

JetGroovy: Groovy with pleasure



2007

# GRAILS

## EXCHANGE

<http://www.grails-exchange.com> | <http://www.grails.org> | <http://skillsmatter.com>



## Who We Are

- Eugene Vigdorichik
  - 10 years in software development
  - Today - Software Architect at JetBrains
- JetBrains
  - World-known vendor of intelligent SD tools
- IntelliJ IDEA
  - Famous productivity-oriented Java IDE



# JetGroovy Plugin

Started in March 2007

Source code is available at  
<http://svn.jetbrains.org/idea/Trunk/groovy>

3 active developers + external contributions

70 kLoc of code

Part of the plugin is written in Groovy

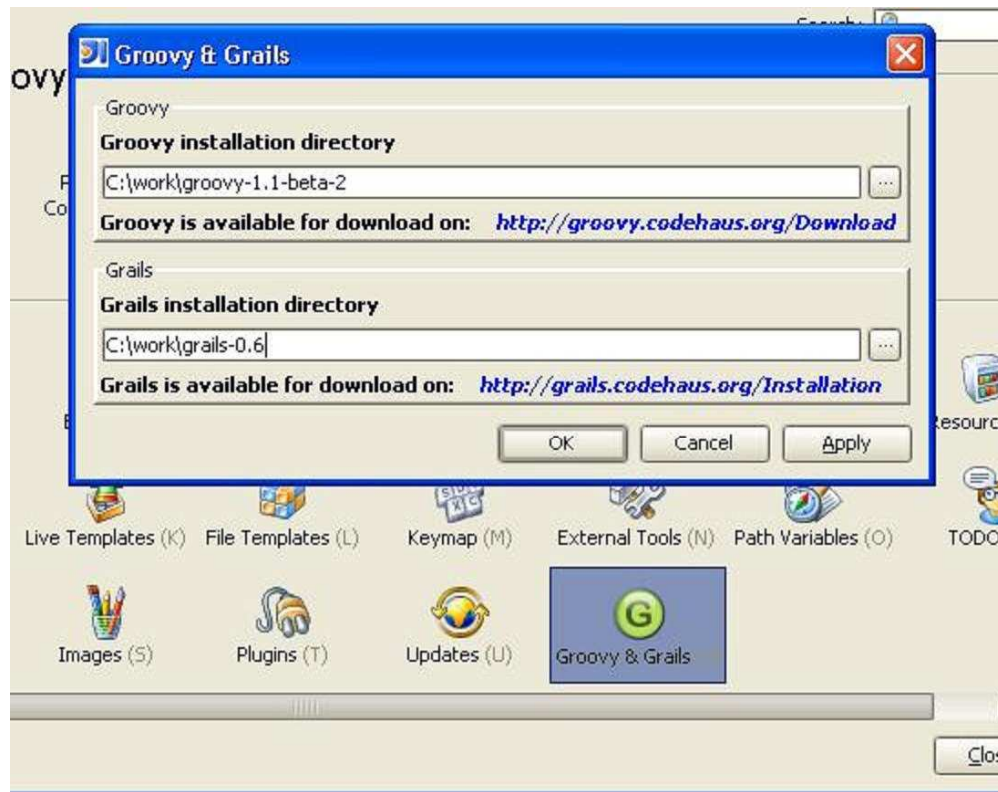


## What's so special about it?

- Easy and intuitive configuration
- More than just syntax highlighting
- Seamless interoperability with Java
- Unparalleled type inference
- Tight integration with advanced IDE debugger
- Intelligent GSP support



