SpringSource Tool Suite 2.9.2

- New and Noteworthy -

Martin Lippert 2.9.2 May 24, 2012 Updated for 2.9.2.RELEASE

www.springsource.com

springsource
A division of vmware

ENHANCEMENTS – 2.9.2

General Updates

vFabric tc Server 2.7.0

STS ships now with support for and the distribution of the latest vFabric tc Server Developer Edition 2.7.0.

Fixed Bugs

- IDE-1246: An internal error occurred during: "Updating Maven Configuration". NPE in RooProjectConfigurator.doConfigure
- IDE-1244: RequestMappings does not include mappings defined on interfaces
- IDE-1243: Search Pointcut Matches returns no results
- STS-2592: RequestMappings and Controllers do not appear in the Spring Explorer view
- STS-2571: update equinox weaving for AJDT to latest from Eclipse 3.8 streams
- STS-2569: NullPointerException in LegacyProjectChecker
- STS-2563: Grails 2.0.3 plugin update
- STS-2553: update tc Server integration for upcoming tc Server 2.7
- STS-2493: Spring nature not automatically added when project has spring-core dependency
- STS-2490: Grails Plugin Manager will not launch
- STS-2274: No property tester contributes a property org.springframework.ide.eclipse.beans.core.model.isInfrastructureBean to type class org.springframework.ide.eclipse.beans.core.internal.model.Bean

ENHANCEMENTS – 2.9.1

General Updates

vFabric tc Server 2.6.4

STS ships now with the latest vFabric tc Server Developer Edition 2.6.4.

Fixed Bugs

- STS-2271: For Spring Integration: Add Visual Support for Payload Enricher
- STS-2498: gateways should display connections to error-channels
- STS-2497: gateways don't display transitions defined in method sub-elements
- STS-2488: When using free-form layout the conector lines cannot be predictably drawn
- STS-2501: Chain elements in SI graphs are transposed in manual layout mode
- IDE-1239: Cant add set to actions in Graphical Editor of Web Flow
- IDE-1232: SpringSource Tool Suite Content Assist Slow, Lags or Hangs
- STS-2502: can't build/deploy mvc sample

ENHANCEMENTS – 2.9.0

General Updates

Eclipse Indigo SR2 (3.7.2)

STS now ships on top of the latest Eclipse Indigo SR2 (3.7.2) release.

vFabric tc Server 2.6.3

STS ships now with the latest vFabric tc Server Developer Edition 2.6.3.

Spring Roo 1.2.1

STS updated the distributed version of Spring Roo to the latest Spring Roo 1.2.1 release.

Spring 3.1.1

The integrated Spring version that STS is using internally got updated to Spring 3.1.1.

Maven Integration for Eclipse – WTP Extension 0.15

The STS distribution now ships with the WTP integration for m2e in version 0.15.

Grails 2.0.1

The Grails version that you can get from the extension install has been upgraded to 2.0.1.

AJDT

The AJDT version included in STS now includes an early build of AspectJ 1.7.0.

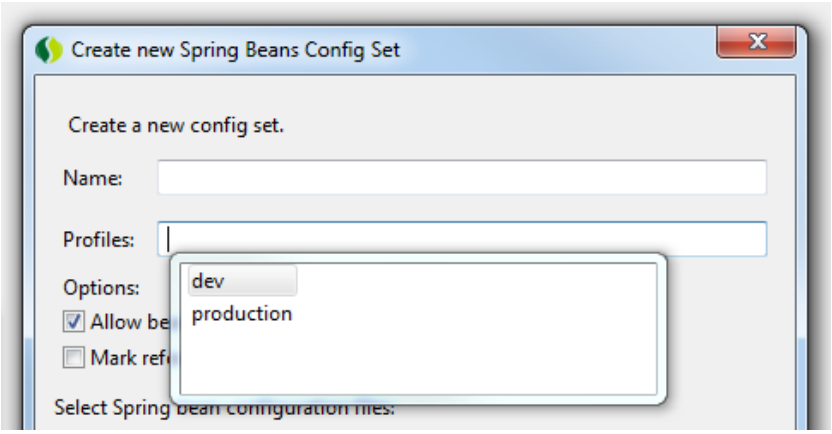
Groovy-Eclipse

The Groovy-Eclipse version available from the dashboard is the 2.6.1 release.

Spring Development Tools

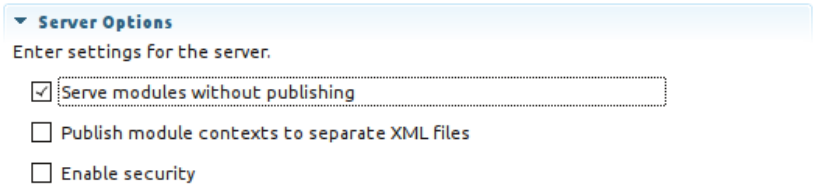
Improved Spring 3.1 Profile Support

In this release we added content assist for selecting profiles to be included while defining a Spring Beans config set. Currently the content assist only works on profiles that are defined in a Spring Beans config xml file, but we are working on making this work with profiles defined through Java annotations as well.



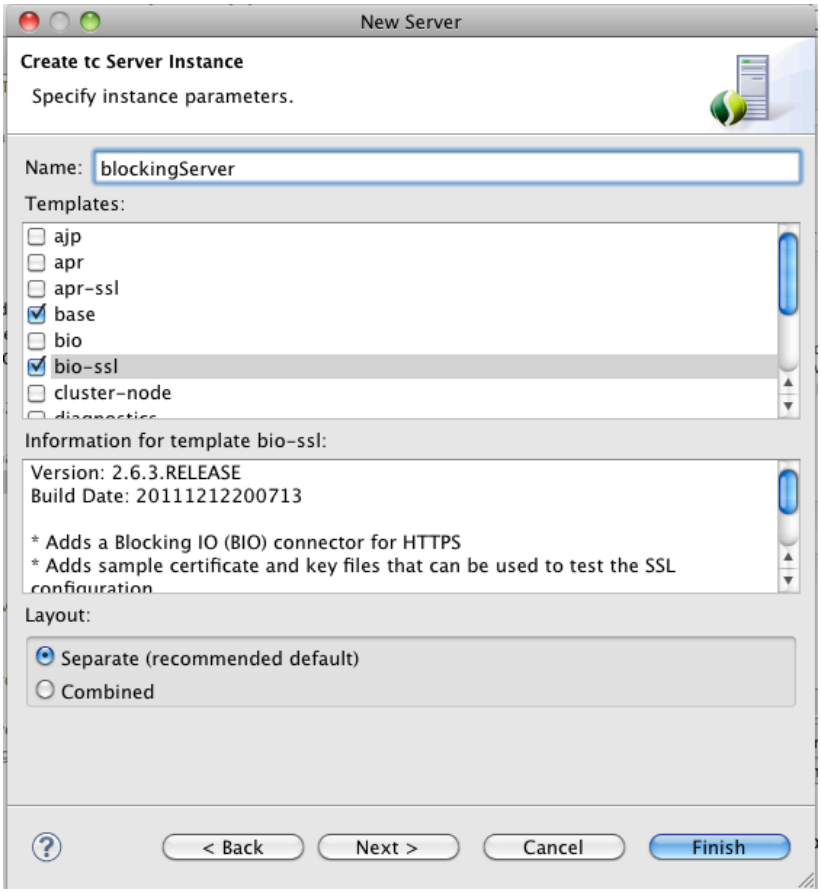
"Serve modules without publishing" now available for tc Server

The "Serve modules without publishing" option can now be enabled in the tc Server editor. When selected web content is directly served from the workspace not requiring publishing.



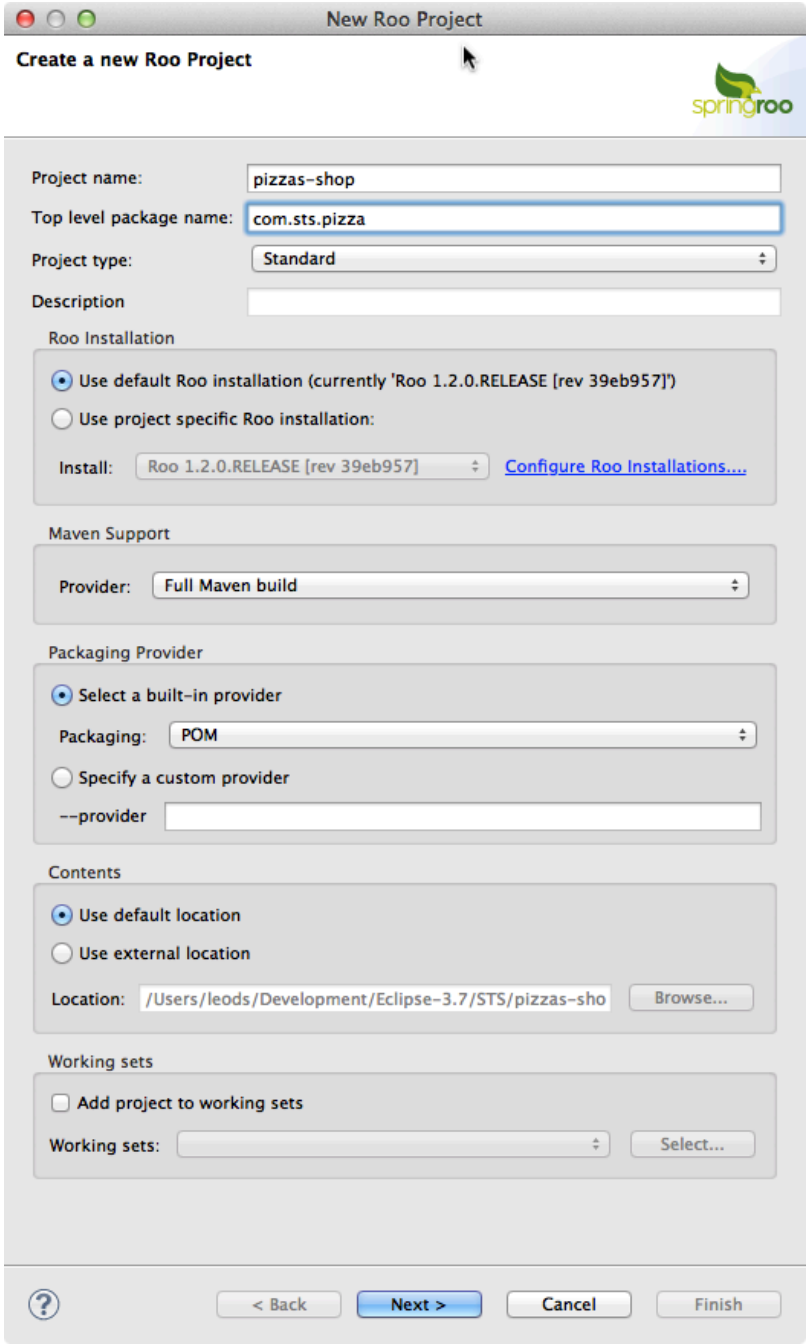
Improvements to vFabric tc Server instance creation wizard

We improved the internal workings of the wizard to create new tc Server instances to avoid problems that we observed in the past that often resulted in instances not being created correctly. Aside of that we also added the ability to take a look at the readme for the instance templates while choosing them within the wizard.



