    toggleButton (
        layoutX: 25,
        layoutY: 300,
        font: "16pt Courier",
        text: "One",
        selected: true,
        toggleGroup: "Group1"
    )
    toggleButton (
        layoutX: 125,
        layoutY: 300,
        font: "16pt Courier",
        text: "Two",
        selected: true,
toggleGroup: "Group1"
    )
}
```

```
}  
}  
});
```

## SceneGraphBuilder.toolBar

### *Generated Object*

javafx.scene.control.ToolBar

### *Attributes*

This node takes the attributes of the `javafx.scene.control.ToolBar` see [javafx.scene.control.ToolBar](#).

### *Content*

If the content is a tooltip, the the `ToolBar`'s tooltip will be set to the `Tooltip`.

If the content is a JavaFX node, then it will be added to the `ToolBar`'s items.

### *Usage*

Used to create a `ToolBar`

### *Examples*

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "ToolBar Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        toolBar ( orientation: "vertical") {
            button(text: "button 1")
            button(text: "button 2")
            button(text: "button 3")
            button(text: "button 4")
            button(text: "button 5")
            button(text: "button 6")
            button(text: "button 7")
            button(text: "button 8")
            button(text: "button 9")
            button(text: "button 10")

        }
    }
}

});

```

### ***Generated Object***

javafx.scene.control.Tooltip

### ***Attributes***

This node takes the attributes of the `javafx.scene.control.Tooltip`. see [javafx.scene.control.Tooltip](#).

### ***Content***

If the content is a JavaFX Node, then the Tooltip's graphic will be set to the JavaFX Node.

### ***Usage***

Used for showing additional information about a Control when the Control is hovered over by the mouse.

### ***Examples***

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Stage Frame",
        x: 100, y: 100, width: 400,
height:400,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5),
parent: group()) {
            button (
                layoutX: 25,
                layoutY: 300,
                text: "Press me"
            ) {
                tooltip(text: "Demonstrates
a tooltip on a Button control")
            }
        }
    }
});

```

## SceneGraphBuilder - effects

JavaFX effects that may be applied to all JavaFX nodes.

### SceneGraphBuilder.blend

*Generated Object*

javafx.scene.effect.Blend

**Attributes**

This node takes the attributes of the javafx.scene.effect.Blend class, see [javafx.scene.effect.Blend](#).

**Content**

A blend may have a **topInput** and **bottomInput** effect as content.

**Usage**

Creates a blend effect applied to a JavaFX Node.

**Examples**

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        rectangle(x: 20, y: 20,width: 100,
height: 100, fill: red) {
            blend(mode: "multiply") {
                topInput() {
                    colorInput(paint: blue,
x: 10, y: 20, width: 200, height: 200)
                }
                bottomInput() {
                    colorInput(paint: red,
x: 5, y: 15, width: 205, height: 205)
                }
            }
        }
    }
}
});

```

***Generated Object***

javafx.scene.effect.Effect

***Attributes***

this node does not take any attributes

***Content***

An effect.

***Usage***

Creates an effect applied to as the bottomInput to a Blend effect

***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        rectangle(x: 20, y: 20,width: 100,
height: 100, fill: red) {
            blend(mode: "multiply") {
                topInput() {
                    colorInput(paint: blue,
x: 10, y: 20, width: 200, height: 200)
                }
                bottomInput() {
                    colorInput(paint: red,
x: 5, y: 15, width: 205, height: 205)
                }
            }
        }
    }
}
});

```

***Generated Object***

javafx.scene.effect.Effect

***Attributes***

this node does not take any attributes

***Content***

An effect.

***Usage***

Creates an effect applied to as the topInput to a Blend effect

***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        rectangle(x: 20, y: 20,width: 100,
height: 100, fill: red) {
            blend(mode: "multiply") {
                topInput() {
                    colorInput(paint: blue,
x: 10, y: 20, width: 200, height: 200)
                }
                bottomInput() {
                    colorInput(paint: red,
x: 5, y: 15, width: 205, height: 205)
                }
            }
        }
    }
}
});

```

### *Generated Object*

javafx.scene.effect.Bloom

### *Attributes*

This node takes the attributes of the javafx.scene.effect.Bloom class, see [javafx.scene.effect.Bloom](#).

### *Content*

An optional Effect may be provides that is used as input to the Bloom effect.

### *Usage*

Creates a bloom effect applied to a JavaFX Node.

### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      rectangle(x: 20, y: 20,width: 100,  
height: 100, fill: red) {  
        bloom()  
      }  
    }  
  }  
});
```

SceneGraphBuilder.boxBlur

### ***Generated Object***

javafx.scene.effect.BoxBlur

### ***Attributes***

This node takes the attributes of the javafx.scene.effect.BoxBlur class, see [javafx.scene.effect.BoxBlur](#).

### ***Content***

An optional Effect may be provides that is used as input to the BoxBlur effect.

### ***Usage***

Creates a BoxBlur effect applied to a JavaFX Node.

### ***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(content: "Blurry Text!", fill:
"red", font: "bold 36pt", x: 10, y: 10,
textOrigin: "top") {
            boxBlur(width: 15, height: 15,
iterations: 3)
        }
    }
}

});

```

## SceneGraphBuilder.colorAdjust

### *Generated Object*

javafx.scene.effect.ColorAdjust

### *Attributes*

This node takes the attributes of the [javafx.scene.effect.ColorAdjust](#) class, see [javafx.scene.effect.ColorAdjust](#).

### *Content*

An optional Effect may be provides that is used as input to the ColorAdjust effect.

### Usage

Creates a ColorAdjust effect applied to a JavaFX Node.

### Examples

```
GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "ColorAdjust Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(content: "Blurry Text!", fill:
"red", font: "bold 36pt", x: 10, y: 10,
textOrigin: "top") {
            colorAdjust(contrast: 4,
brightness: -0.8, hue: 0.2)
        }
    }
}

});
```

### SceneGraphBuilder.colorInput

#### Generated Object

javafx.scene.effect.ColorInput

#### Attributes

This node takes the attributes of the [javafx.scene.effect.ColorInput](#) class, see [javafx.scene.effect.ColorInput](#).

### *Content*

There is no content for this node.

### *Usage*

Creates a ColorInput effect applied to a JavaFX Node.

### *Examples*

```
GroovyFX.start({  
def sg = new SceneGraphBuilder(it);  
  
sg.stage(  
    title: "ColorInput Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
        text(content: "Some Text!", fill:  
"red", font: "bold 36pt", x: 10, y: 10,  
textOrigin: "top") {  
            colorInput(paint: blue, x: 20,  
y: 10, width: 200, height: 200)  
        }  
    }  
}  
  
});
```

### **SceneGraphBuilder.displacementMap**

#### *Generated Object*

javafx.scene.effect.DisplacementMap

### Attributes

This node takes the attributes of the `javafx.scene.effect.DisplacementMap` class, see [javafx.scene.effect.DisplacementMap](#).

### Content

An optional Effect may be provided that is used as input to the DisplacementMap effect.

### Usage

Creates a DisplacementMap effect applied to a JavaFX Node.

### Examples

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  FloatMap mapData = new FloatMap();  
  mapData.setWidth(220);  
  mapData.setHeight(100);  
  
  for (int i = 0; i < 220; i++) {  
    double v = (Math.sin(i / 50.0 * Math.PI)  
- 0.5) / 40.0;  
    for (int j = 0; j < 100; j++) {  
      mapData.setSamples(i, j, 0.0f,  
(float) v);  
    }  
  }  
})  
  
sg.stage(  
  title: "DisplacementMap Effect Example",  
  x: 100, y: 100, width: 400, height:400,  
  visible: true,  
  style: "decorated",
```

```
) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
        text(content: "Blurry Text!", fill:  
"red", font: "bold 36pt", x: 10, y: 10,  
textOrigin: "top") {  
            displacementMap(mapData:  
mapData)  
        }  
    }  
}
```

```
}  
  
});
```

## **SceneGraphBuilder.dropShadow**

### *Generated Object*

javafx.scene.effect.DropShadow

### *Attributes*

This node takes the attributes of the `javafx.scene.effect.DropShadow` class, see [javafx.scene.effect.DropShadow](#).

### *Content*

An optional Effect may be provides that is used as input to the DropShadow effect.

### *Usage*

Creates a DropShadow effect applied to a JavaFX Node.

### *Examples*

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        rectangle(x: 20, y: 20,width: 100,
height: 100, fill: red) {
            dropShadow(offsetY: 3, color:
[0.4, 0.4, 0.4])
        }
    }
}

});

```

## SceneGraphBuilder.gaussianBlur

### *Generated Object*

javafx.scene.effect.GaussianBlur

### *Attributes*

This node takes the attributes of the javafx.scene.effect.GaussianBlur class, see [javafx.scene.effect.GaussianBlur](#).

### *Content*

An optional Effect may be provides that is used as input to the GaussianBlur effect.

### *Usage*

Creates a GaussianBlur effect applied to a JavaFX Node.

*Examples*

```
GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 10, y: 140, cache: true,
content: "Blurry Text", fill: "red", font:
"bold 36pt") {
            gaussianBlur()
        }
    }
}
});
```

**SceneGraphBuilder.glow**

*Generated Object*

javafx.scene.effect.Glow

*Attributes*

This node takes the attributes of the javafx.scene.effect.Glow class, see [javafx.scene.effect.Glow](#).

*Content*

An optional Effect may be provided that is used as input to the Glow effect.

### Usage

Creates a Glow effect applied to a JavaFX Node.