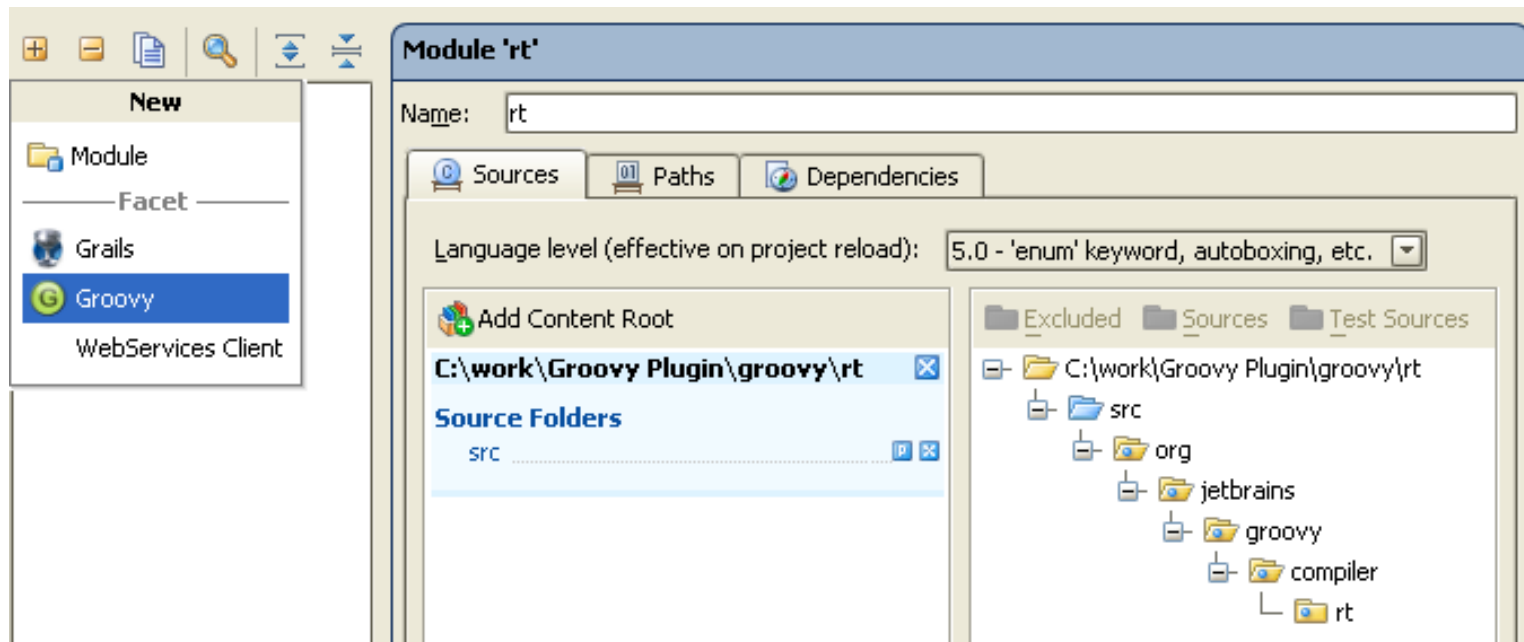
# Configuration



Necessary jars  
are kept in  
application level  
libraries



## Configuration (2)



- Groovy and Grails support for module is configured through facets. No separate module type needed.



## Basic Actions

- Syntax highlighting
- Formatting code
- File structure view
- Structural “surround with”
- Actions for Groovy class and script creation

The same feature set that is available for Java



## JetGroovy is type-based

- Groovy allows for optional typing
- Types are mandatory in Java
- Types may be often inferred

When types are not available, pessimistic assumptions are made



# Untyped References

```
if(compileList) {  
    // if there are Java sources do a complete re-compile  
    if(cleanJavaCompile && compileList.find { it.name.endsWith(".java") }) {  
        compileList.clear()  
        def m = new IdentityMapper()  
        def sfs = new SourceFileScanner(this);  
        for(srcPath in src.list()) {
```