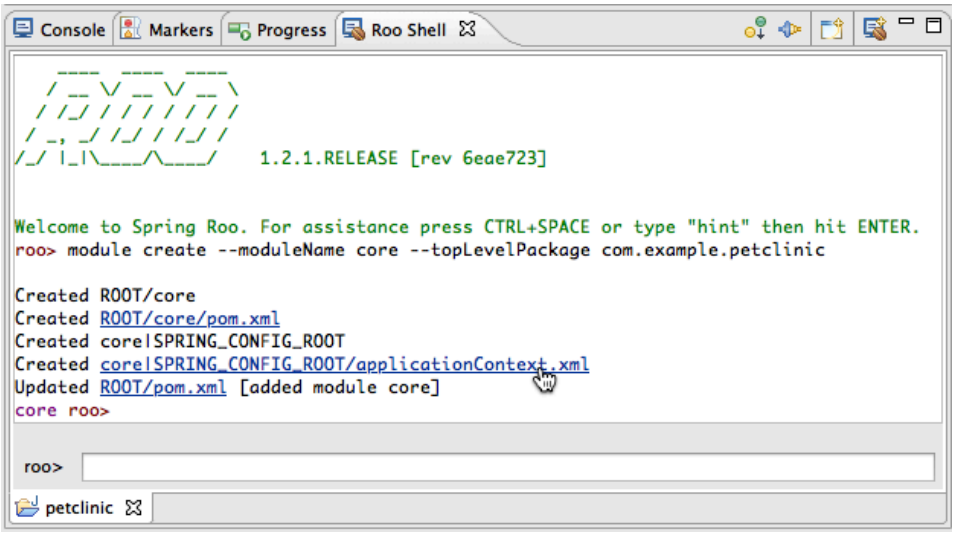
Support for Roo 1.2 packaging options

The "New Roo Project" Wizard adds support for specifying the project packaging. This is analogous to the "-packaging" option of Roo's "project" shell command, introduced in Roo 1.2



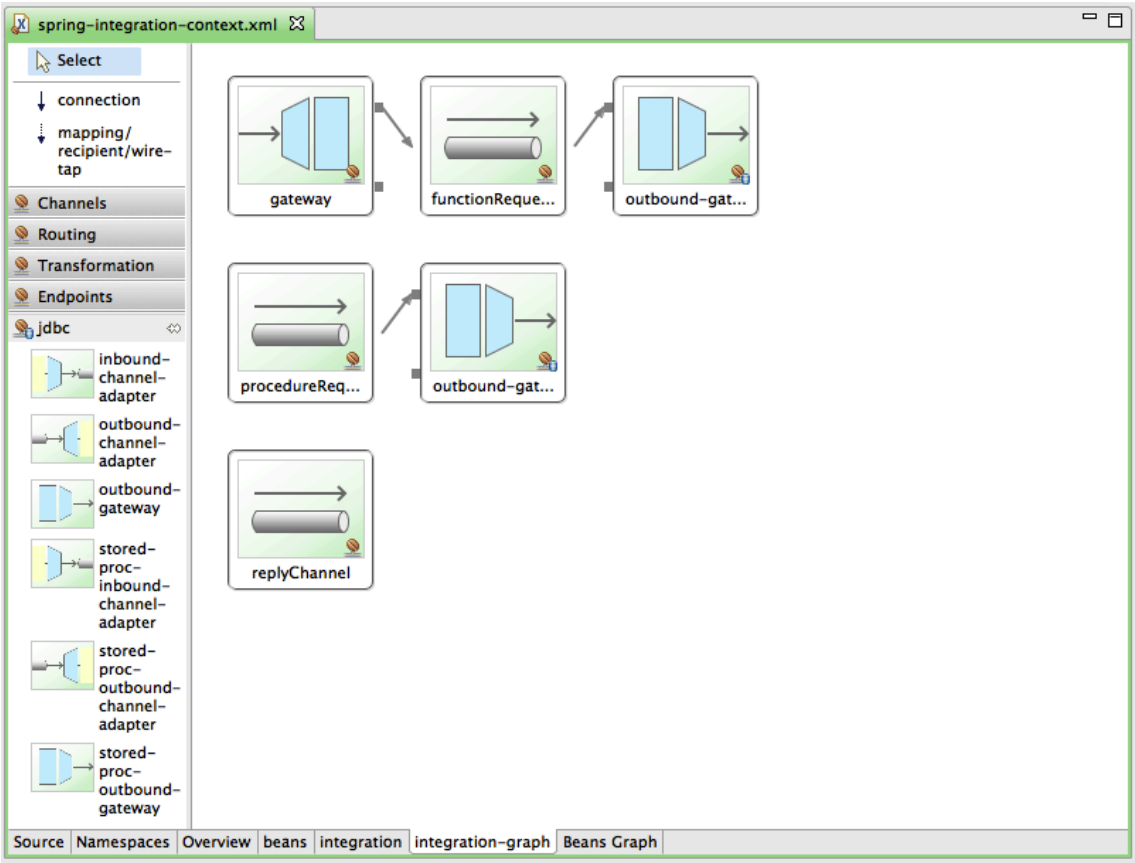
Support for multi module Roo projects

STS is now aware of multi module Roo projects and launches the the Roo shell for the parent project when importing a multi-module Roo project in STS. The Roo shell will create hyperlinks to resources within modules and open the resource in the Eclipse editor when clicked.



Support for Spring Integration 2.1

This releases adds support for Spring Integration 2.1.0. Visualizations have been added for the new SI-Gemfire, SI-Redis, and SI-RMI projects. All existing Spring Integration projects have been updated to support new visualization elements. Below you can see visual editing support for the new stored procedure adapters in SI-JDBC.



springsource
A division of vmware

Spring Integration 2.1 Project Templates

SpringSource Tool Suite 2.9.0 now ships with templating support for Spring Integration. Thus, when creating a new project using the Spring Template Project Wizard, you can now select between the following 3 new Spring Integration targeted templates:

- Spring Integration Project (Standalone) - Simple
- Spring Integration Project (Standalone) - File
- Spring Integration Project (War)

The "Simple" template creates a basic Spring Integration project, which runs as a standalone Java application, using only core Spring Integration components. In order to illustrate File polling capabilities, the "File" template uses additional components to poll file directories as well as to route those files. Lastly, the "War" template allows users to easily create basic Spring Integration projects that are targeted to run in servlet containers as part of a WAR deployment. For illustration purposes the "War" template uses the Spring Integration Twitter adapter.

Grails Development Tools

Extensions Page

The version of Grails available from the extensions page is now 2.0.1. Related to that the version of Groovy installed by default from the extensions page has been upgraded from 1.7 to 1.8.

Better type inferencing inside of Grails unit tests

Inside of unit tests for controllers, STS now supports the complete unit test DSL in content assist and type inferencing. For example below, you can see that the special controller properties like params, actionName, and request are available. Furthermore, all of the relevant properties and methods from ControllerTestMixin are available in content assist.

```
@TestFor(OtherController)
@Mock(Other)
class OtherTests {
  void testFoo() {
    controller.params
    controller.actionName
    controller.request

    co applicationContext : GrailsWebApplicationContext - GrailsUnitTestMixin (Controller
  }      config : ConfigObject - GrailsUnitTestMixin (Controller unit test DSL)
}        controller : OtherController - OtherTests (Controller unit test DSL)
         flash : FlashScope - ControllerUnitTestMixin (Controller unit test DSL)
         grailsApplication : GrailsApplication - GrailsUnitTestMixin (Controller unit test DSL)
         groovyPages : Map - ControllerUnitTestMixin (Controller unit test DSL)
         loadedCodecs : Set - GrailsUnitTestMixin (Controller unit test DSL)
         messageSource : MessageSource - GrailsUnitTestMixin (Controller unit test DSL)
         metaClass : MetaClass - GrailsUnitTestMixin (Controller unit test DSL)
         model : Map - ControllerUnitTestMixin (Controller unit test DSL)
         modelAndView : ModelAndView - OtherTests (Controller unit test DSL)
         params : GrailsParameterMap - ControllerUnitTestMixin (Controller unit test DSL)
         request : GrailsMockHttpServletRequest - ControllerUnitTestMixin (Controller unit te
         response : GrailsMockHttpServletResponse - ControllerUnitTestMixin (Controller uni
         servletContext : MockServletContext - ControllerUnitTestMixin (Controller unit test
         session : MockHttpSession - ControllerUnitTestMixin (Controller unit test DSL)
         validationErrorsMap : Map - GrailsUnitTestMixin (Controller unit test DSL)
         view : String - ControllerUnitTestMixin (Controller unit test DSL)
         views : Map - ControllerUnitTestMixin (Controller unit test DSL)
```

DSL Support

Named queries

STS now has full support for the named query DSL:

http://grails.org/doc/latest/ref/Domain%20Classes/namedQueries.html.