### Examples

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
      text(x: 10, y: 140, cache: true,  
content: "Blurry Text", fill: "red", font:  
"bold 36pt") {  
        glow(level: 0.75)  
      }  
    }  
  }  
});
```

### SceneGraphBuilder.imageInput

#### Generated Object

javafx.scene.effect.ImageInput

#### Attributes

This node takes the attributes of the `javafx.scene.effect.ImageInput` class, see [javafx.scene.effect.ImageInput](#).

#### Content

An image which is set to the ImageInput's source property.

*Usage*

Creates a ImageInput effect applied to a JavaFX Node.

*Examples*

```
GroovyFX.start({  
def sg = new SceneGraphBuilder(it);  
  
sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
)  
{  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
        rectangle(x: 10, y: 140, width: 100,  
height: 50 fill: red ) {  
            imageInput(x: 10, y: 140) {  
  
image{"file://PIA01320_hires.jpg", width:  
100, height: 50)  
            }  
        }  
    }  
}  
});
```

SceneGraphBuilder.innerShadow

*Generated Object*

javafx.scene.effect.InnerShadow

*Attributes*

This node takes the attributes of the javafx.scene.effect.InnerShadow class, see [javafx.scene.effect.InnerShadow](#).

*Content*

An optional Effect may be provides that is used as input to the InnerShadow effect.

*Usage*

Creates a InnerShadow effect applied to a JavaFX Node.

*Examples*

```
GroovyFX.start({  
def sg = new SceneGraphBuilder(it);  
  
sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
    ) {  
  
        scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
            text(x: 10, y: 140, cache: true,  
content: "Some Text", fill: "red", font:  
"bold 36pt") {  
                innerShadow(offsetX: 4, offsetY:  
4)  
            }  
        }  
    }  
});
```

## SceneGraphBuilder.lighting

### *Generated Object*

javafx.scene.effect.Lighting

### *Attributes*

This node takes the attributes of the javafx.scene.effect.Lighting class, see [javafx.scene.effect.Lighting](#).

### *Content*

A blend may have a **bumpInput**, **contentInput** effect and a Light (distant, point, or spot).

### *Usage*

Creates a blend effect applied to a JavaFX Node.

### *Examples*

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 500, height:300,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 20, y: 120, content: "Light
Effect", font: "bold 90pt", fill: red) {
            lighting(surfaceScale: 5.0) {
                distant(azimuth: -135)
                bumpInput() {
                    dropShadow()
                }
                contentInput() {
                    glow()
                }
            }
        }
    }
}

});

```

SceneGraphBuilder.lighting.bumpInput

*Generated Object*

javafx.scene.effect.Effect

***Attributes***

This node has no attributes

***Content***

An effect

***Usage***

Defines the bumpInput for a Lighting effect.

***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 500, height:300,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 20, y: 120, content: "Light
Effect", font: "bold 90pt", fill: red) {
            lighting(surfaceScale: 5.0) {
                distant(azimuth: -135)
                bumpInput() {
                    dropShadow()
                }
                contentInput() {
                    glow()
                }
            }
        }
    }
}
});

```

SceneGraphBuilder.lighting.contentInput

*Generated Object*

javafx.scene.effect.Effect

***Attributes***

This node has no attributes

***Content***

An effect

***Usage***

Defines the contentInput for a Lighting effect.

***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 500, height:300,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 20, y: 120, content: "Light
Effect", font: "bold 90pt", fill: red) {
            lighting(surfaceScale: 5.0) {
                distant(azimuth: -135)
                bumpInput() {
                    dropShadow()
                }
                contentInput() {
                    glow()
                }
            }
        }
    }
}

});

```

SceneGraphBuilder.lighting.distant

*Generated Object*

javafx.scene.effect.Light.Distant

***Attributes***

This node takes the attributes from the javafx.scene.effect.Light.Distant class, see [javafx.scene.effect.Light.Distant](#)

***Content***

There is no content for this node.

***Usage***

Defines a Distant light for the Lighting effect.

***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 500, height:300,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 20, y: 120, content: "Light
Effect", font: "bold 90pt", fill: red) {
            lighting(surfaceScale: 5.0) {
                distant(azimuth: -135)
                bumpInput() {
                    dropShadow()
                }
                contentInput() {
                    glow()
                }
            }
        }
    }
}
});

```

SceneGraphBuilder.lighting.point

*Generated Object*

javafx.scene.effect.Light.Point

**Attributes**

This node takes the attributes from the javafx.scene.effect.Light.Point class, see [javafx.scene.effect.Light.Point](#)

**Content**

There is no content for this node.

**Usage**

Defines a Point light for the Lighting effect.

**Examples**

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 500, height:300,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 10, y: 10, content: "Light
Effect", font: "bold 90pt", fill: red,
textOrigin: "top") {
            lighting(surfaceScale: 5.0) {
                point(x: -100, y: -100, z:
50 )
            }
        }
    }
}
});

```

### SceneGraphBuilder.lighting.spot

*Generated Object*

javafx.scene.effect.Light.Spot

*Attributes*

This node takes the attributes from the [javafx.scene.effect.Light.Spot](#) class, see [javafx.scene.effect.Light.Spot](#)

### *Content*

There is no content for this node.

### *Usage*

Defines a Spot light for the Lighting effect.

### *Examples*

```
GroovyFX.start({  
def sg = new SceneGraphBuilder(it);  
  
sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 500, height:300,  
    visible: true,  
    style: "decorated",  
) {  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
        text(x: 10, y: 10, content: "Light  
Effect", font: "bold 90pt", fill: red,  
textOrigin: "top") {  
            lighting(surfaceScale: 5.0) {  
                spot(x: 0, y: 100, z: 50,  
pointsAtX: 100, pointsAtY: 0, pointsAtZ: 0,  
specularExponent: 2)  
            }  
        }  
    }  
}  
});
```

## **SceneGraphBuilder.motionBlur**

### *Generated Object*

javafx.scene.effect.MotionBlur

### *Attributes*

This node takes the attributes of the javafx.scene.effect.MotionBlur class, see [javafx.scene.effect.MotionBlur](#).

### *Content*

An optional Effect may be provided that is used as input to the MotionBlur effect.

### *Usage*

Creates a MotionBlur effect applied to a JavaFX Node.

### *Examples*

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(content: "Blurry Text!", fill:
"red", font: "bold 36pt", x: 10, y: 10,
textOrigin: "top") {
            motionBlur(radius: 15, angle:
-30)
        }
    }
}

});

```

## SceneGraphBuilder.perspectiveTransform

### *Generated Object*

javafx.scene.effect.PerspectiveTransform

### *Attributes*

This node takes the attributes of the javafx.scene.effect.PerspectiveTransform class, see [javafx.scene.effect.PerspectiveTransform](#).

### *Content*

An optional Effect may be provides that is used as input to the PerspectiveTransform effect.

### ***Usage***

Creates a PerspectiveTransform effect applied to a JavaFX Node.

### ***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        group( cache: true, layoutX: 0,
layoutY: 0) {
            rectangle(x: 10, y: 10, width:
280, height: 80, fill: "blue")
            text(x: 20, y: 65, content:
"Perspective", fill: "yellow", font: "bold
36pt")
                perspectiveTransform(ulx: 10,
uly: 10, urx: 310, ury:40,
                                                lrx: 310,
lry: 60, llx: 10, lly: 60)
            }
        }
    }
});

```

javafx.scene.effect.Reflection

**Attributes**

This node takes the attributes of the javafx.scene.effect.Reflection class, see [javafx.scene.effect.Reflection](#).

**Content**

An optional Effect may be provides that is used as input to the Reflection effect.

**Usage**

Creates a Reflection effect applied to a JavaFX Node.

**Examples**

```
GroovyFX.start({  
def sg = new SceneGraphBuilder(it);  
  
sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
        text(x: 10, y: 140, cache: true,  
content: "Blurry Text", fill: "red", font:  
"bold 36pt") {  
            reflection(fraction: 0.7)  
        }  
    }  
}  
});
```

SceneGraphBuilder.sepiaTone

### ***Generated Object***

javafx.scene.effect.SepiaTone

### ***Attributes***

This node takes the attributes of the javafx.scene.effect.SepiaTone class, see [javafx.scene.effect.SepiaTone](#).

### ***Content***

An optional Effect may be provides that is used as input to the SepiaTone effect.

### ***Usage***

Creates a SepiaTone effect applied to a JavaFX Node.

### ***Examples***

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "Blend Effect Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
        text(x: 10, y: 140, cache: true,
content: "Blurry Text", fill: "red", font:
"bold 36pt") {
            sepiaTone(level: 0.7)
        }
    }
}
});

```

## SceneGraphBuilder.shadow

### *Generated Object*

javafx.scene.effect.Shadow

### *Attributes*

This node takes the attributes of the javafx.scene.effect.Shadow class, see [javafx.scene.effect.Shadow](#).

### *Content*

An optional Effect may be provides that is used as input to the Shadow effect.

### *Usage*

Creates a Shadow effect applied to a JavaFX Node.

*Examples*

```
GroovyFX.start({  
def sg = new SceneGraphBuilder(it);  
  
sg.stage(  
    title: "Blend Effect Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {  
        text(x: 10, y: 140, cache: true,  
content: "Blurry Text", fill: "red", font:  
"bold 36pt") {  
            shadow(color: blue)  
        }  
    }  
}  
});
```

**SceneGraphBuilder - input**

Common Mouse and Key Events for JavaFX Nodes

**empty**

**SceneGraphBuilder.onKeyPressed**

*Generated Object*

javafx.event.EventHandler

*Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<KeyEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.KeyEvent](#). The closure takes a `KeyEvent` object as its argument.

*Content*

None

*Usage*

Used to pass key pressed events to a javafx node.

*Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
           fill: rgb(0,0,255),
onKeyPressed: {println 'mouse clicked
event' })
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onKeyPressed(onEvent: { event-> println
'key pressed event ' + event.character})
}
```

### SceneGraphBuilder.onKeyReleased

*Generated Object*

`javafx.event.EventHandler`

*Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<KeyEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.KeyEvent](#). The closure takes a `KeyEvent` object as its argument.

*Content*

None

*Usage*

Used to pass key released events to a javafx node.

## Examples

```
circle ( centerX: 50, centerY: 50, radius:
25,
        fill: rgb(0,0,255),
onKeyReleased: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onKeyReleased(onEvent: { event-> println
'key pressed event ' + event.character})
}
```

## SceneGraphBuilder.onKeyTyped

### Generated Object

javafx.event.EventHandler

### Attributes

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<KeyEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.KeyEvent](#). The closure takes a `KeyEvent` object as its argument.