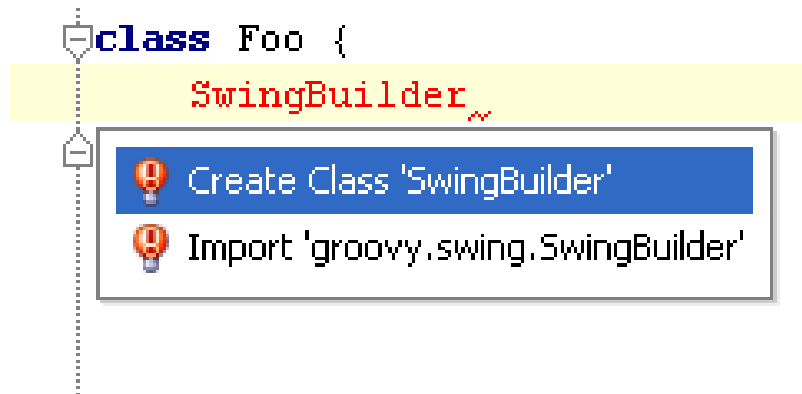
Cannot determine type statically

- Method
  - endsWith(String):boolean of class java.lang.String
- Dynamically typed usages (1 usage)
  - Found usages (1 usage)
    - Src (1 usage)
      - C:\work\grails-0.6\src\groovy\org\codehaus\groovy\grails\compiler (1 usage)
        - GrailsCompiler.groovy (1 usage)
          - {110, 54} if(cleanJavaCompile && compileList.find { it.name.endsWith(".java") }) {
- Found usages (45 usages)



# Compiler Errors

```
class Foo {  
    SwingBuilder ~
```



```
class Foo implements List<String> {
```



- All `groovyc` errors are reported
- Auto-import is available on incorrect class references
- “Implement methods” quick-fix



## Code Inspections

```
if(classpath) configuration.classpath = classpath.toString()  
configuration.targetDirectory = destdir  
if(encoding) configuration.sourceEncoding = encoding  
Cannot assign 'java.io.File' to 'java.lang.String'
```

- Check whether:
  - Assignment is correct
  - Method is applicable
  - Private API is used from outside
- Provide warnings instead of errors



# Intention Actions

```
ForToEach.groovy  
  
int[] arr  
for (i in arr) {  
    // Intention: Replace with ".each"  
}
```

```
if (item instanceof Item && item?.description != null) {  
    // Intention: Flip &&  
    // Intention: Replace && with ||  
    // ... = "<html><body>${item.description}</bod
```

- 20 Groovy specific intentions
  - Convert `for` to `each`
  - Data types conversions
  - Control-flow related actions



## Groovy + Java

Groovy needs to call Java:

- Lots of available APIs/legacy code
- Static typing = better performance

Java might call Groovy

- Partial refactoring
- Explicit design

IDE should understand cross-language references



## Joint Compilation

### 3-phase process:

- Java stubs are generated
  - Java compiler is invoked on user-written Java code and stubs
  - Groovy compiler is called with Java output included in classpath
- 
- The same feature is implemented by the groovy-core team



# Cross-language Actions(1)

The screenshot shows an IDE with two code editors. The top editor, 'Container.groovy', contains the following code:

```
class Container {  
    Component component  
  
    {  
        component.doWork()  
    }  
}
```

The bottom editor, 'Component.java', contains the following code:

```
public class Component {  
    public void doWork() {  
        //....  
    }  
}
```

A tooltip for the 'doWork()' call in the Groovy file shows the signature: 'Component public void doWork()'. A panel titled 'Usages of doWork() in Project Files' is open, showing a tree view of usages:

- Method
  - doWork():void of class Component
- Found usages {1 usage}
  - Test (1 usage)
    - (1 usage)
  - Container.groovy (1 usage)
    - (5, 19) component.doWork()

- Navigation
- Find Usages
- Rename



## Cross-language Actions(2)

```
class Container {
    String bean
}
```

```
public class User {
    Container container;

    void setBean (java.lang.String p)
    container.setBean("");
}
```

Usages of bean in Project Files

- Field
  - bean
- Found usages {1 usage}
  - Test (1 usage)
    - (1 usage)
  - User (1 usage)
    - (5, 19) container.setBean("");

- Groovy properties are recognized as getters from Java



# Refactorings

```
class CharUtils {  
    def isUpper(def c) {  
        Character.isUpperCase c  
    }  
}
```

- Rename
- Move Class
- Introduce Variable
- Inline Variable

**Introduce Variable**

Specify type explicitly

Variable of type:

Name:

Declare final     Replace all occurrences

OK    Cancel    Help

In progress:

- Extract Method
- Inline Method



## Code Generation

The screenshot shows an IDE window titled 'MyList.groovy'. The code is `class MyList implements List {`. A 'Select methods to implement' dialog is open, showing a tree view of `java.util.List` with methods: `size():int`, `isEmpty():boolean`, `contains(Object):boolean`, `iterator():Iterator <E>`, and `toArray():Object[]`. The `size():int` method is selected.