First, for defining named queries, STS provides full content assist and type inferencing inside of the namedQuery static field of a domain class. For example, hovering over a reference to "gt" will bring up the JavaDoc for the "gt" method of HibernateCriteriaBuilder (and pressing F3 will navigate to the definition of "gt"):

```
static namedQueries = {
    recentPublications {
        def now = new Date()
        gt 'datePublished', now - 365
    }
    oldP
    }
    publ
```

Provided by Criteria builder DSL

Criteria grails.orm.HibernateCriteriaBuilder.gt(String propertyName, Object propertyValue)

Creates a "greater than" Criterion based on the specified property name and value

And it is possible to reference other named queries inside the definition of one:

```
publicationsWithBookInTitle {
    like 'title', '%Book%'
}
recentPublicationsWithBookInTitle {
    // calls to other named queries…
    recentPublications()
    publicationsWithBookInTitle()
}
```

Second, STS also provides full inferencing and content assist support for using named queries. Here are some examples that are taken from the Grails user guide, see:

http://grails.org/doc/latest/ref/Domain%20Classes/namedQueries.html).

All of these examples are fully supported by content assist and type inferencing:

```
// get all recent publications…
def recentPubs = Publication.recentPublications.list()
// get up to 10 recent publications, skip the first 5…
recentPubs = Publication.recentPublications.list(max: 10, offset: 5)

// get the number of recent publications…
def numberOfRecentPubs = Publication.recentPublications.count()

// get a recent publication with a specific id…
def pub = Publication.recentPublications.get(42)

// get all recent publications where title = 'Some Title'
def pubs = Publication.recentPublications.findAllWhere(title: 'Some Title')

// get a recent publication where title = 'Some Title'
pub = Publication.recentPublications.findWhere(title: 'Some Title')

// dynamic finders are supported
pubs = Publication.recentPublications.findAllByTitle('Some Title')

// get all old publications with more than 350 pages
pubs = Publication.oldPublicationsLargerThan(350).list()

// get all old publications with more than 350 pages and the word 'Grails' in the title
pubs = Publication.oldPublicationsLargerThan(350).findAllByTitleLike('%Grails%')

// get all recent publications with 'Book' in their title
pubs = Publication.recentPublicationsWithBookInTitle().list()
```

Note that due to Grails bug http://jira.grails.org/browse/GRAILS-8387, the return values of named query methods like "list", "get", and "findWhere" are all dynamically typed and so do not provide any useful type inferencing in the editor.

Case insensitive dynamic finder content assist

Dynamic finders can now be invoked in content assist in a case insensitive way:

```
class Person {
    String name
    Date dob

    static printYoungPeopleNamedAl() {
        def younguns = Person.findallbydobgr
    }
}
```

findAllByDobGreaterThan : List – Person (GORM)
findAllByDobGreaterThanEquals : List – Person (GORM)
findAllByDobGreaterThan(Object dob) : List – Person (GORM)
findAllByDobGreaterThanEquals(Object dob) : List – Person (GORM)

Grails aware search

Searching references to controller types and their action fields or methods is now 'grails aware' (meaning it understands and finds references if made via certain Grails specific idioms. Below are some examples of the recognized idioms:

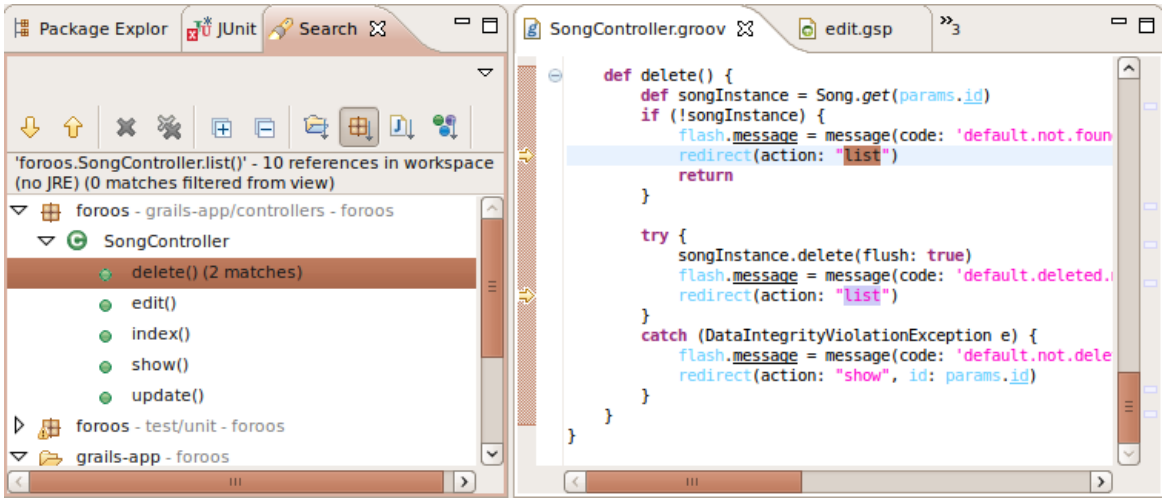
Inside controllers:

render(controller: "song", action: "edit")

This counts as a reference to SongController and SongController.edit

redirect(view: "song")

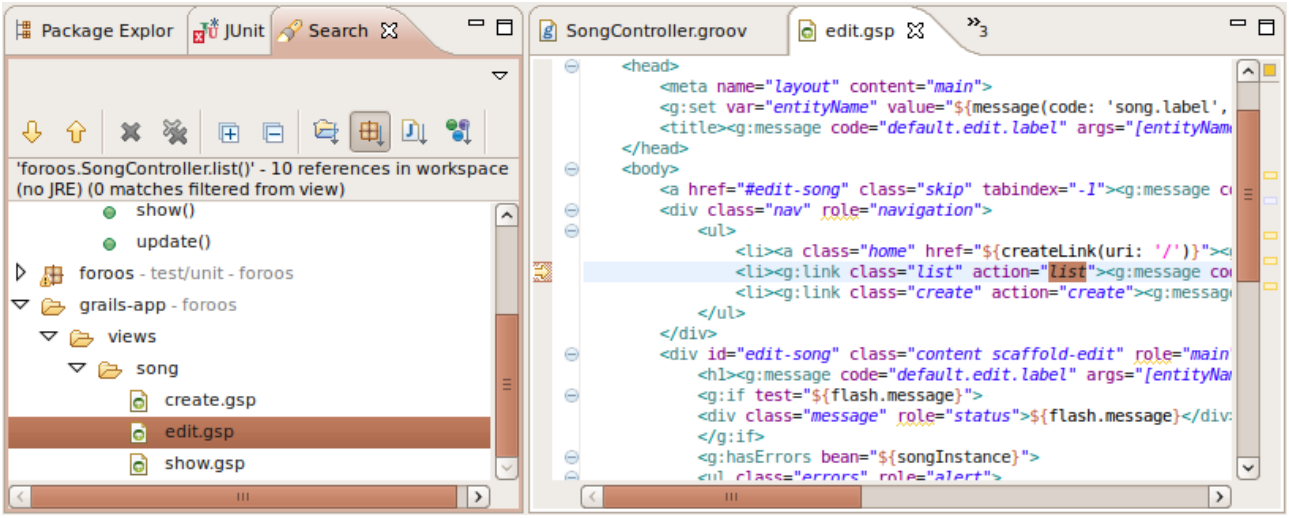
This counts as a reference to SongController



Inside gsp files

<g:link controller='song' action='list'>

This counts as references to SongController and SongController.list



Inside URL mappings:

"/product"(controller:"product", action:"list")

This counts as a reference to **ProductController** and **ProductController.list**

"/showPeople" {

controller = 'person' // This counts as a reference to **PersonController**

action = 'list' // This counts as a reference to **PersonController.list**

}

name personList: "/showPeople" {

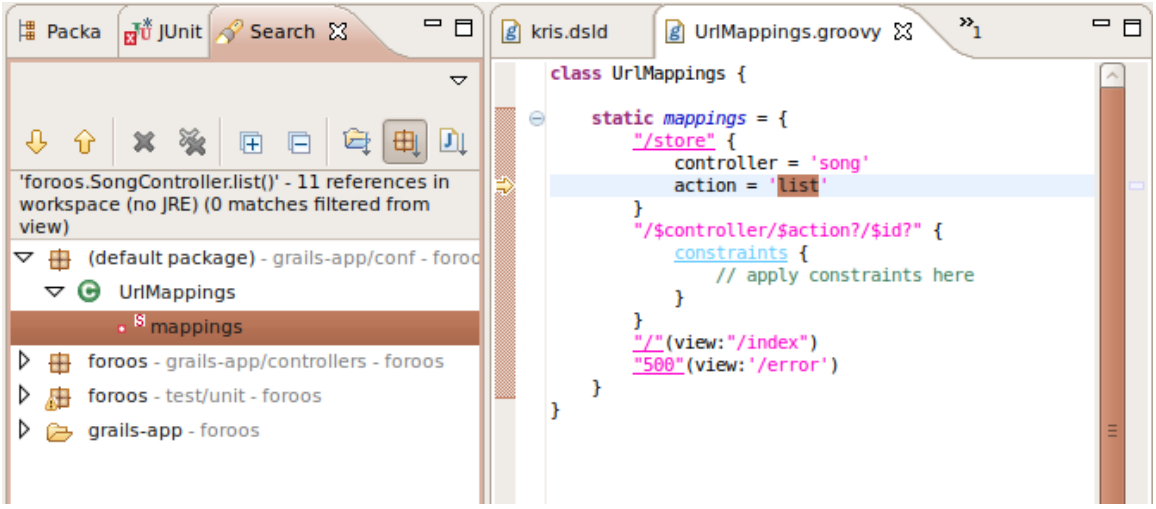
controller = 'person' // counts as a reference to **PersonController**

}

"/product/$id"(controller:"product") { // counts a reference to **ProductController**

action = [GET:"show", PUT:"update", DELETE:"delete", POST:"save"] //Counts as references to **show**, **update**, **delete** and **save** in **ProductController**