### Content

None

### Usage

Used to pass key typed events to a javafx node.

### Examples

```
circle ( centerX: 50, centerY: 50, radius:
25,
        fill: rgb(0,0,255),
onKeyTyped: {println 'mouse clicked event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onKeyTyped(onEvent: { event-> println
'key pressed event ' + event.character})
}
```

## SceneGraphBuilder.onMouseClicked

### *Generated Object*

javafx.event.EventHandler

### *Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

### *Content*

None

### *Usage*

Used to pass mouse clicked events to a javafx node.

### *Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
           fill: rgb(0,0,255),
onMousePressed: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMousePressed(onEvent: { println 'mouse
clicked event'})
}
```

```
rectangle(x:50,y:50,width:200,height:200) {
    onMouseClicked() {
        println 'mouse clicked event'
    }
}
```

**!Update on 05/05/2012**

**!Usage for latest groovyfx checkout:**

```
rectangle(x:50,y:50,width:200,height:200) {  
    onMouseClicked() {  
        println 'mouse clicked event'  
    }  
}
```

## SceneGraphBuilder.onMouseDragged

### *Generated Object*

javafx.event.EventHandler

### *Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

### *Content*

None

### *Usage*

Used to pass mouse dragged events to a javafx node.

### *Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
           fill: rgb(0,0,255),
onMouseDragged: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMouseDragged(onEvent: { println 'mouse
clicked event'})
}
```

## SceneGraphBuilder.onMouseEntered

### *Generated Object*

javafx.event.EventHandler

### *Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

### *Content*

None

### *Usage*

Used to pass mouse entered events to a javafx node.

### *Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
           fill: rgb(0,0,255),
onMouseEntered: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMouseEntered(onEvent: { println 'mouse
clicked event'})
}
```

## SceneGraphBuilder.onMouseExited

### *Generated Object*

javafx.event.EventHandler

### *Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

### *Content*

None

### *Usage*

Used to pass mouse exited events to a javafx node.

### *Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
        fill: rgb(0,0,255),
onMouseExited: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMouseExited(onEvent: { println 'mouse
clicked event'})
}
```

## SceneGraphBuilder.onMouseClicked

### *Generated Object*

javafx.event.EventHandler

### *Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

### *Content*

None

### *Usage*

Used to pass mouse moved events to a javafx node.

### *Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
        fill: rgb(0,0,255),
onMouseMoved: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMouseMoved(onEvent: { println 'mouse
clicked event'})
}
```

## SceneGraphBuilder.onMousePressed

### *Generated Object*

javafx.event.EventHandler

### *Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

### *Content*

None

### *Usage*

Used to pass mouse moved pressed to a javafx node.

### *Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
           fill: rgb(0,0,255),
onMousePressed: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMousePressed(onEvent: { println 'mouse
clicked event'})
}
```

## SceneGraphBuilder.onMouseReleased

*Generated Object*

javafx.event.EventHandler

*Attributes*

**onEvent** either a Groovy closure or a `javafx.event.EventHandler<MouseEvent>`, see [javafx.event.EventHandler](#) and [javafx.scene.input.MouseEvent](#). The closure takes a `MouseEvent` object as its argument.

*Content*

None

*Usage*

Used to pass mouse released events to a javafx node.

*Examples*

```
circle ( centerX: 50, centerY: 50, radius:
25,
           fill: rgb(0,0,255),
onMouseReleased: {println 'mouse clicked
event'})
```

```
circle (centerX: 100, centerY: 50, radius:
25, fill: red) {
    onMouseReleased(onEvent: { println
'mouse clicked event'})
}
```

## SceneGraphBuilder - nodes

Custom or specialized SceneGraphBuilder nodes that generate javafx.scene.Node objects.

All JavaFX nodes may take content that includes transforms ([SceneGraphBuilder - transforms](#)), effects([SceneGraphBuilder - effects](#)), and mouse and key events ([SceneGraphBuilder - input](#))

### SceneGraph.text

#### *Generated Object*

javafx.scene.text.Text

#### *Attributes*

This node takes the attributes of the javafx.scene.text.Text class. see [javafx.scene.text.Text](#)

#### *Content*

This node may take general javafx node content like effects.

#### *Usage*

Creates a Text shape.

#### *Examples*

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Text Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            text(x: 10, y: 10,
                text: "This is text.",
                fill: red, font: "bold
20pt",
                textOrigin: "top"
            )
        }
    }
});

```

### SceneGraphBuilder.imageView

#### *Generated Object*

javafx.scene.image.ImageView

#### *Attributes*

This node takes the attributes of the [javafx.scene.image.ImageView](#) class. see [javafx.scene.image.ImageView](#)

#### *Content*

This node takes a image node as content.

### Usage

Creates an ImageView to display an image.

### Examples

```
import groovyx.javafx.SceneGraphBuilder
import groovyx.javafx.GroovyFX

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Hello Image (Groovy)",
        visible: true,
    ) {
        scene(fill: "lightgray", width: 600,
height:500,) {
            imageView(x: 25, y: 40) {

                image("file:PIA01320_hires.jpg", width:
500, height: 450, preserveRatio: true);
            }
        }
    }
});
```

### SceneGraphBuilder.imageView.image

#### Generated Object

javafx.scene.image.Image

#### Value

If a value is passed it is used as the url location for the image.

#### Attributes

This node takes the attributes of the [javafx.scene.image.Image](#) class. see [javafx.scene.image.Image](#).

In addition

**url**: the URL location for the image.

*Content*

This node does not take any content

*Usage*

Creates an Image.

*Examples*

```
import groovyx.javafx.SceneGraphBuilder
import groovyx.javafx.GroovyFX

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Hello Image (Groovy)",
        visible: true,
    ) {
        scene(fill: "lightgray", width: 600,
height:500,) {
            imageView(x: 25, y: 40) {

                image("file:PIA01320_hires.jpg", width:
500, height: 450, preserveRatio: true);
            }
        }
    }
});
```

SceneGraph.node

*Generated Object*

javafx.scene.Node

### *Value*

The custom node object

### *Attributes*

This node takes the attributes of the `javafx.scene.Node` class. see [javafx.scene.Node](#)

In addition, if a value is not present, then the following attribute is required:

**node**: the custom node object

### *Content*

This node may take general javafx node content like effects, transforms and inputs

### *Usage*

Creates a Custom Node.

### *Examples*

```
import groovyx.javafx.SceneGraphBuilder
import groovyx.javafx.GroovyFX
import javafx.scene.shape.Rectangle;
import javafx.scene.paint.Color;
import javafx.scene.layout.Region;

class MyCustomNode extends Region {

    public MyCustomNode() {
        getChildren().add(create());
    }

    protected javafx.scene.Node create() {
        return new Rectangle(width: 10,
height: 10, fill: Color.BLUE);
    }
}
```

```
GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Custom Node example",
        x: 100, y: 100, width: 200,
height:200,
        visible: true,
        style: "decorated",
        onHide: { println "Close"}
    ) {

        scene(fill: hsb(128, 0.5, 0.5, 0.5))
    {
        node(new MyCustomNode(),
layoutX: 10, layoutY: 10) {
            scale(x: 5, y: 5)
            onChange(property: "hover",
changed: {observable, oldValue, newValue ->
println "hover: " + oldValue + " ==> " +
newValue})
        }
    }
}
```

```
    }  
  }  
});
```

## SceneGraph.container

### *Generated Object*

javafx.scene.Parent

### *Value*

The custom container object

### *Attributes*

This node takes the attributes of the javafx.scene.Parent class. see [javafx.scene.Parent](#)

In addition, if a value is not present, then the following attribute is required:

**container:** the custom parent object

### *Content*

This node may take general javafx node content like effects, transforms and inputs. Also it make take JavaFX nodes that are added to the custom container.

### *Usage*

Creates a Custom Container.

### *Examples*

```
import groovyx.javafx.SceneGraphBuilder  
import groovyx.javafx.GroovyFX  
import javafx.scene.shape.Rectangle;  
import javafx.scene.paint.Color;  
import javafx.scene.layout.Region;  
import javafx.geometry.HPos;  
import javafx.geometry.VPos;  
  
class SimpleContainer extends Region {
```

```

public SimpleContainer() {
}

@Override
protected void layoutChildren() {
    List<Node> managed =
getManagedChildren();
    def width = getWidth();
    def height = getHeight();
    def partWidth =
width/managed.size();

    def x = 0
    for(n in managed) {
        layoutInArea(n, x, 0, partWidth,
height, 0,
                HPos.CENTER, VPos.CENTER);
        x += partWidth;
    }
}
}

```

```

GroovyFX.start({
    def sg = new SceneGraphBuilder();
    sg.stage(
        title: "Custom Container example",
        x: 100, y: 100, width: 200,
height:200,
        visible: true,
    ) {
        scene(fill: hsb(128, 0.5, 0.5, 0.5))
    }
}

```

```
{  
    container(new SimpleContainer())  
    {  
        button(text: "one")  
        button(text: "two")  
        button(text: "three")  
    }  
}
```