- Available as:
  - Quick fixes
  - Editor actions
- Aware of Generics



## Code Completion(1)

```
G FileUtils.groovy  
  
def iterate(File[] files) {  
    files.ea  
}
```

m	each (Closure closure)	void
m	eachWithIndex (Closure closure)	void

- GDK is recognized
- Type of qualifier should be known
- References in the scope with same qualifier are suggested when type is unknown.



## Code Completion(2)

```
SwingExample.groovy
import groovy.swing.SwingBuilder

def builder = new SwingBuilder()

builder.pa
```

panel ()	JPanel
passwordField ()	JPasswordField

- Groovy SwingBuilder methods and respective properties are completed

- Accounts for builder setting closure delegate

```
def builder = new SwingBuilder()

builder.panel(){
    but
```

button ()	JButton
buttonGroup ()	ButtonGroup



## Code Completion(3)

- Listener methods are suggested as closures

```
SwingExample.groovy
import groovy.swing.SwingBuilder

def builder = new SwingBuilder()

builder.panel {fo
```

P	focusable
P	focusCycleRoot
P	focusGained
P	focusLost
P	focusTraversalKeysEnabled
P	focusTraversalPolicy
P	focusTraversalPolicyProvider
P	font
P	foreground



## Code Completion(4)

G SpreadOperatorExample.groovy

```
def list = [0, 1, 2]
```

```
list*.di|
```

		div (Character character)	Number
		div (Number number)	Number

- Spread operator is applied to each list element
- If component type is inferred, then spread operator may be completed



# Type Inference

```
TypeInference.groovy
def v
if (f()) {
    v = 0
} else {
    v = 1.0
}

v.in
```

m	inject (Object o, Closure closure)	Object
m	inspect ()	String
m	intdiv (Character character)	Number
m	intdiv (Number number)	Number
m	intValue ()	int
m	invokeMethod (String s, Object o)	Object

- Type of the method is inferred from all return values
- Type of local variable is inferred from its initializer
- Type of “read” variable reference is inferred from assignments reaching this reference
- Java-like inference for generic methods’ type arguments



## Debug and Run(1)

BloglinesClient.groovy      Indexer.groovy

```
slurper.endDocument()  
opml = slurper.parse(callBloglinesListsub)
```

- Compile '...glinesClient.groovy'      Ctrl+Shift+F9
- Create "BloglinesClient"...
- Run "BloglinesClient"      Ctrl+Shift+F10
- Debug "BloglinesClient"

```
class AmbiguousInvocationTest extends GroovyTestCase {  
    def dump  
    void set  
    dump  
}
```

- Compile '...vocationTest.groovy'      Ctrl+Shift+F9
- Create "AmbiguousInvocationT..."...
- Run "AmbiguousInvocationT..."      Ctrl+Shift+F10
- Debug "AmbiguousInvocationT..."



# Debug(2)

The screenshot shows an IDE with a Groovy test file and a debug console. The test file contains the following code:

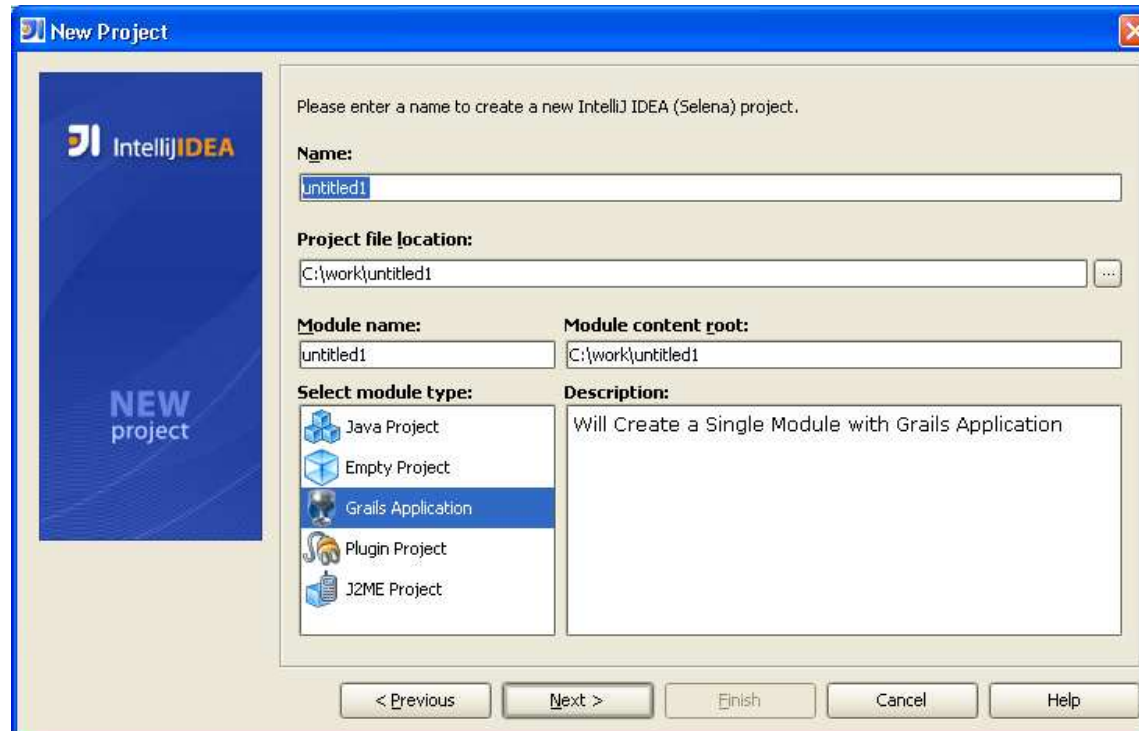
```
void testAmbiguousInvocationWithFloats() {  
    assert "float args" == dummy.foo("bar", 1.0f, 2.0f)  
    assert "float args" == dummy.foo("bar", (float)1, (float)2)  
    assert "float args" == dummy.foo("bar", (Float)1, (Float)2)  
}  
  
void testAmbiguousInvocationWithInts() {  
    assert "int args" == dummy.foo("bar", 1, 2)  
    assert "int args" == dummy.foo("bar", (int)1, (int)2)  
    assert "int args" == dummy.foo("bar", (Integer)1, (Integer)2)  
}
```

The debug console shows the following stack trace:

```
Debug AmbiguousInvocationTest  
Frames  
"main"@1 in group "main": RUNNING  
testAmbiguousInvocationWithFloats():18, AmbiguousInvocationTest  
invoke0():-1, NativeMethodAccessorImpl (sun.reflect)  
invoke():39, NativeMethodAccessorImpl (sun.reflect)  
invoke():25, DelegatingMethodAccessorImpl (sun.reflect)  
invoke():585, Method (java.lang.reflect)  
runTest():164, TestCase (junit.framework)  
runBare():130, TestCase (junit.framework)  
protect():106, TestResult$1 (junit.framework)  
runProtected():124, TestResult (junit.framework)  
run():109, TestResult (junit.framework)  
Variables  
this = {groovy.AmbiguousInvocationTest@321}"testAmbiguousInvocationWithFloats(groovy.AmbiguousInvocationTest@321)"  
dummy = {groovy.DummyMethods@741}  
metaClass = {groovy.lang.MetaClassImpl@742}"groovy.lang.MetaClassImpl@1d2b01b[class groovy.AmbiguousInvocationTest@321]"  
useAgleDoxNaming = false  
fName = {java.lang.String@743}"testAmbiguousInvocationWithFloats"  
Console  
Running: 0 of 2 testAmbiguousInvocationWithFloats  
AmbiguousInvocationTest (groovy)  
testAmbiguousInvocationWithFloats  
testAmbiguousInvocationWithInts  
Output Statistics  
"C:\Program Files\Java\jdk1.5.0_09\bin\java" -Xdebug -Xrunjdwp:transport=dt_socket,address=127.0.0.1:3030, Connected to the target VM, address: '127.0.0.1:3030', transport: 'socket'
```