}



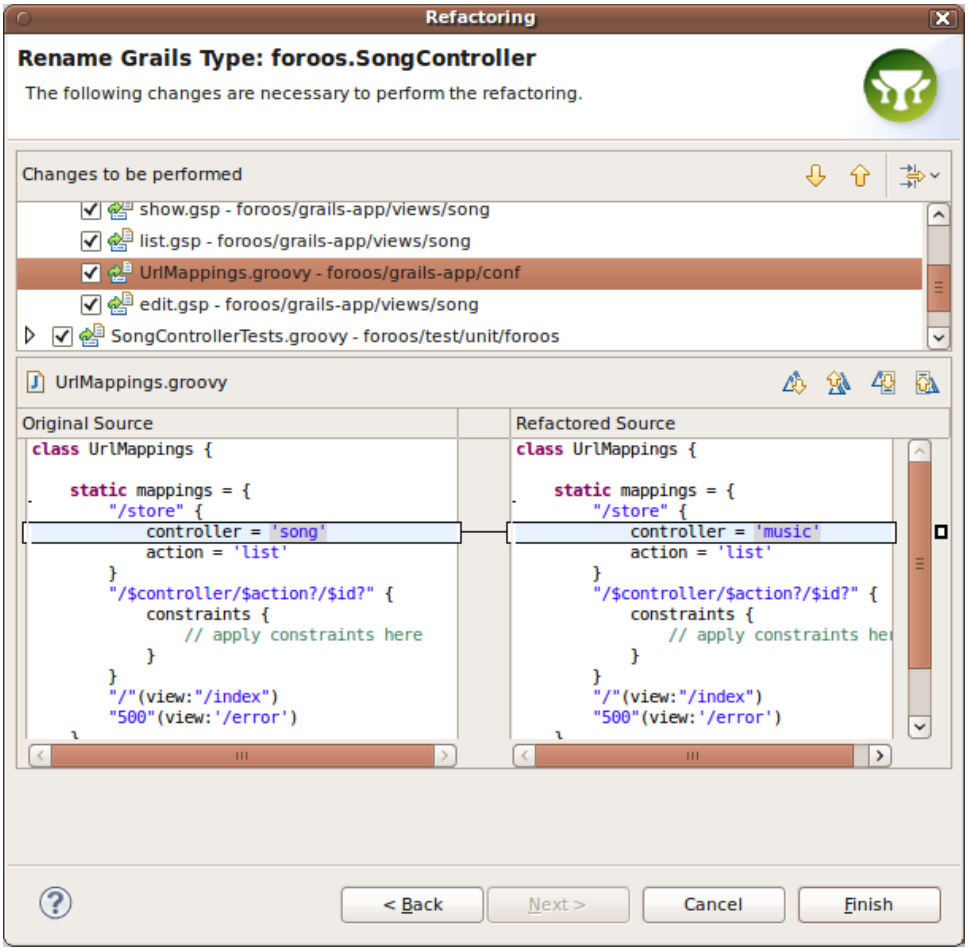
Grails aware refactoring

The same idioms as described in the previous section are also recognized and replaced appropriately when performing rename refactorings, renaming either:

- a controller class
- an action method or fields in a controller class.
- a gsp file (because it triggers a correspondding action rename).

In contrast with the searching support, which is new in STS 2.9.0, the refactoring functionality existed already in STS 2.8.0. However, the set of recognized idioms has been expanded. The following idioms are newly recognized:

- redirect | render idioms are now recognized for controller type renames (previously they were only recognized for action renames)
- references inside URLMappings (see examples above)



Grails 2.0 support

Where queries

Grails 2.0 has introduced the notion of where queries.

See http://grails.org/doc/2.0.0.RC2/guide/GORM.html#whereQueries.

STS provides editing support for this mini-DSL. For example, you can define where queries and build one query on top of another:

```
def bieberBooks = Publication.where {
    title ==~ "Bieber"
}
def upcomingBieberBooks = bieberBooks.where {
    datePublished > new Date()
}
```

Hovering and navigation of fields work as expected.

JavaDoc and source code for Gorm methods in domain classes

Source code is now included with the Grails distribution for the gorm-datastore jars. This means that Javadoc for gorm methods like "attach" and "validate" are available for hovers:

```
Publication pub = Publication.recentPublications.get(42)
pub.attach()
```

Provided by GORM Instance API

Publication org.grails.datastore.gorm.GormInstanceApi.attach()

Attaches an instance to an existing session. Requries a session-based model

DSL support in Grails unit and integration tests

STS now has much improved support for the Grails unit and integration test mini-DSL. For example, the @TestFor and @Mock annotations are used to populate implicit fields inside of your test class:

```
@TestFor(PublicationController)
@Mock(Publication)
class PublicationControllerTests {

    void testIndex() {
        controller.index()  // inferred type of PublicationController
    }
```

PublicationController PublicationControllerTests.controller
The controller class under test

Controller action return values are available where appropriate:

```
void testList() {
    controller.list()
    assert model.publicationInstanceList.size() == 0
    assert model.publicationInstanceTotal == 0
}
```

And the various mixin classes in unit tests are recognized in the editor, as described here:

http://grails.org/doc/2.0.0/guide/testing.html#unitTesting

Here you can see some of the ControllerUnitTestMixin fields and methods being referenced:

```
void testList() {
    // ControllerUnitTestMixin
    view
    session
    response
    params
```

GrailsParameterMap ControllerUnitTestMixin.params
Provided by Controller unit test DSL

GrailsParameterMap grails.test.mixin.web.ControllerUnitTestMixin.getParams()

The Grails 'params' object which is an instance of GrailsParameterMap

As expected, pressing F3 will navigate to the definition of any of these fields in the appropriate mixin class.

See STS-2222, STS-2225, and STS-2235 for more information. Also, please note that bug STS-2319 is still open. If your controller action contains a redirect, STS will not be able to infer the return values of the action.

Groovy-Eclipse

STS 2.9.0 provides Groovy-Eclipse 2.6.1 from the extensions page. This release includes a number of enhancements, described below:

Split assignment/declaration quickfix

The split assignment and split declaration quickfixes are available on any assignment or declaration expression and works like this:

```
def name = getName()
```

Remove unnecessary semicolons
Split declaration

becomes this:

```
def name
name = getName()
```

Swap operands quickfix

The swap operands quickfix reorders the left and right sides of binary expressions. This code:

```
if (shouldExecute && isStale) {
    refresh()
}
```

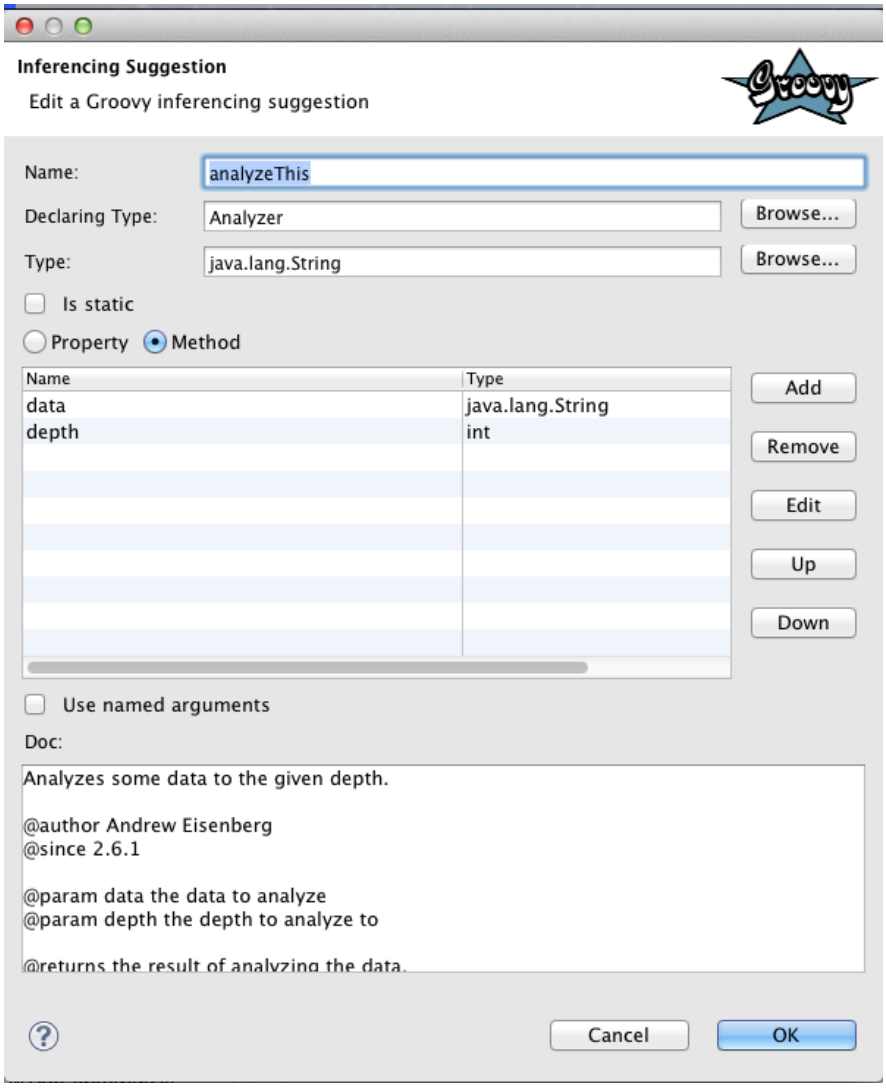
Remove unnecessary semicolons
Swap infix operands.

becomes this:

```
if (isStale && shouldExecute) {
    refresh()
}
```

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:



Which will then get 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, it is possible to use Javadoc tags inside of DSLD doc tags. For example, this gives a similar effect to 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
"""
```

Note: 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:

```
class Analyzer {
    /**
     * The methodology to use for analyzing
     */
    String methodology
}

new Analyzer(methodology: "Fouriet transform");
```

String Analyzer.methodology

The methodology to use for analyzing

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:

```
abstract class Analyzer {
    void analyzeThis(String name, URL url) {
    }
    void analyzeThat(String name, URL url) {
    }
}
class ConcreteAnalyzer extends Analyzer {
    an
}   analyzeThat(String name, URL url) : void - Override method in 'Analyzer'
    analyzeThis(String name, URL url) : void - Override method in 'Analyzer'
```

becomes this:

```
class ConcreteAnalyzer extends Analyzer {
    public void analyzeThat(String name, URL url) {
        // TODO Auto-generated method stub
        super.analyzeThat(name, url);
    }
}
```

Code select and inferencing for static imports

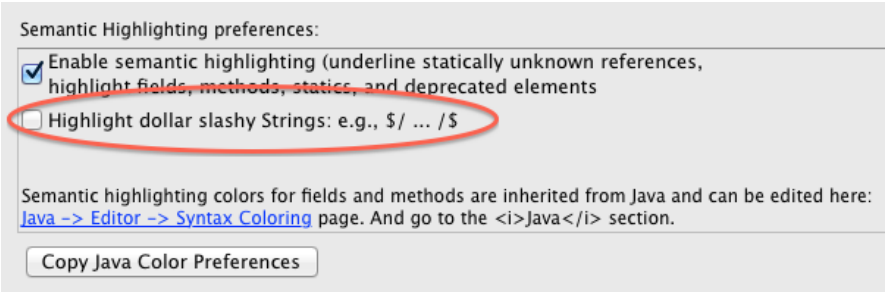
We have done significant work around supporting static imports. References to static imports now allow navigation, static imports are now appropriately renamed during refactoring, and they are found during search.

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:

```
def PATTERN1 = ~(/$/)
seeing.some.rather.funny.syntax.highlighting
def PATTERN2 = ~(/$/)
```

It is now possible to disable slashy strings by going to the Preferences > Groovy > Editor preferences page:



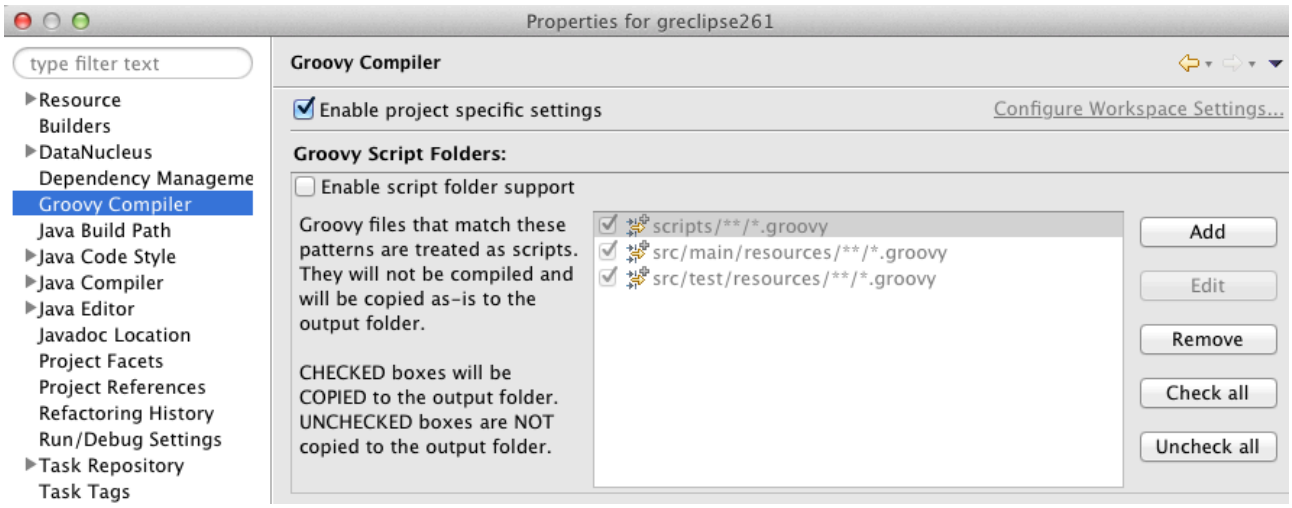
And the file is now highlighted correctly (the underlines are expected):

```
def PATTERN1 = ~(/$/)
seeing.some.rather.funny.syntax.highlighting
def PATTERN2 = ~(/$/)
```

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 selecting "Enable project specific settings", the script folder settings for an individual project override 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.

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:

```
class Tree {
    Tree[] children
    def val
    Tree plus(Tree other) { /*...merge */ }
}
def p = new Tree(val:9)
def q = new Tree(val:10)
def r = p + q
print r.val
```

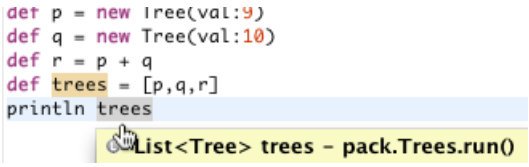
Object pack.Tree.val

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'

}
```

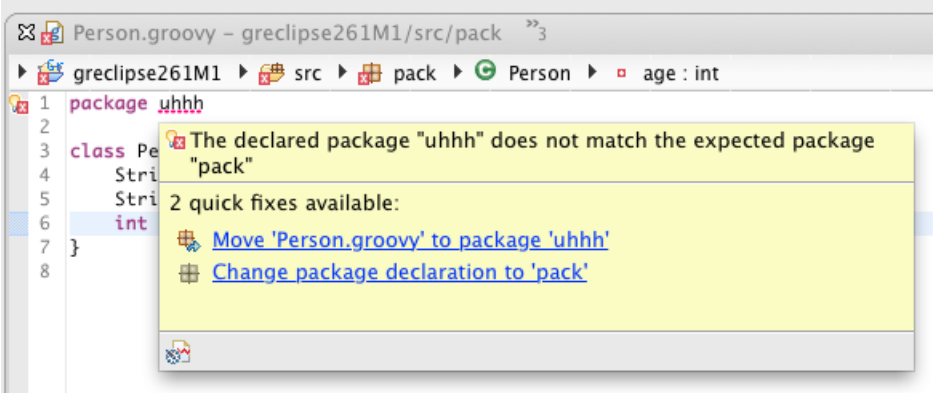
Better inferencing of list and map literals

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:



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

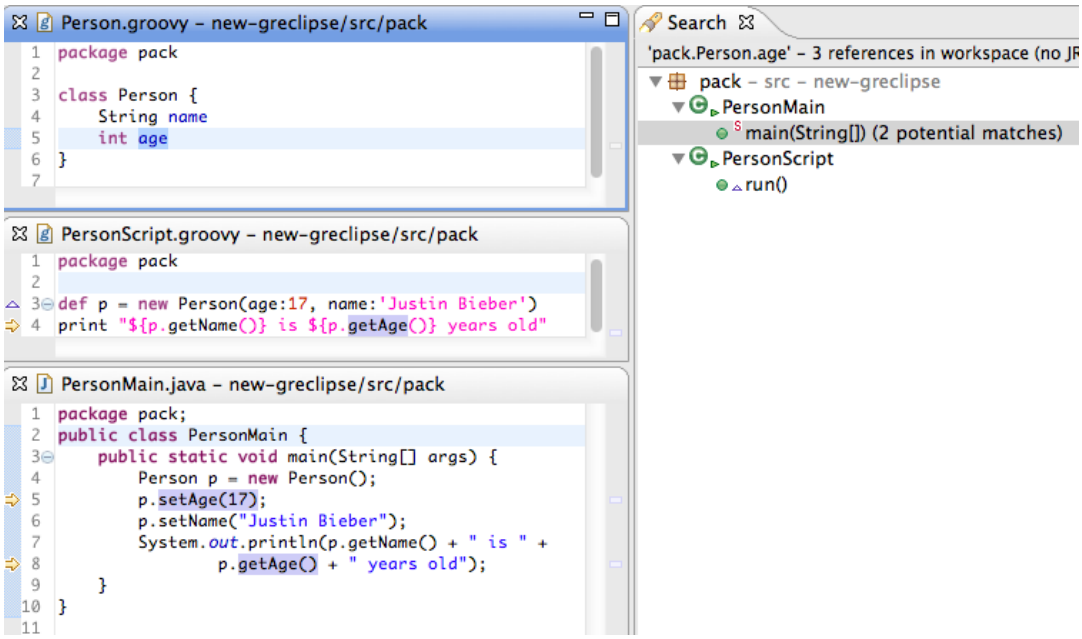
The **Convert to closure** and **Convert to method** quick assists are now available from the context menu under the Groovy Refactor section:

Also, the keybindings are Alt-G F and ALT-G M respectively. (patch from Geoff Denning)