```

    }
}
);

```

**SceneGraphBuilder - paint**

Provides support for creating colors and paints.

Colors may be set by using one of the built-in color names or using one of the built-in list formats.

Lists can be:

**rgb format** - ["rgb", r, g, b, opacity] where the first item is "rgb" to indicate an rgb format. The second item in the list represents the red value in a range of 0 to 255, the third and fourth itesm are green and blue respectively. The last item, if present, is the opacity in a range from 0 to 1.

**hsb format** - ["hsb", h, s, b, opactiy] where the first item is "hsb" to indicate hsb format. The second item in the list represents the red value in a range of 0 to 255, the third and fourth itesm are green and blue respectively. The last item, if present, is the opacity in a range from 0 to 1.

**javafx.scene.paint.Color** format: [ r, g, b, opacity] where the first item is the red value in a range from 0 to 1, the third and fourth itesm are green and blue respectively. The last item, if present, is the opacity in a range from 0 to 1.

Colors may also be represented as a web format contained in a string, as in "#FFFFFF".

There is also support for the JavaFX CSS style color formats such as **textFill: "rgb(100%,100%,0)"** and\* fill: "radial (20%,20%), 100% focus(20%,20%) stops (0.0,red) (0.50,darkred) (1.0,black)".\* see [JavaFX CSS Reference Guide](#)

Built-in colors include:

aliceblue	darkseagreen	lightgray	palegoldenrod	whitesmoke
antiquewhite	darkslateblue	lightgreen	palegreen	yellow
aqua	darkslategray	lightgrey	paleturquoise	yellowgreen
aquamarine	darkslategrey	lightpink	palevioletred	
antiquewhite	darkturquoise	lightsalmon	papayawhip	
aqua	darkviolet	lightseagreen	peachpuff	
aquamarine	deeppink	lightskyblue	peru	
azure	deepskyblue	lightslategray	pink	
beige	dimgray	lightslategrey	plum	

bisque	dimgrey	lightsteelblue	powderblue
black	dodgerblue	lightyellow	purple
blanchedalmond	firebrick	lime	red
blue	floralwhite	limegreen	rosybrown
blueviolet	forestgreen	linen	royalblue
brown	fuchsia	magenta	saddlebrown
burlywood	gainsboro	maroon	salmon
cadetblue	ghostwhite	mediumaquamarine	sandybrown
chartreuse	gold	mediumblue	seagreen
chocolate	goldenrod	mediumorchid	seashell
coral	gray	mediumpurple	sienna
cornflowerblue	green	mediumseagreen	silver
cornsilk	greenyellow	mediumslateblue	skyblue
crimson	grey	mediumspringgreen	slateblue
cyan	honeydew	mediumturquoise	slategray
darkblue	hotpink	mediumvioletred	slategrey
darkcyan	indianred	midnightblue	snow
darkgoldenrod	indigo	mintcream	springgreen
darkgray	ivory	mistyrose	steelblue
darkgreen	khaki	moccasin	tan
darkgrey	lavender	navajowhite	teal
darkkhaki	lavenderblush	navy	thistle
darkmagenta	lawngreen	oldlace	tomato
darkolivegreen	lemonchiffon	olive	transparent

darkorange	lightblue	olivedrab	turquoise
darkorchid	lightcoral	orange	violet
darkred	lightcyan	orangered	wheat
darksalmon	lightgoldenrodyellow	orchid	white

## Scenegraph.hsb

### *Generated Object*

javafx.scene.paint.Color

### *Value Argument*

**hue** - the hue in degrees component of an hsb color using an integer in the range of 0 to 360

**saturation** - the saturation component of an hsb color in the range of 0.0 to 1.0

brightness - the brightness component of an hsb color in the range of 0.0 to 1.0

**opacity** - the optional opacity of a color in the range 0.0 to 1.0. If opacity is not present then the opacity will be 1.0.

### *Attributes*

This node takes no attributes

### *Content*

This node takes no content.

### *Usage*

This node creates a color based on rgb values. see [javafx.scene.paint.Color#hsb](#)

### *Examples*

```

GroovyFX.start({
def sg = new SceneGraphBuilder(it);

sg.stage(
    title: "HSB Example",
    x: 100, y: 100, width: 400, height:400,
    visible: true,
    style: "decorated",
    primary: true
) {

    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {

    }

}
});

```

## Scenegraph.linearGradient

### *Generated Object*

javafx.scene.paint.LinearGradient

### *Attributes*

This node takes the attributes of the `javafx.scene.paint.LinearGradient` class. see [javafx.scene.paint.LinearGradient](#)

In addition, it takes the following attributes:

**start** - a list of 2 elements with the first value used to set the startX attribute of the LinearGradient and the second value to set the startY attribute.

**end** - a list of 2 elements with the first value used to set the endX attribute of the LinearGradient and the second value to set the endY attribute.

**stops** - The value of the stops attribute should be a List of 2-element Lists containing an offset and color: `[[0.0, black], [1.0, red]]`

### *Content*

This node may optionally take one or more stop nodes as content. see [Scenegraph.stop](#)

#### *Usage*

This node creates a `javafx.scene.paint.LinearGradient`.

#### *Examples*

```

import groovyx.javafx.GroovyFX;
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(title: "Gradient Example
(GroovyFX)", width: 1020, height: 450,
visible: true) {
        scene(fill: linearGradient(endX: 0,
stops: [[0.0, white], [1.0, lightgray]])) {

            def width = 240
            def height = 180

            rectangle(width: width, height:
height,
                    fill:
linearGradient(stops:[[0.0, red],[1.0,
black]]),
                    stroke:
linearGradient(stops: [[0.0, blue],
[1.0,red]]),
                    strokeWidth: 10 )

        }
    }
});

```

### *Generated Object*

javafx.scene.paint.RadialGradient

### *Attributes*

This node takes the attributes of the `javafx.scene.paint.RadialGradient` class. see [javafx.scene.paint.RadialGradient](#)

In addition, it takes the following attributes:

**center** - a list of 2 elements with the first value used to set the centerX attribute of the LinearGradient and the second value to set the centerY attribute.

**stops** - The value of the stops attribute should be a List of 2-element Lists containing an offset and color: `[[0.0, black], [1.0, red]]`

### *Content*

This node may optionally take one or more stop nodes as content. see [Scenegraph.stop](#)

### *Usage*

This node creates a radial gradient paint.

### *Examples*

```

import groovyx.javafx.GroovyFX;
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(title: "Gradient Example
(GroovyFX)", width: 1020, height: 450,
visible: true) {
        scene(fill: linearGradient(endX: 0,
stops: [[0.0, white], [1.0, lightgray]])) {

            def width = 240
            def height = 180

            circle(radius: height / 2) {
                radialGradient(radius:
0.85, center:[0.5, 1], stops:[[0, magenta],
[0.75, indigo]])
            }

        }
    }
});

```

Scenegraph.rgb

*Generated Object*

javafx.scene.paint.Color

*Value Argument*

red - the red component of an rgb color using an integer in the range of 0 to 255

green - the green component of an rgb color using an integer in the range of 0 to 255

blue - the blue component of an rgb color using an integer in the range of 0 to 255

opacity - the optional opacity of a color in the range 0.0 to 1.0. If opacity is not present then the opacity will be 1.0.

***Attributes***

This node takes no attributes

***Content***

This node takes no content.

***Usage***

This node creates a color based on rgb values. see [javafx.scene.paint.Color#rgb](#)

***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Arc Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: rgb(33,33,33) ) {
            arc(centerX: 100, centerY: 100,
                radiusX: 50, radiusY: 50,
                startAngle: -45, length:
180,
                fill: red, type: "chord")
        }
    }
});

```

## Scenegraph.stop

### *Generated Object*

javafx.scene.paint.Stop used in conjunction with Linear and Radial Gradients

### *Attributes*

This node takes the attributes of the javafx.scene.paint.Stop class. see [javafx.scene.paint.Stop](#)

Specifically:

**offset** - number ranging from 0 to 1 to indicate where the gradient stop should be placed

**color** - The color of the gradient at this offset. The default is black.

***Content***

This node does not take content

***Usage***

This node creates a gradient stop.

***Examples***

```

import groovyx.javafx.GroovyFX;
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(title: "Gradient Example
(GroovyFX)", width: 1020, height: 450,
visible: true) {
        scene(fill: linearGradient(endX: 0,
stops: [[0.0, white], [1.0, lightgray]])) {

            def width = 240
            def height = 180

            circle(radius: height / 2) {
                radialGradient(radius:
0.85, center:[0.5, 1]) {
                    stop(offset: 0, color:
magenta)
                    stop(offset: 0.75,
color: indigo)
                }
            }
        }
    }
});

```

JavaFX shapes

## **SceneGraphBuilder.arc**

### *Generated Object*

javafx.scene.shape.Arc

### *Attributes*

This node takes the attributes from the javafx.scene.shape.Arc class. see [javafx.scene.shape.Arc](#).

### *Content*

This node does not take any content.

### *Usage*

Create an Arc shape

### *Examples*

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Arc Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            arc(centerX: 100, centerY: 100,
                radiusX: 50, radiusY: 50,
                startAngle: -45, length:
180,
                fill: red, type: "chord")
        }
    }
});

```

## SceneGraphBuilder.circle

### *Generated Object*

javafx.scene.shape.Circle

### *Attributes*

This node takes the attributes from the javafx.scene.shape.Circle class. see [javafx.scene.shape.Circle](#).

### *Content*

This node does not take any content.

### *Usage*

Create an Circle shape

*Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Circle Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            circle(centerX: 100, centerY:
100,
                radius: 50, fill: red)
        }
    }
});
```

**SceneGraphBuilder.cubicCurve**

*Generated Object*

javafx.scene.shape.CubicCurve

*Attributes*

This node takes the attributes from the `javafx.scene.shape.CubicCurve` class. see [javafx.scene.shape.CubicCurve](#).

*Content*

This node does not take any content.

*Usage*

Create an CubicCurve shape

*Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "CubicCurve Example
(Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            cubicCurve(startX: 10, startY:
10,
                        endX: 100, endY: 100,
                        controlX1: 40, controlY1: 0,
                        controlX2: 50, controlY2:
110,
                        stroke: red, strokeWidth: 2,
                        fill: null)
        }
    }
});
```

SceneGraphBuilder.ellipse

*Generated Object*

javafx.scene.shape.Ellipse

*Attributes*

This node takes the attributes from the `javafx.scene.shape.Ellipse` class. see [javafx.scene.shape.Ellipse](#).

#### *Content*

This node does not take any content.

#### *Usage*

Create an Ellipse shape

#### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Ellipse Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            ellipse(centerX: 100, centerY:
100,
                    radiusX: 50, radiusY: 25,
fill: red)
        }
    }
});
```

SceneGraphBuilder.line

#### *Generated Object*

`javafx.scene.shape.Line`

#### *Attributes*

This node takes the attributes from the `javafx.scene.shape.Line` class. see [javafx.scene.shape.Line](#).

#### *Content*

This node does not take any content.