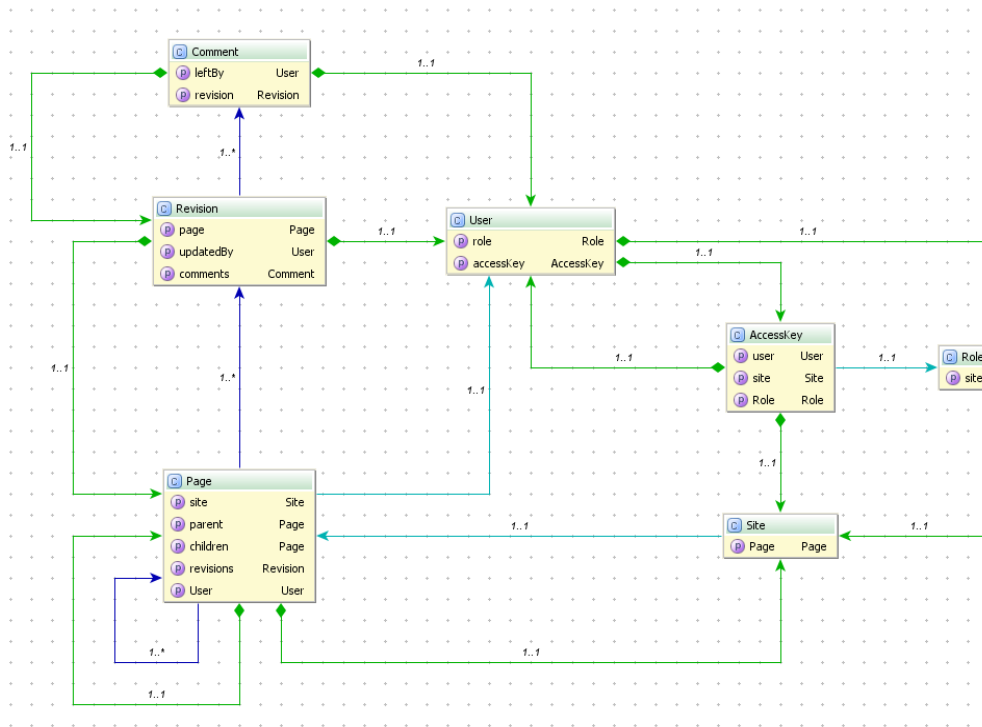
# Grails Project Setup



- Dedicated module type
- `grails create-app` is invoked



# Entity-Relation Diagram



- Visualizes your domain model
- Editable, with changes propagated to Groovy code:
  - Relation insertion
  - Relation deletion
  - Relation switch



# Groovy Server Pages

- Formatter
- Groovy support
- HTML code inspection/code assist

```
<html>
  <head><title>index.gsp</title></head>
  <body>
    <b><% println "hello gsp" %></b>
    <% wrd = "Groovy"
    for (c in wrd) {
      <%
    <h1><%=c.toLo%> <br /></h1>
    <%
  }
</body>
</html>
```

toLowerCase (char)	char
toLowerCase (int)	int



## Groovy Server Pages (2)

```
PervasivesTagLib.groovy
class PervasivesTagLib {
    static namespace = 'p'

    def log = {
    }
}

index.gsp
<p:
p:log
```

- GSP specific tag completion:
  - Default tags
  - User-defined tags



## And We Keep Improving...

- More dynamic stuff in the future
- Extended set of refactorings
- Advanced Groovy-specific code inspections
- “Create from usage” intention actions
- More Grails goodies



## How to get the stuff

- Download and install IntelliJ IDEA 7.0
- Install JetGroovy using Plugin Manager
- Download Groovy from <http://groovy.codehaus.org>
- Download Grails from <http://grails.codehaus.org>



# JetGroovy

# Q&A



## We Need Your Feedback!

- We are keeping JetGroovy Confluence in sync with the development  
<http://www.jetbrains.net/confluence/display/GRVY/Groovy+Home>
- You are welcome to post bugs and feature requests at  
<http://www.jetbrains.net/jira/browse/grvy>
- JetGroovy has a dedicated forum/newsgroup at  
<http://www.intellij.net/eap>