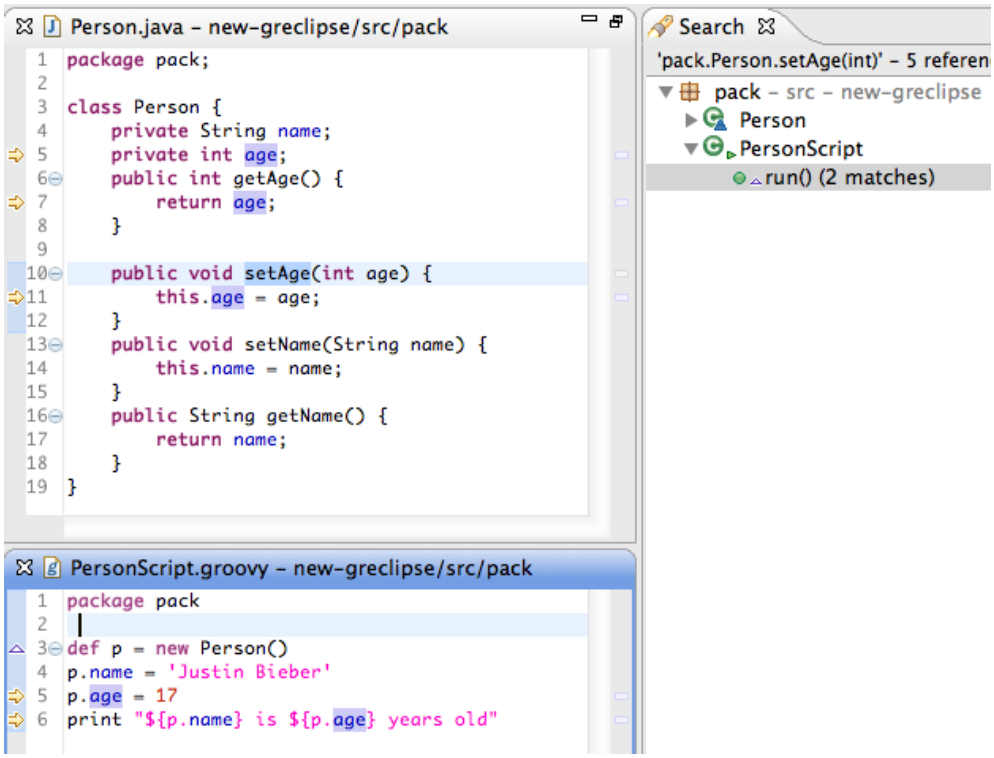
Search and refactoring

Search and refactoring of generated getters, setters, and properties

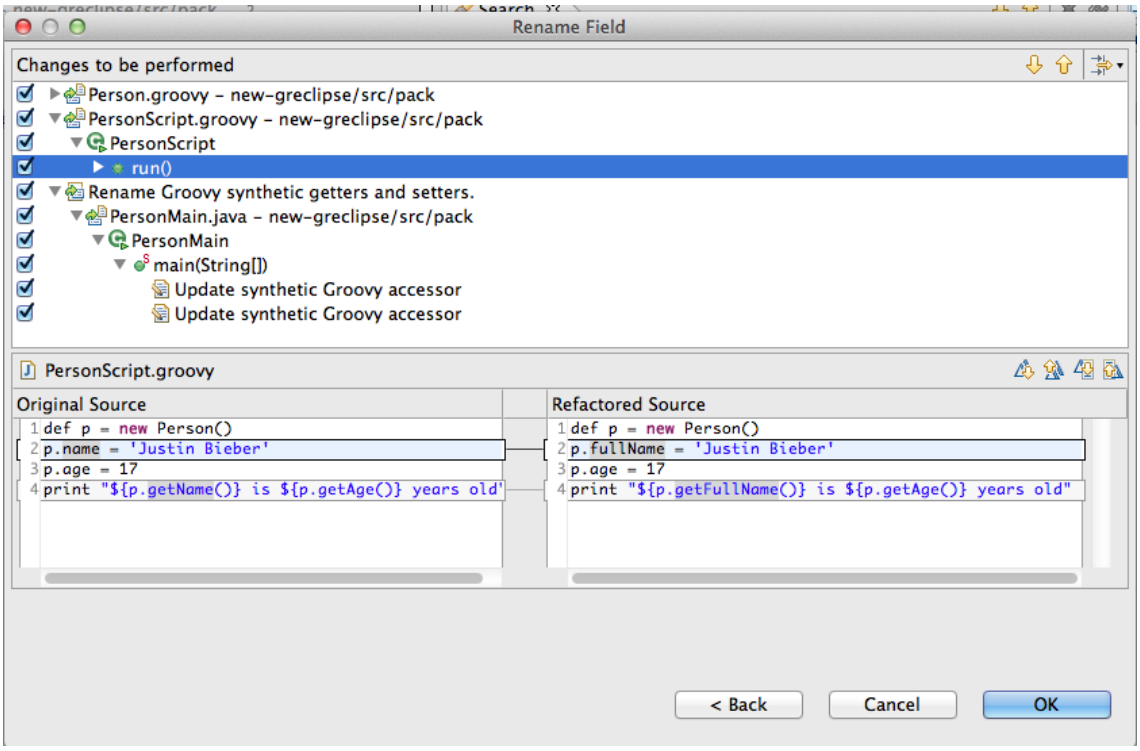
Groovy-Eclipse now allows you to search for references to generated getters, setters and properties. For example, searching for references to a Groovy property (such as age in this example) will find all references to getAge and setAge in both Java and Groovy files:



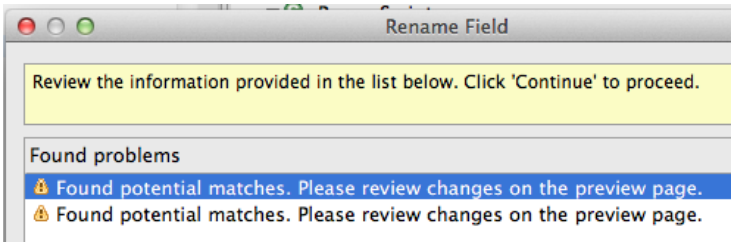
Similarly, references Java to getters and setters can be found inside of Groovy files even when they are referenced as properties. In this example, the class Person is defined in Java with explicit getters and setters. Searching for references on setAge will return references to the generated age property in the Groovy script:



This also works for refeactoring. 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:

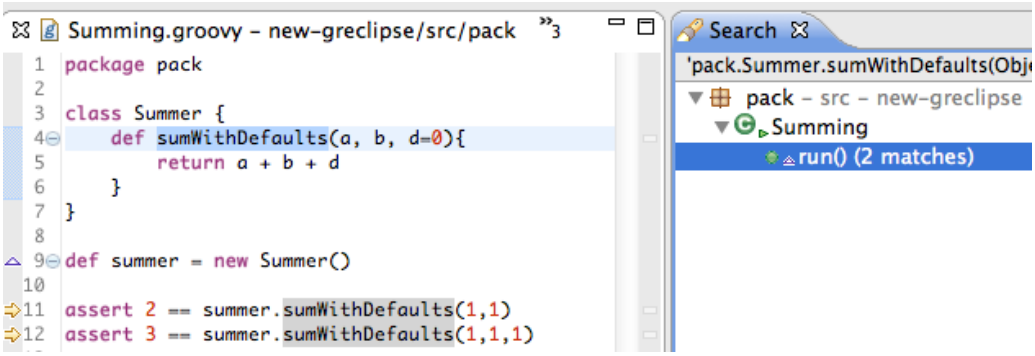


This warning comes about since some of the synthetic references in Java files cannot be determined to be precise by the Java search engine. By the nature of the language, refactoring of Groovy code can never be as precise as Java code is. It is always recommended to view the preview page before executing a refactoring.

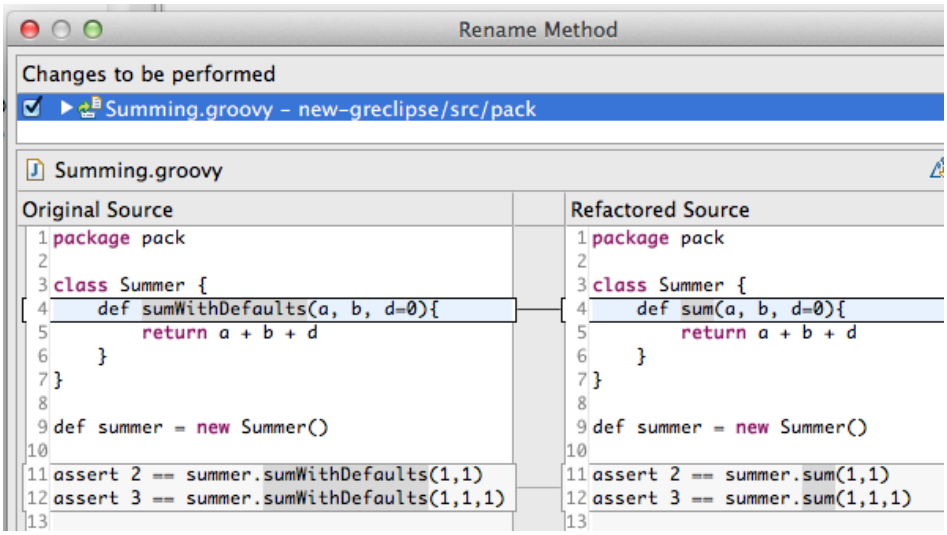
For more information on this feature, see issues GRECLIPSE-1204, GRECLIPSE-1010, and GRECLIPSE-1205.

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:



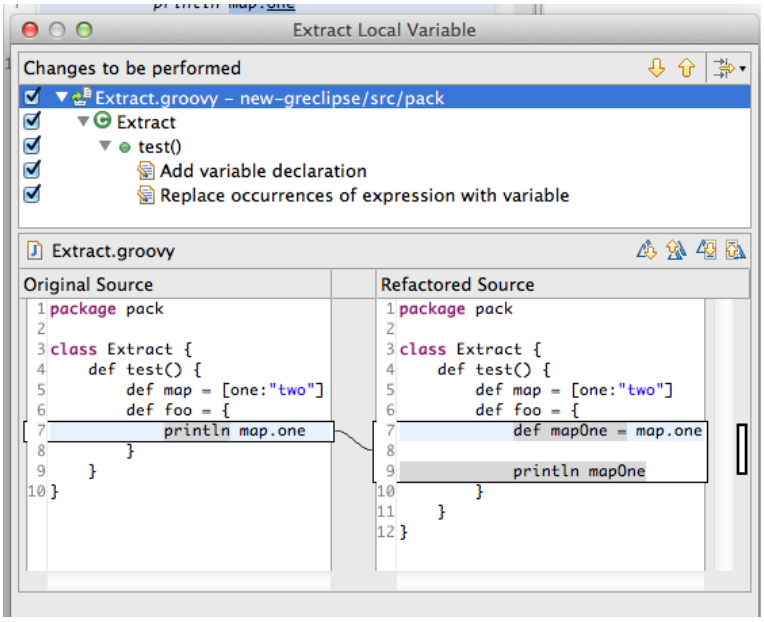
Similarly, rename refactor will correctly rename all references no matter how many parameters are used:



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 preceeding the variables first use. See this example, where 'map.one' is extracted to a variable and placed inside the enclosing closure:



Parser recovery

Further recovery enhancements have been made to the Groovy Parser. This enables it to cope better with malformed (unfinished) code and that enables content assist to offer suggestions in more places than before. Here are a couple of examples of the latest improvements:

These two situations show that correct content assist options are available even though there is a missing close paren:

```
import java.awt.BorderLayout

new HashMap().put('key',BorderLayout.
```

AFTER_LAST_LINE : String - Bor
AFTER_LINE_ENDS : String - Bo
BEFORE_FIRST_LINE : String - B
BEFORE_LINE_BEGINS : String -
CENTER : String - BorderLayout

```
import java.awt.BorderLayout

String message = ('foo' + BorderLayout.
```

AFTER_LAST_LINE : String - Bor
AFTER_LINE_ENDS : String - Bor
BEFORE_FIRST_LINE : String - B
BEFORE_LINE_BEGINS : String -
CENTER : String - BorderLayout

It is now possible to work on the **if** condition without the **then** block {...} being defined yet:

```
File f = new File("abc")

if (f.i)
```

isAbsolute() : boolean - File (Groovy)
isDirectory() : boolean - File (Groovy)
isFile() : boolean - File (Groovy)
isHidden() : boolean - File (Groovy)