#### *Usage*

Create a Line shape

#### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Line Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ) {
        scene (fill: white) {
            line(startX: 10, startY: 10,
                endX: 150, endY: 150,
                stroke: red, strokeWidth: 2)
        }
    }
});
```

#### **SceneGraphBuilder.path**

##### *Generated Object*

`javafx.scene.shape.Path`

##### *Attributes*

This node takes the attributes from the `javafx.scene.shape.Path` class. see [javafx.scene.shape.Path](#).

***Content***

This node takes path elements as content, `arcTo`, `cubicCurveTo`, `hLineTo`, `lineTo`, `moveTo`, `quadCurveTo`, `vLineTo` and `closePath`

***Usage***

Create a shape based on a Path.

***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, fill: red) {
                moveTo(x: 0, y: 0)
               .lineTo(x: 100, y: 0)
                arcTo(x: 200, y: 0, radiusX:
25, radiusY: 20)
               .lineTo(x: 300, y: 0)
               .lineTo(x: 150, y: 100)
                closePath()
            }
        }
    }
});

```

SceneGraphBuilder.path.arcTo

*Generated Object*

javafx.scene.shape.ArcTo

*Attributes*

This node takes the attributes from the `javafx.scene.shape.ArcTo` class. see [javafx.scene.shape.ArcTo](#).

***Content***

This node does not have any content

***Usage***

Create an `ArcTo` path element.

***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, fill: red) {
                moveTo(x: 0, y: 0)
               .lineTo(x: 100, y: 0)
                arcTo(x: 200, y: 0, radiusX:
25, radiusY: 20)
               .lineTo(x: 300, y: 0)
               .lineTo(x: 150, y: 100)
                closePath()
            }
        }
    }
});

```

SceneGraphBuilder.path.closePath

*Generated Object*

javafx.scene.shape.ClosePath

*Attributes*

This node takes the attributes from the `javafx.scene.shape.ClosePath` class. see [javafx.scene.shape.ClosePath](#).

#### ***Content***

This node does not have any content

#### ***Usage***

Create an `ClosePath` path element.

#### ***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, fill: red) {
                moveTo(x: 0, y: 0)
               .lineTo(x: 100, y: 0)
                arcTo(x: 200, y: 0, radiusX:
25, radiusY: 20)
               .lineTo(x: 300, y: 0)
               .lineTo(x: 150, y: 100)
                closePath()
            }
        }
    }
});

```

SceneGraphBuilder.path.cubicCurveTo

*Generated Object*

javafx.scene.shape.CubicCurveTo

*Attributes*

This node takes the attributes from the `javafx.scene.shape.CubicCurveTo` class. see [javafx.scene.shape.CubicCurveTo](#).

#### ***Content***

This node does not have any content

#### ***Usage***

Create an `CubicCurveTo` path element.

#### ***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, stroke: red) {
                moveTo(x: 0, y: 0)
                cubicCurveTo(controlX1: -10,
controlY1: 20,
                        controlX2: 80,
controlY2: 100,
                        x: 100, y: 50)
            }
        }
    }
});

```

SceneGraphBuilder.path.hLineTo

*Generated Object*

javafx.scene.shape.HLineTo

*Attributes*

This node takes the attributes from the `javafx.scene.shape.HLineTo` class. see [javafx.scene.shape.HLineTo](#).

#### *Content*

This node does not have any content

#### *Usage*

Create an `HLineTo` path element.

#### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, stroke: red) {
                moveTo(x: 25, y: 50)
                hLineTo(x: 250)
                vLineTo(y: 150)
            }
        }
    }
});
```

`SceneGraphBuilder.path.lineTo`

*Generated Object*

javafx.scene.shape.LineTo

***Attributes***

This node takes the attributes from the javafx.scene.shape.LineTo class. see [javafx.scene.shape.LineTo](#).

***Content***

This node does not have any content

***Usage***

Create an LineTo path element.

***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, fill: red) {
                moveTo(x: 0, y: 0)
               .lineTo(x: 100, y: 0)
                arcTo(x: 200, y: 0, radiusX:
25, radiusY: 20)
               .lineTo(x: 300, y: 0)
               .lineTo(x: 150, y: 100)
                closePath()
            }
        }
    }
});

```

SceneGraphBuilder.path.moveTo

*Generated Object*

javafx.scene.shape.MoveTo

*Attributes*

This node takes the attributes from the `javafx.scene.shape.MoveTo` class. see [javafx.scene.shape.MoveTo](#).

#### ***Content***

This node does not have any content

#### ***Usage***

Create an `MoveTo` path element.

#### ***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, fill: red) {
                moveTo(x: 0, y: 0)
               .lineTo(x: 100, y: 0)
                arcTo(x: 200, y: 0, radiusX:
25, radiusY: 20)
               .lineTo(x: 300, y: 0)
               .lineTo(x: 150, y: 100)
                closePath()
            }
        }
    }
});

```

SceneGraphBuilder.path.quadCurveTo

*Generated Object*

javafx.scene.shape.QuadCurveTo

*Attributes*

This node takes the attributes from the `javafx.scene.shape.QuadCurveTo` class. see [javafx.scene.shape.QuadCurveTo](#).

#### *Content*

This node does not have any content

#### *Usage*

Create an `QuadCurveTo` path element.

#### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder()

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, stroke: red) {
                moveTo(x: 0, y: 0)
                quadCurveTo(controlX: -100,
controlY: 0, x: 100, y: 50)
            }
        }
    }
});
```

`SceneGraphBuilder.path.vLineTo`

### ***Generated Object***

javafx.scene.shape.VLineTo

### ***Attributes***

This node takes the attributes from the javafx.scene.shape.VLineTo class. see [javafx.scene.shape.VLineTo](#).

### ***Content***

This node does not have any content

### ***Usage***

Create an VLineTo path element.

### ***Examples***

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Path Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            path( translateX: 50,
translateY: 50, stroke: red) {
                moveTo(x: 25, y: 50)
                hLineTo(x: 250)
                vLineTo(y: 150)
            }
        }
    }
});

```

### SceneGraphBuilder.polygon

#### *Generated Object*

javafx.scene.shape.Polygon

#### *Attributes*

This node takes the attributes from the [javafx.scene.shape.Polygon](#) class. see [javafx.scene.shape.Polygon](#).

#### *Content*

This node does not take any content.

## Usage

Create a Polygon shape

## Examples

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Polygon Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            polygon(
                points: [ 0, -10, 10, 10,
-10, 10, 0, -10],
                translateX: 100, translateY:
100,
                fill: red)
        }
    }
});
```

## SceneGraphBuilder.polyline

### Generated Object

javafx.scene.shape.Polyline

### Attributes

This node takes the attributes from the `javafx.scene.shape.Polyline` class. see [javafx.scene.shape.Polyline](#).

### *Content*

This node does not take any content.

### *Usage*

Create a Polyline shape

### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Polyline Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ) {
        scene (fill: white) {
            polyline(
                points: [ 0, -50, 50, 50,
-50, 50, 0, 0],
                translateX: 100, translateY:
100,
                stroke: red, strokeWidth: 2)
            }
        }
    });
```

**SceneGraphBuilder.quadCurve**

### *Generated Object*

javafx.scene.shape.QuadCurve

### Attributes

This node takes the attributes from the `javafx.scene.shape.QuadCurve` class. see [javafx.scene.shape.QuadCurve](#).

### Content

This node does not take any content.

### Usage

Create an `QuadCurve` shape

### Examples

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder()

    sg.stage(
        title: "QuadCurve Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            quadCurve(startX: 10, startY:
10,
                    endX: 100, endY: 100,
                    controlX: 75, controlY: 0,
                    stroke: red, strokeWidth: 2,
fill: null)
        }
    }
});
```

`SceneGraphBuilder.rectangle`

### *Generated Object*

javafx.scene.shape.Rectangle

### *Attributes*

This node takes the attributes from the javafx.scene.shape.Rectangle class. see [javafx.scene.shape.Rectangle](#).

### *Content*

This node does not take any content.

### *Usage*

Create a Rectangle shape

### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Rectangle Example (Groovy)",
        width: 200, height:200,
        visible: true,
    ){
        scene (fill: white) {
            rectangle( x: 25, y: 25, width:
150, height: 75,
                arcWidth: 20, arcHeight: 20,
                fill: blue, stroke: red,
strokeWidth: 2)
            }
        }
    });
```

## **SceneGraphBuilder.SVGPath**

### *Generated Object*

javafx.scene.shape.SVGPath

### *Attributes*

This node takes the attributes from the `javafx.scene.shape.SVGPath` class. see [javafx.scene.shape.SVGPath](#).

### *Content*

This node does not take any content.

### *Usage*

Create a shape based on a SVG Path.

### *Examples*

```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "SVGPath Example (Groovy)",
        width: 400, height:250,
        visible: true,
    ){
        scene (fill: white) {
            SVGPath(
                content: "M0,0 L100,0 A25,20
0 0,0 200,0 L300,0 150,100z",
                translateX: 50, translateY:
50,
                fill: transparent,
                stroke: red, strokeWidth: 2)
        }
    }
});

```

### SceneGraphBuilder - stage

Defines various types of JavaFX windows

### SceneGraphBuilder.popup

*Generated Object*

javafx.stage.Popup

*Attributes*

This node accepts the attributes for a `javafx.stage.Popup` class. See the JavaDoc for [javafx.stage.Popup](#)

#### *Content*

A JavaFX Node.

#### *Usage*

Creates a JavaFX popup window.

#### *Examples*

```
GroovyFX.start({  
  def sg = new SceneGraphBuilder(it);  
  
  def popup = null;  
  popup = sg.popup(autoHide: true) {  
    stackPane() {  
      rectangle(width: 200, height: 200,  
fill: blue,  
      stroke: cyan, strokeWidth: 5,  
arcHeight: 20, arcWidth: 20)  
      button( text: "OK", onAction:  
{popup.hide()})  
    }  
  }  
  
  def stage = sg.stage(  
    title: "Popup Example",  
    x: 100, y: 100, width: 400, height:400,  
    visible: true,  
    style: "decorated",  
  ) {  
  
    scene(fill: hsb(128, 0.5, 0.5, 0.5) ) {
```

```
stackPane() {  
    button(text: "Show Popup",  
onAction: { popup.show(stage) })  
    }  
}
```

```
}  
});
```

## SceneGraphBuilder.stage

### *Generated Object*

javafx.stage.Stage

### *Value Argument*

[Describe the value argument accepted by the node. If no value argument is accepted either state it or remove this section]

### *Attributes*

This node accepts the attributes for a javafx.stage.Stage class. See the JavaDoc for [javafx.stage.Stage](#)

In addition:

**primary** - boolean value to indicate whether or not to create a new stage (**false**) or to use the primary stage (**true**) created for JavaFX.