It will even work if the trailing paren of the if condition isn't specified yet:

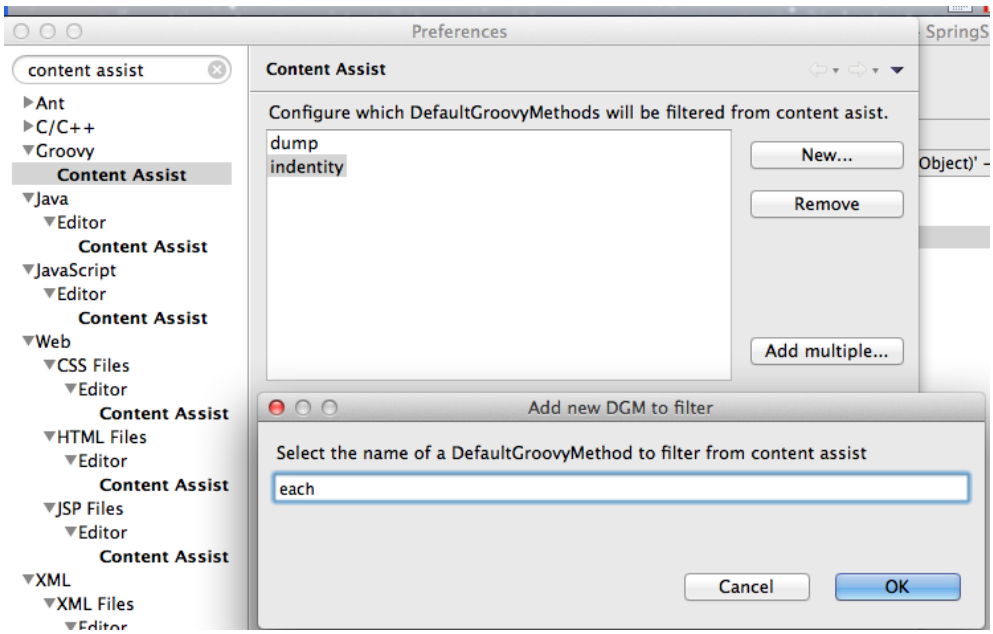
```
File f = new File("abc")

if (f.is
```

isAbsolute() : boolean - File (Groovy)
isDirectory() : boolean - File (Groovy)
isFile() : boolean - File (Groovy)
isHidden() : boolean - File (Groovy)

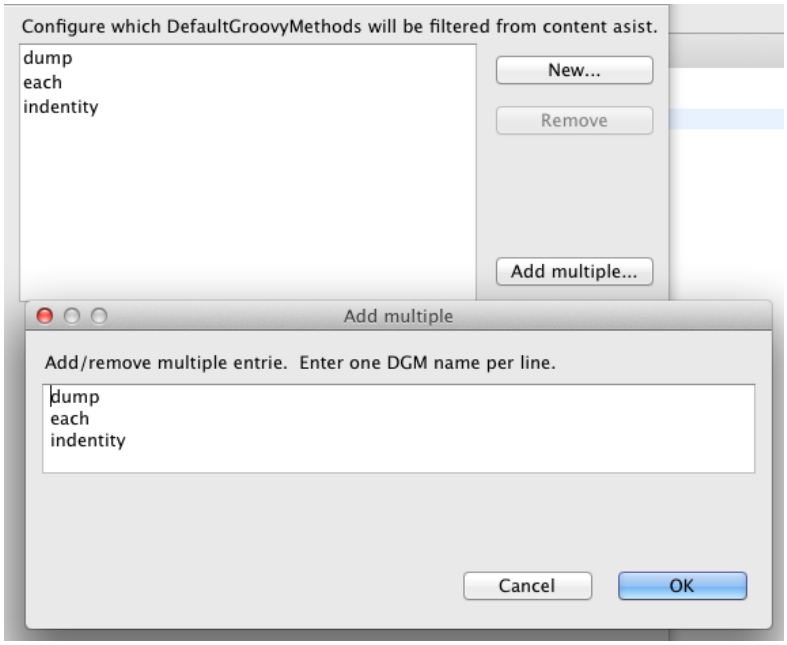
Content assist

Suppressing DGMs (default groovy methods) from content assist

It is now possible to selectively suppress DGMs from cluttering up 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:

```
3  1.each
     eachWithIndex(Closure closure) : Object –
```

Support for named arguments in constructors

When a Groovy class has no explicit constructor, it is possible to build an instance of the class using named arguments as described here (http://groovy.codehaus.org/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 arguments:

```
4  class Customer {
5      Integer id
6      String name
7      Date dob
8
9      static void main(args) {
10         def customer = new Customer(dob: new Date(), )
11         println("Hello ${customer.name}")
12     }
13 }
```

```
Customer() – Customer (Groovy)
id : __ – int : named parameter : (null)
name : __ – java.lang.String : named parameter : (null)
```

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 }
```

```
myName
hisName
name
```

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/**/)*

(where /**/ is the location where content assist is invoked)

For more information, see issue 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-1114):

```
4 def foo() { }
5 (1..10).each {
6     foo
7 }        foo() : Object - Extract (Groovy)
```

Similarly, the relevant fields like "closure" and "owner" are now available in content assist when inside a closure (GRECLIPSE-1267):

```
(1..10).each {
    del
}    delegate :
```

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     Add unimplemented methods        1 method to implement:
11 }   Make type 'Manager' abstract        - pack.Person.findColleagues()
       Rename in file (⌘2 R)
       Rename in workspace
       Remove unnecessary semicolons
```

1. Make class abstract, which adds the "abstract" modifier to the sub-class
2. 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
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   Remove unnecessary semicolons
5
```

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         /* ... */   Rename in file (⌘2 R)          Convert method declaration to closure
10    }                Rename in workspace
11 }                   Convert method declaration to closure
```

is converted into this closure:

```
8     def findColleaguesByName = { String name ->
9         /* ... */   Convert closure declaration to method   Convert closure declaration to method
10    }                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 }
```