### *Content*

Content contains a Scene

### *Usage*

Creates a JavaFX window.

### *Examples*

```
import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

GroovyFX.start({
    def sg = new SceneGraphBuilder();

    sg.stage(
        title: "Stage Example (Groovy)",
        width: 650, height:450,
        visible: true,
    )
});
```

## SceneGraphBuilder - transforms

JavaFX transforms that may be applied to all JavaFX nodes.

### SceneGraphBuilder.affine

#### *Generated Object*

javafx.scene.transform.Affine

#### *Attributes*

This node takes the attributes of the `javafx.scene.transform.Affine` class, see [javafx.scene.transform.Affine](#)

#### *Content*

This node has no content

#### *Usage*

Defines an affine transform that may be applied to a JavaFX node.

#### *Examples*

```

GroovyFX.start({ primaryStage ->
    def sg = new
SceneGraphBuilder(primaryStage);

    sg.stage(
        title: "CircleT3D (Groovy)",
        width: 400, height:300,
        visible: true,
        resizable: true
    ) {
        scene(fill: lightgreen, camera: new
PerspectiveCamera()) {
            rectangle(x: 20, y: 20, width:
100, height: 50, fill: blue) {
                affine(tx: -20, mxx: 2, mxy:
0.5, mxz: 1,
                    ty: -50, myx: 1, myy:
1.2, myz: 1,
                    tz: 0, mzx: 1, mzy: 1,
mzz: 0)
            }
        }
    }
})

```

SceneGraphBuilder.rotate

*Generated Object*

javafx.scene.transform.Rotate

*Attributes*

This node takes the attributes of the `javafx.scene.transform.Rotate` class, see [javafx.scene.transform.Rotate](#)

#### *Content*

This node has no content

#### *Usage*

Defines a rotate transform that may be applied to a JavaFX node.

#### *Examples*

```
GroovyFX.start({ primaryStage ->
    def sg = new
    SceneGraphBuilder(primaryStage);
    if
    (!Platform.isSupported(ConditionalFeature.SC
    ENE3D)) {

        System.out.println("*****
        *****");
        System.out.println("*
        WARNING: common conditional SCENE3D isn\'t
        supported    *");

        System.out.println("*****
        *****");
    }
    sg.stage(
        title: "CircleT3D (Groovy)",
        width: 400, height:300,
        visible: true,
        resizable: true
    ) {
        scene(fill: lightgreen, camera: new
        PerspectiveCamera()) {
```

```
        circle(centerX: 200, centerY:
150, radius: 75, fill: blue) {
            rotate(angle: 30, axis:
Rotate.Y_AXIS)
        }
```

```
}  
}  
})
```

**SceneGraphBuilder.scale**

*Generated Object*

javafx.scene.transform.Scale

*Attributes*

This node takes the attributes of the javafx.scene.transform.Scale class, see [javafx.scene.transform.Scale](#)

*Content*

This node has no content

*Usage*

Defines a scale transform that may be applied to a JavaFX node.

*Examples*

```

GroovyFX.start({ primaryStage ->
    def sg = new
    SceneGraphBuilder(primaryStage);

    sg.stage(
        title: "CircleT3D (Groovy)",
        width: 400, height:300,
        visible: true,
        resizable: true
    ) {
        scene(fill: lightgreen, camera: new
    PerspectiveCamera()) {
        circle(centerX: 200, centerY:
    150, radius: 75, fill: blue) {
            scale(x: 1.2)
        }
    }
}
})

```

## SceneGraphBuilder.shear

### *Generated Object*

javafx.scene.transform.Shear

### *Attributes*

This node takes the attributes of the javafx.scene.transform.Shear class, see [javafx.scene.transform.Shear](#)

### *Content*

This node has no content

### *Usage*

Defines a shear transform that may be applied to a JavaFX node.

## Examples

```
GroovyFX.start({ primaryStage ->
    def sg = new
    SceneGraphBuilder(primaryStage);

    sg.stage(
        title: "CircleT3D (Groovy)",
        width: 400, height:300,
        visible: true,
        resizable: true
    ) {
        scene(fill: lightgreen, camera: new
    PerspectiveCamera()) {
        circle(centerX: 200, centerY:
    150, radius: 75, fill: blue) {
            shear(x: -0.35, y: 0)
        }
    }
}
})
```

### SceneGraphBuilder.translate

#### Generated Object

javafx.scene.transform.Translate

#### Attributes

This node takes the attributes of the javafx.scene.transform.Translate class, see [javafx.scene.transform.Translate](#)

#### Content

This node has no content

#### Usage

Defines a translate transform that may be applied to a JavaFX node.

```
GroovyFX.start({ primaryStage ->
    def sg = new
    SceneGraphBuilder(primaryStage);

    sg.stage(
        title: "CircleT3D (Groovy)",
        width: 400, height:300,
        visible: true,
        resizable: true
    ) {
        scene(fill: lightgreen, camera: new
    PerspectiveCamera()) {
        circle(centerX: 200, centerY:
    150, radius: 75, fill: blue) {
            translate(x: -100, y: -50,
    z: -50)
        }
    }
}
})
```

## SceneGraphBuilder - web

JavaFX web browser support.

### SceneGraph.htmlEditor

*Generated Object*

javafx.scene.web.HTMLEditor

*Attributes*

This node takes the attributes from the `javafx.scene.web.HTMLEditor` class. see [javafx.scene.web.HTMLEditor](#)

*Content*

This node takes the common node content, effects, transforms, and input.

**Usage**

Creates a HTMLEditor JavaFX node.

**Examples**



```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

def url = "http://www.yahoo.com";
GroovyFX.start({
    def sg = new SceneGraphBuilder();

    def webEngine = sg.webEngine(location:
url);
    def goAction = {
webEngine.load(urlBox.getText()) };
    def stage = sg.stage(
        title: "HTML Editor (Groovy)",
        visible: true,
    ) {
        scene(fill: hsb(128, 0.5, 0.5, 0.5),
width: 400, height:400 ) {
            borderPane() {
                html = htmlEditor()
                bottom(align: "center",
margin: [10,0]) {
                    button(text: "Print
HTML", onAction: { println html.htmlText})
                }
            }
        }
    }
});

```

## SceneGraph.webEngine

### *Generated Object*

javafx.scene.web.WebEngine

### *Attributes*

This node takes the attributes from the javafx.scene.web.WebEngine class. see [javafx.scene.web.WebEngine](#)

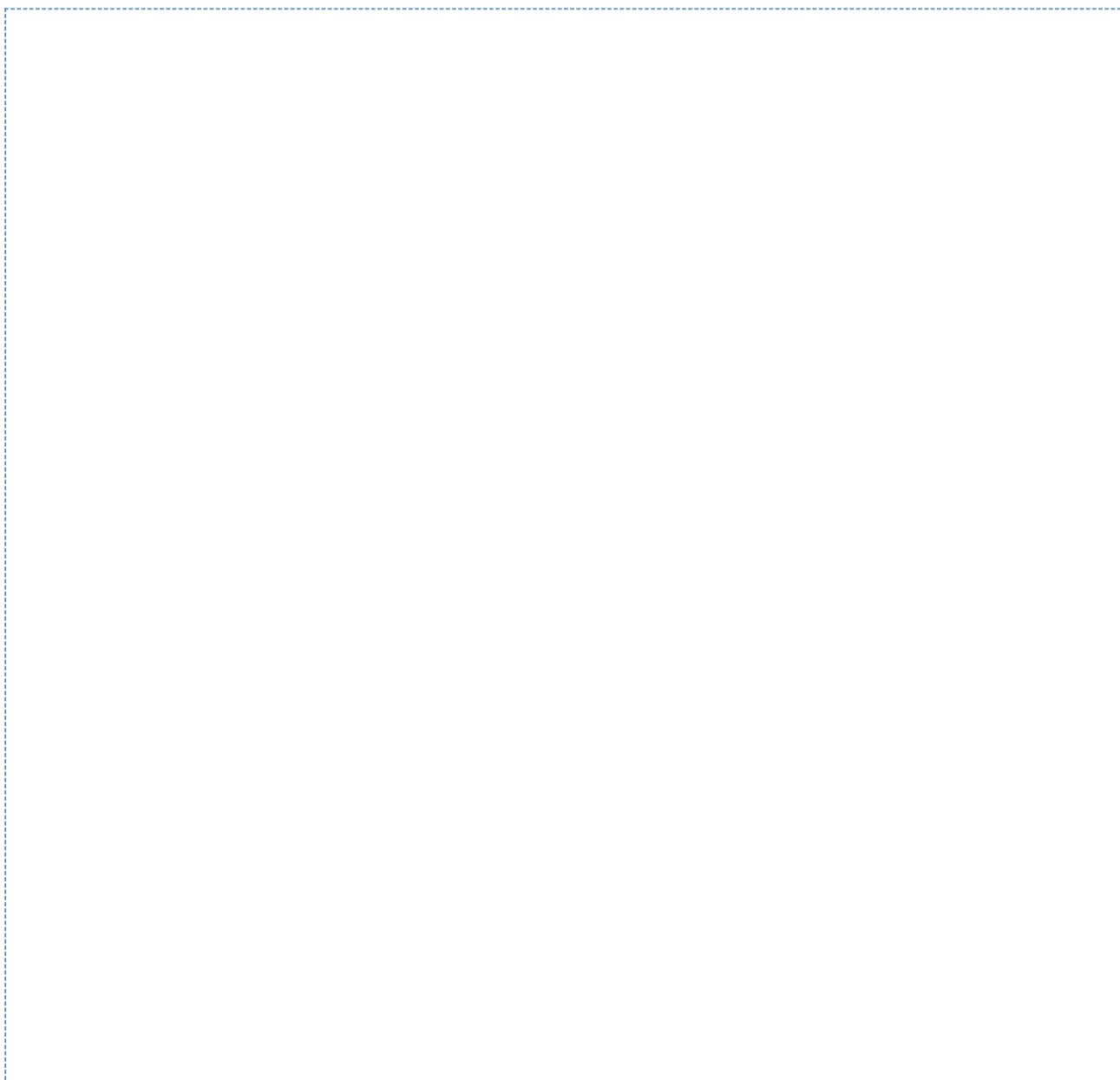
### *Content*

This node takes no content.

### *Usage*

Creates a WebEngine.

### *Examples*



```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

def url = "http://www.yahoo.com";
GroovyFX.start({
    def sg = new SceneGraphBuilder();

    def webEngine = sg.webEngine(location:
url);
    def goAction = {
webEngine.load(urlBox.getText()) };
    def stage = sg.stage(
        title: "Web Engine (Groovy)",
        visible: true,
    ) {
        scene(fill: gray, width: 640,
height: 500) {
            vbox() {
                hbox() {
                    urlBox = textBox( text:
url, action: goAction, prefWidth:400);
                    button(text: "Go",
onAction: goAction)
                }
                wv = webView(engine:
webEngine)
            }
        }
    }
});

```

## SceneGraph.webView

### *Generated Object*

javafx.scene.web.WebView

### *Attributes*

This node takes the attributes from the javafx.scene.web.WebView class. see [javafx.scene.web.WebView](#)

### *Content*

This node takes the common node content, effects, transforms, and input.

### *Usage*

Creates a WebView JavaFX node.

### *Examples*



```

import groovyx.javafx.GroovyFX
import groovyx.javafx.SceneGraphBuilder

def url = "http://www.yahoo.com";
GroovyFX.start({
    def sg = new SceneGraphBuilder();

    def webEngine = sg.webEngine(location:
url);
    def goAction = {
webEngine.load(urlBox.getText()) };
    def stage = sg.stage(
        title: "Web Engine (Groovy)",
        visible: true,
    ) {
        scene(fill: gray, width: 640,
height: 500) {
            vbox() {
                hbox() {
                    urlBox = textBox( text:
url, action: goAction, prefWidth:400);
                    button(text: "Go",
onAction: goAction)
                }
                wv = webView(engine:
webEngine)
            }
        }
    }
});

```

## TimelineBuilder

Provides Groovy DSL support for creating JavaFX timelines.

Timeline durations may be represented using the format [number.type](#), as 1.s is one second.

Options are:

**ms** - millisecond, eg 200.ms

**s** - second, eg 30.s

**m** - minute, eg 1.m

**h** - hour, eg 2.h

### TimelineBuilder.timeline

#### *Generated Object*

Creates a `javafx.animation.Timeline` instance.

#### *Attributes*

This node takes the attributes defined on the `javafx.animation.Timeline` class. see [javafx.animation.Timeline](#)

#### *Content*

This node may take one or more `at` nodes. see

#### *Usage*

Creates a timeline object.

#### *Examples*

```
import groovyx.javafx.TimelineBuilder;  
import javafx.animation.Interpolator;  
import groovyx.javafx.GroovyFX  
import  
org.codehaus.groovyfx.javafx.binding.PropertyBinding;  
import javafx.application.Platform;  
  
x = 5.0;  
y = 5.0;
```

```
GroovyFX.start({
    def tlb = new TimelineBuilder();
    def tl = tlb.timeline(cycleCount: 2,
onFinished: { Platform.exit()}) {
        at (10.s,
            onFinished: { println "x =
${this.x}, y = ${this.y}" }
        )
        {
            change (this,"x") {
                to 2.0
                tween "ease_both"
            }
            change (this,"y") {
                to 125
            }
        }
    }
}
```

```
println( tl )
tl.play();
});
```

## **TimelineBuilder.timeline.at**

### *Generated Object*

Creates a javafx.animation.KeyFrame instance.

### *Value*

This object takes a duration value.

### *Attributes*

This node takes the attributes defined on the javafx.animation.KeyFrame class. see [javafx.animation.KeyFrame](#)

The onFinished attribute may take either a javafx.event.EventHandler object or a groovy closure object. The groovy closure will be called with the javafx.event.Event object as a single argument..

### *Content*

This node may take one or more \*change \*nodes. see

### *Usage*

Creates a timeline KeyFrame.

### *Examples*

```
import groovyx.javafx.TimelineBuilder;
import javafx.animation.Interpolator;
import groovyx.javafx.GroovyFX
import
org.codehaus.groovyfx.javafx.binding.Property
yBinding;
import javafx.application.Platform;

x = 5.0;
y = 5.0;
```

```
GroovyFX.start({
    def tlb = new TimelineBuilder();
    def t1 = tlb.timeline(cycleCount: 2,
onFinished: { Platform.exit()}) {
        at (10.s,
            onFinished: { println "x =
${this.x}, y = ${this.y}" }
        )
        {
            change (this,"x") {
                to 2.0
                tween "ease_both"
            }
            change (this,"y") {
                to 125
            }
        }
    }
}
```

```
println( tl )
tl.play();
});
```

## **TimelineBuilder.timeline.at.change**

### *Generated Object*

Creates a javafx.animation.KeyValue instance.

### *Value*

This object takes a either a javafx.beans.WritableValue argument, or takes an object instance and a property name within that instance. These are used as the property to change as a result of the timeline.

### *Attributes*

This node takes the attributes defined on the javafx.animation.KeyFrame class. see [javafx.animation.KeyFrame](#)

The onFinish attribute may take either a javafx.event.EventHandler object or a groovy closure object. The groovy closure will be called with the javafx.event.Event object as a single argument..

### *Content*

This node must have one "to" node and may have one "tween" node. see

### *Usage*

Creates a timeline KeyValue for a key frame..

### *Examples*

```
import groovyx.javafx.TimelineBuilder;
import javafx.animation.Interpolator;
import groovyx.javafx.GroovyFX
import
org.codehaus.groovyfx.javafx.binding.Property
yBinding;
import javafx.application.Platform;

x = 5.0;
y = 5.0;
```

```
GroovyFX.start({
    def tlb = new TimelineBuilder();
    def t1 = tlb.timeline(cycleCount: 2,
onFinished: { Platform.exit()}) {
        at (10.s,
            onFinished: { println "x =
${this.x}, y = ${this.y}" }
        )
        {
            change (this,"x") {
                to 2.0
                tween "ease_both"
            }
            change (this,"y") {
                to 125
            }
        }
    }
}
```

```
println( tl )
tl.play();
});
```

## Gaelyk

Gaelyk is a lightweight Groovy toolkit for developing and deploying Groovy applications on Google App Engine.

Gaelyk is a thin layer on top of the Google App Engine SDK, that is aimed at simplifying the creation of applications written in the Groovy dynamic language to be deployed on GAE. Thanks to Groovlets (servlet scripts basically) and templates, and entities, you can follow an MVC paradigm for creating your applications. Several nice syntax shortcuts simplify the usage of the low-level SDK APIs, such as an email.send to: "recipient@gmail.com" shorthand and more.

If you want to learn more about this little framework, please have a look at the extensive [tutorial](#).

Gaelyk was also presented recently at the Devovx conference in Belgium, along with Patrick Chanezon describing the GAE platform.

You can view the [slides of the presentation](#)

You can [download Gaelyk](#) and a template project in the download section of the website and [get involved in the community](#). Particularly, you can join the [Gaelyk Google Group](#) for posting questions and chatting with the users and developers.

Gaelyk is an Open Source project licensed under the terms of the Apache Software License 2.0. The project is [hosted on GitHub](#).

A number of applications online already use this lightweight toolkit.

- For example, the [Groovy Web Console](#) lets you execute and share Groovy scripts online.
- The [Gaelyk website](#) itself is of course a Gaelyk application.
- The iUI iPhone JavaScript toolkit [demo website](#) is also powered by Gaelyk (best viewed with an iPhone).
- The [Averone's](#) company website is developed with Gaelyk.
- The [Phone4Water](#) website.

## GMaven

### GMaven

Documentation for [GMaven](#) has moved.

### Groovy-eclipse-compiler

An alternative approach to Groovy and Maven integration is to use the groovy-eclipse-compiler plugin for Maven. There are benefits and disadvantages to each approach. For more information on this as well as a description of the differences between gmaven and groovy-eclipse-compiler, see [Groovy-Eclipse compiler plugin for Maven](#).

### Groovy Transforms

Provides additional [AST Transformations](#)

## Contribution Overview

This module works like a sandbox for additional [AST Transformations](#) that can't be included in Groovy core for a particular reason, like additional dependencies or scope. Contributions are welcome!

### *Team Members*

Andres Almiray [aalmiray at users dot sourceforge dot net]

## Download

[groovy-transforms-0.1](#)

### *Installing*

Download the binary package and drop it in your \$HOME/.groovy/lib directory

### *Pre-requisites*

Each AST may require additional libraries or configuration settings, please refer to their documentation page before attempting to use them.

## Documentation

The following is a list of currently available AST Transformations

- [Scalify](#) — Provides integration with the Scala Programming language

### *Contributing*

Please contact the team members by e-mail or send a message to the Groovy mailing lists.

### *Mailing List(s)*

<http://groovy.codehaus.org/Mailing+Lists>

### *Issue tracker*

<http://jira.codehaus.org/secure/BrowseProject.jspa?id=10242>

## Scalify

Provides integration with the [Scala Programming language](#)

### *Description*

@Scalify simplifies the task of integrating Groovy and Scala code. This transformation adds the required bytecode to your Groovy classes to make them appear as native Scala classes. It also helps when implementing or extending a Scala trait/class from Groovy.