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 }
```

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 2.6.1 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 thought 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.

Gradle Tooling

Editing Support

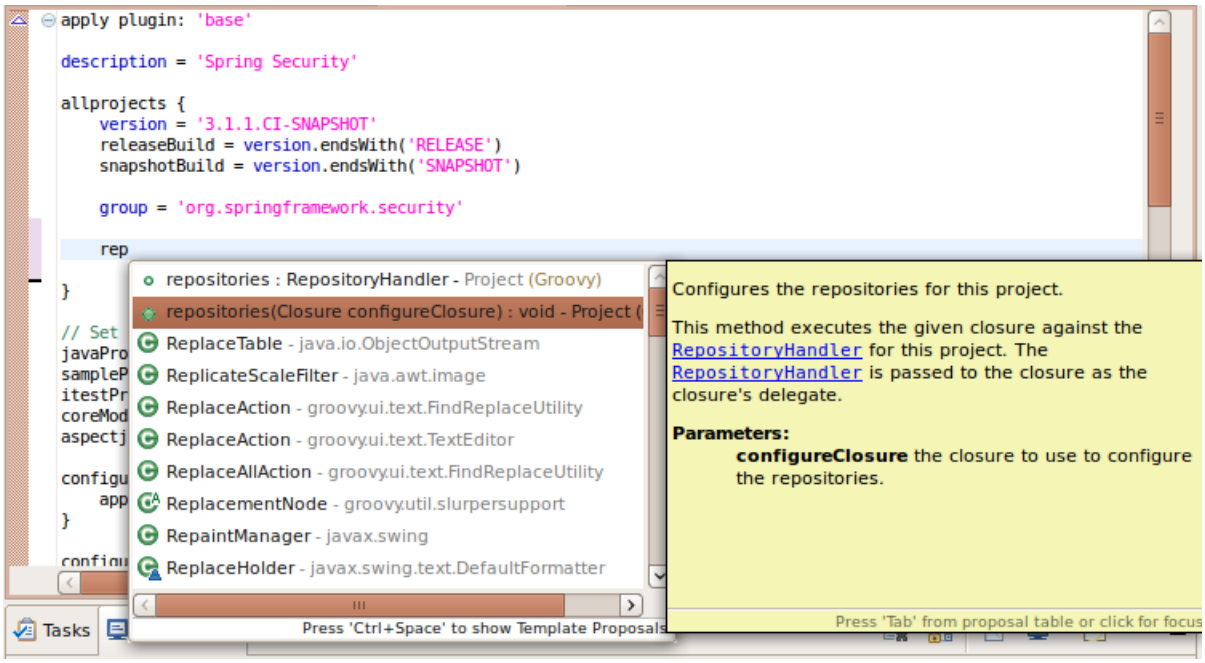
We now provide some basic editing support for .gradle files. To benefit from this a recent version of Greclipse must be installed (version 2.6.1.M1 is required).

Support consists of two separate pieces each of which can be enabled/disabled individually.

1) Groovy Eclipse DSL Descriptor support:

STS Gradle tool support now ships with a simple Groovy Eclipse DSL Descriptor.

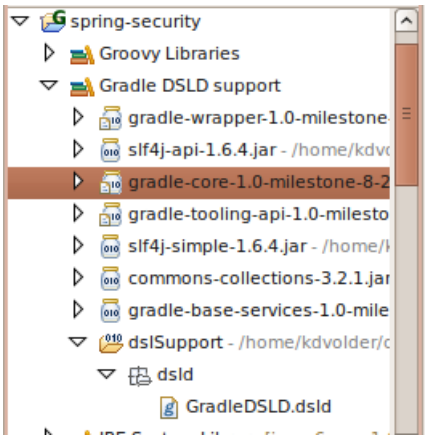
Although the DSLD file is still limited and very much a work in progress it will already provide some useful content-assist and JavaDoc hovers.



Sample JavaDoc hover:

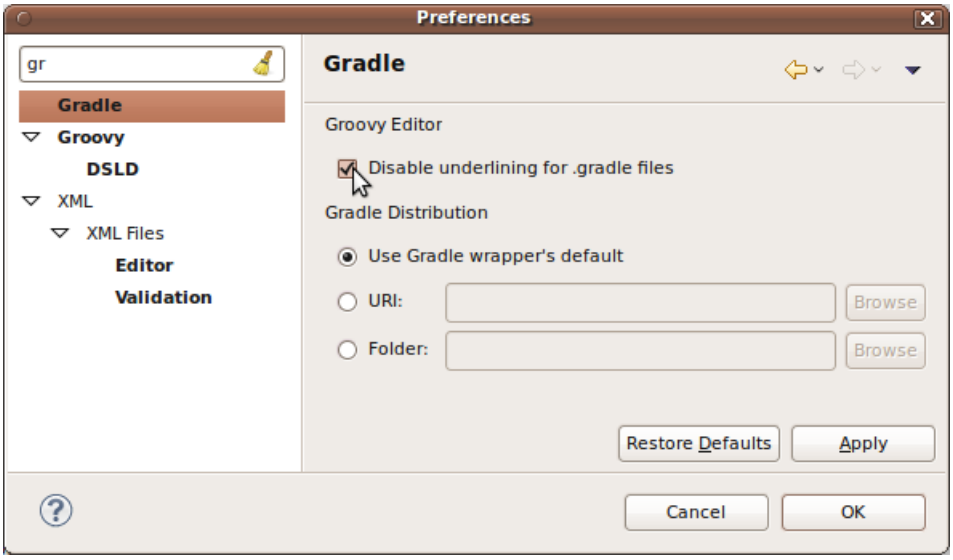
DSLD support is enabled automatically when importing a Gradle project with the import wizard. It can also be disabled/enabled after the import with the Gradle > Enable/Disable DSL Support menu on an already imported project.

Enabling DSLD support will convert the project into a 'Groovy Project' and add the required classpath entries.



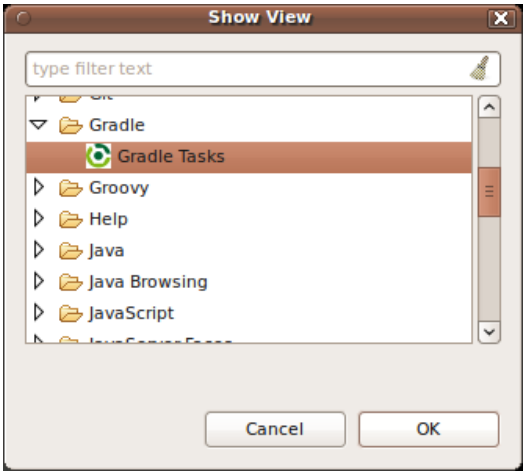
2) An option to suppress all underlining in .gradle files.

By default, the Groovy Eclipse editor underlines all identifiers for which it cannot infer a type. This can be disturbing when editing a Gradle script file where many of the identifiers can't be inferred. STS now provides the option to disable this underlining in .gradle files:



Gradle tasks view

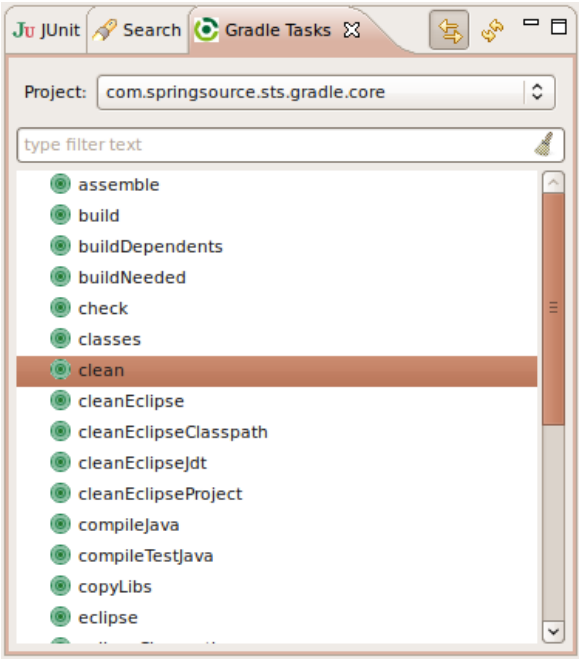
Gradle tooling now provides a basic 'Tasks View' that can be opened via **'Windows > Show View > Gradle > Gradle Tasks'**:



The tasks view shows a list of tasks associated with a particular Gradle project (see image below). The project can be selected manually using the popup menu in the view itself.

Alternatively a 'link with selection' option ('double arrow' toggle button in the tool bar) will make the view automatically track the 'current project' based on elements selected in other views (e.g. project explorer or outline view).

Double clicking on any task in the view will launch the task. If a launch configuration for this task does not exist a new launch configuration will be created, otherwise the existing configuration will be reused.



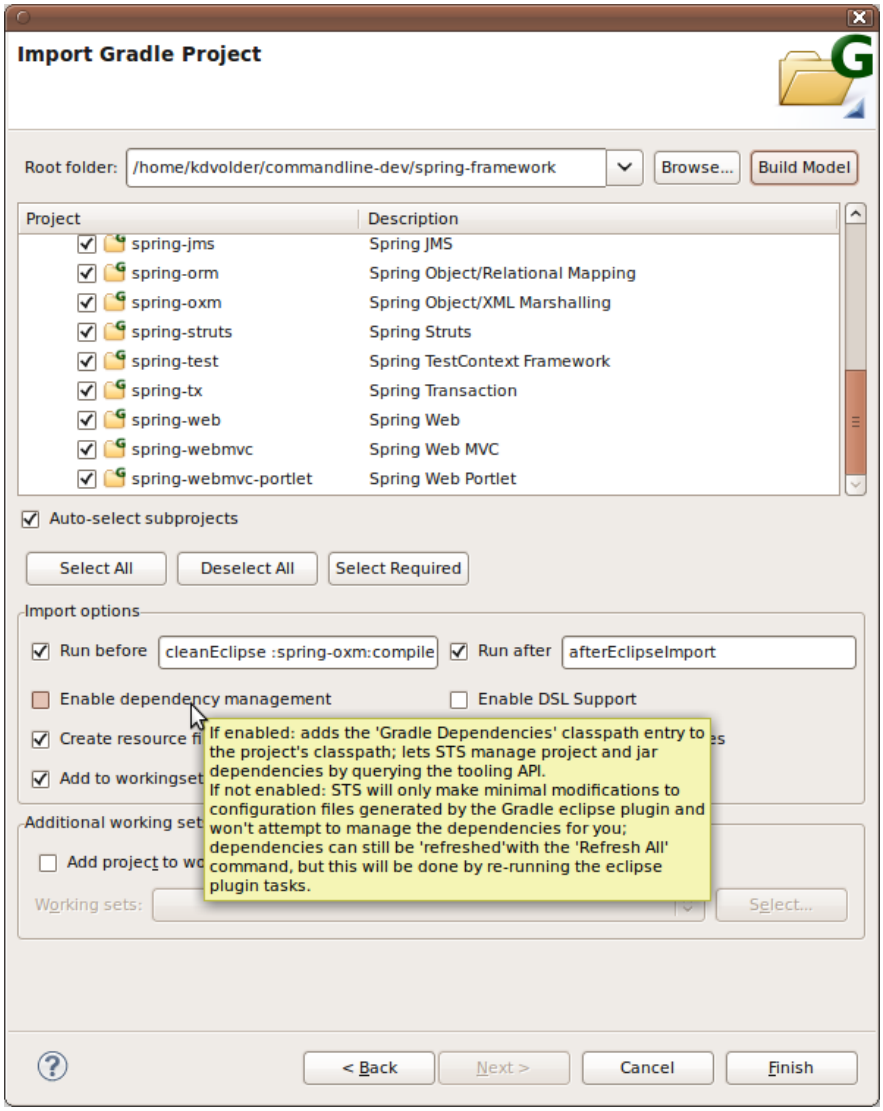
Currently the view is very basic and always shows an unfiltered list of all the tasks in the selected project, sorted alphabetically. In the future we plan to provide ways to customize sorting and filtering the list. We are still considering options on how to further develop this part of the UI and welcome any feedback.

Working without the 'Gradle Dependencies' classpath container

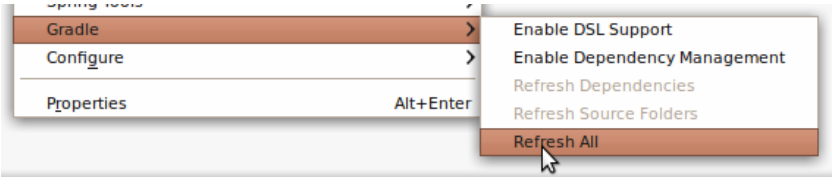
STS 2.9.0 now supports importing and working with Gradle projects without adding the 'Gradle Dependencies' classpath container. This option is provided for users who already have working and carefully tuned build scripts based on the Gradle eclipse plugin.

It is also useful if a project depends on features that are currently supported by the Gradle eclipse plugin, but not via the Gradle Tooling API (essentially, only "pure Java Nature" projects are currently supported by the tooling API).

The import option to achieve this is shown here:



When dependency management is disabled, it is no longer possible to refresh individual aspects of a project's configuration. Thus the **Refresh Dependencies** and **Refresh Source Folders** menus are disabled. However it is still possible to refresh a project as a whole, using the **Refresh All** menu command:



Fixed Bugs and Enhancement Requests

Here is a full list of resolved bugs and enhancement requests for the 2.9.0 release:

https://issuetracker.springsource.com/secure/IssueNavigator.jspa?reset=true&jqlQuery=project+%3D+STS+AND+fixVersion+in+%2811804%2C+11803%2C+11802%2C+11800%2C+11799%29+AND+status+in+%28Resolved%2C+Closed%29

New & Noteworthy of previous releases

STS 2.8.x:

http://download.springsource.com/release/STS/doc/STS-new_and_noteworthy-2.8.1.RELEASE.pdf

STS 2.7.x:

http://download.springsource.com/release/STS/doc/STS-new_and_noteworthy-2.7.2.RELEASE.pdf

STS 2.6.x:

http://download.springsource.com/release/STS/doc/STS-new_and_noteworthy-2.6.1.SR1.pdf

STS 2.5.x and before:

http://download.springsource.com/release/STS/doc/STS-new_and_noteworthy-2.5.2.SR1.pdf