### *Usage*

Place @Scalify at the class level, example;

```

// -- begin scala code
import scala.reflect.BeanProperty

trait Output {
    @BeanProperty var output: String
}

// -- end Scala code

// -- begin Groovy code
import groovyx.transform.Scalify

@Scalify
class GroovyOutput implements Output {
    String output
}

```

When the trait Output is compiled it will generate bytecode similar to

```

public interface Output {
    public abstract void
setOutput(java.lang.String);
    public abstract java.lang.String
getOutput();
    public abstract void
output_$eq(java.lang.String);
    public abstract java.lang.String output();
}

```

The first pair of methods is your typical POJO accessors (generated by Scala's @BeanProperty) which Groovy can

handle quite well. The second pair of methods represent Scala's *native* accessors, these will be written by @Scalify (if not present already in your class definition). Additionally @Scalify will make sure that your class implements `scala.ScalaObject` if it does not already.

One last trick up the sleeve is that @Scalify will generate operator friendly methods, these are the currently supported methods:

Operator	Groovy	Scala
+	plus()	\$plus()
-	minus()	\$minus()
*	multiply()	\$times()
/	div()	\$div()
%	mod()	\$percent()
^	xor()	\$up()
&	and()	\$amp()
	or()	\$bar()
**	power()	\$times\$times()
<<	leftShift()	\$less\$less()
>>	rightShift()	\$greater\$greater()
-	negative()	unary_\$minus()
+	positive()	unary_\$plus()
~	bitwiseNegate()	unary_\$tilde()

## ❌ Full compatibility

While @Scalify can generate all the required boilerplate code it cannot (at the moment) generate a required class attribute (called pickle) that the Scala compiler uses to gather metadata information about a Scala class. This means you won't be able to call a property setter using the short notation, in other words

```
new GroovyOutput().output = "something"
```

will result in a compilation error while the following will work

```
new GroovyOutput().output_$eq("something")
```

### Dependencies

Make sure to have the Scala compiler & libraries on your classpath. You will also need `asm`, grab the latest version available with your Groovy distribution

## Related Projects

This page lists projects using Groovy in some way. Please add your project to this page.

### Lightweight containers, J2EE & AOP

- [Spring's Groovy Support](#)
- [Mule](#) open source ESB has full [Groovy Scripting support](#). There is also a [scripting example](#) using Groovy and Python.
- [NanoContainer](#) is a script front-end for PicoContainer.
- [PicoContainer](#) has [Groovy Pico](#). here's an [example test case](#)
- [ServiceMix](#) is an open source JBI / ESB and has full [Groovy support](#)
- [OpenEJB](#) which has a Groovy telnet shell for querying EJBs while the server's live 😊
- [GAP](#) using Groovy [dynAop](#) and Pico
- The Turbine [Fulcrum Groovy Service](#) executes Groovy scripts managed by the Fulcrum ResourceManager Service. It provides the following features:
  - caching of Groovy scripts to improve performance
  - thread-safe execution of compiled Groovy scripts
  - seamless integration with the existing Avalon infrastructure.
- [1060 NetKernel](#) has [Groovy support](#) for creating scripted services and service orchestration.
- [GrailsFlow](#) is a Workflow and Business Process Engine written in Groovy and Grails.

### Blog/Wiki, CMS, Portal projects

- [SnipSnap](#) has some nice Groovy support allowing to admins to write [GSP](#) pages.
- [XWiki](#) also comes with [GSP](#) support and has some nice examples of using it, e.g. [live RegEx testing](#).
- [biscuit](#) is an all-groovy wiki made by Groovy committer Jeremy Rayner.
- [blojsom](#)
- [WidgetWeb](#)
- [eXo Platform](#)
- [Magnolia CMS](#) provides full Groovy support starting with version 4.3, see [Boris' blog post about Groovy](#)

[support in Magnolia CMS \(and Greg's video!\)](#)

- [Portofino](#)

## Web frameworks

- [NanoWeb](#)
- [RIFE](#)
- [Struts 2](#) Create Struts 2 actions in Groovy
- [Simple](#) Advanced Groovy templating, provides a lightweight version of Struts Tiles
- [AribaWeb](#) Full stack component-based framework for creating AJAX-enabled apps; embeds Groovy with full support for scripting components and business objects.
- [Groovy Tapestry](#)
- [GvTags](#) Template engine with tag lib support for Groovy and JSP tag library
- [Woko: POJOs on the Web!](#) Full stack DDD framework for building Object Oriented, Multi Profile webapps, with support for Groovy
- [OOWeb](#) Lightweight, fast, object-oriented web framework. Can be embedded by groovy.
- [Gracelets](#) A combination of Groovy and JSF/Facelets allowing you to write simple, compact and reloadable pages, controllers and libraries all inside the Servlet/JSF framework. Learn more about it [here](#).
- [Jaiwls](#) Component oriented Framework with bundled Servlet-Webserver (Jetty) and HTML Widget library. Can use static compiled java-classes or runtime Groovy-scripts.
- [Portofino](#) is an open source web framework. Among its features: content management, CRUD, database connectivity, calendars, charts, security and easy customization.

## Testing Tools

- [Canoo WebTest](#) allows specifying Test Steps scripted in Groovy, bundling a series of Tests Steps with the MacroStepBuilder, and creating a whole WebTest using the AntBuilder. See [GroovyWebTest](#)
- [Spock](#) is a testing and specification framework for Java and Groovy applications
- [accept4j](#) is an acceptance testing tool that bridges the gap between automated tests and business requirements in Java applications

## Web Service Tools

- [soapui](#) Has a Groovy test step. The Groovy Script step allows you to specify an arbitrary Groovy script during the execution of a TestCase. The script has full access to the soapui object model and can thus perform a variety of tasks. Read more: [Groovy Scripts](#)

## Other Server Software

- [EasyGSP](#) allows you to use Groovy to create websites in a LAMP stack.
- [Groovy Runner](#) is a way to run Groovy scripts (and templates) in a LAMP server setup. Use groovy the same way you use PHP.
- [GroovyConsole servlet/portlet](#) is a servlet/portlet that let you type groovy commands and executes them on the server, returning the output - useful for playing with the living e

## Scientific Programming

- [GroovyLab](#) : a MATLAB-like scientific programming environment based on Groovy with extensive plotting support, default access to the best Java scientific libraries, user-friendly interface, easy installation of toolboxes, Computer Algebra support and many other features

## Other

- [GroovyRules](#) is a JSR-94 compliant lightweight rules engine that permits defining rules in Groovy
- [GroovyHelp](#) is a powerful API documentation view, search and compare utility.
- [PLEAC-Groovy](#) is an implementation of the Solutions of the Perl Cookbook in the Groovy language.

- [FreeMind](#) is a mindmapping tool that supports [scripting mindmaps with Groovy](#). Furthermore, the Groovy compiler can produce a visual representation of Groovy code as a FreeMind \*.mm file. Start the compiler with the -Dantlr.ast=mindmap option.
- [Freeplane](#) is a successor to FreeMind with greatly enhanced [Groovy scripting support](#) with a clean scripting API. Scripts can be bound to menu items and allow to implement Freeplane add-ons. Spread-sheet-like formulas are Groovy-based.
- [Funkee](#) is a Groovy Scripting Host that shows a nice little smiley on your desktop that you can click to open a Groovy source code editor, or for running Groovy scripts, as well as other scripting languages.
- [GAQ](#) is a lightweight framework for developing, scheduling, running and reviewing automated tasks. It combines the scripting ease of Groovy, the power of Ant, and the cross platform scheduling of Quartz.
- [Luxor XUL](#) is an XUL language extension that comes with [Groovy Support](#).
- [Hierarchical Inherited Rule-Interpreted XML](#) is a dynamic XML Engine that creates dynamic DOMs built on dynamic rules and expressions where Groovy is available to be used as the expression interpreter.
- [JFacets: the Profile Based framework](#), proposes a clean and consistent solution to the recurrent "user profiling" issues (users/roles, security, personalization etc) through the concept of facets, that can be written in Groovy
- [Grails Camelot](#): an Idea Repository, and a grails showcase.
- [Groovy MDA](#) A model driven architecture tool that uses groovy. By default it can be used to generate JPA entity beans from a UML model. You can also easily add your own templates to generate other types of projects.
- [ORM Lite](#): a light ORM library in 1000 lines of Groovy code
- [mixGin](#): MixGin provides a fluent API for an easy interaction between groovy and java fragments within java code. Those code can be mixed arbitrarily, including automatic category detection and data access of variables declared in groovy within java and vice versa
- [Groovy Eclipse EMF Builder](#) Jörn Dinkla's project for using the Eclipse Modeling Framework with Groovy.
- [Disk usage tool](#) Jörn Dinkla's project for investigating the harddisk for usage information; based on Groovy and JFreeChart.
- [CHSM/Groovy](#) allows concurrent, hierarchical, state machines to be constructed using Groovy markup for modeling and controlling reactive systems.
- [Market Data Works](#) uses Groovy to customize and perform Market Data transformations.
- [Metawidget](#) generates User Interfaces at runtime based on GroovyBean properties and annotations. Supports GWT, JSF, JSP, Spring, Struts, Swing and more.
- [TEAM CTHULHU's Plexus](#) an open source, peer-to-peer, 3D game which uses Groovy, Free Pastry, and Sauerbraten
- [SimpleJPA with Groovy](#) - SimpleJPA is a java persistence API layer for Amazon SimpleDB.
- [GPars](#) (the project previously known as GParallelizer): allows Groovy code to easily handle tasks concurrently and in parallel, leveraging JSR-166y's *ForkJoin* framework and *Parallel Arrays*, Java's executor services, JCSP, and other libraries. GPars is now bundled as standard with Groovy distributions.
- [Repast Symphony](#) is a free and open source agent-based modeling toolkit that [uses Groovy](#) to simplify agent specification.
- [OpenCoeli](#) : an Astronomy application written in Groovy
- [CodeNarc](#) - Static Analysis for Groovy source code
- [AJAX Groovy Shell](#) with a vt100.js interface
- [SexScripts](#) allows creation and share of erotic-oriented scripts made with Groovy and running in a Swing application (including a Groovy editor)
- [GrooScript](#) converts groovy code to javascript.
- [GVM](#), the Groovy enVironment Manager. Allows easy installation of Groovy technologies.