# SpringSource Tool Suite 2.8.0

## - New and Noteworthy -

| | | | |
|---|---|---|---|
| Martin Lippert | 2.8.0.RELEASE | October 18, 2011 | Updated for 2.8.0.RELEASE |

www.springsource.com

# ENHANCEMENTS – 2.8.0

## General Updates

### Eclipse Indigo SR1, including support for Java 7.

STS updated the base Eclipse installation to Eclipse Indigo SR1 that comes with full Java IDE support for Java 7.

Take a look at the details of the Java 7 support:

http://www.eclipse.org/jdt/ui/r3_8/Java7news/whats-new-java-7.html

### Mylyn 3.6.2

The Mylyn version within STS is upgraded to the latest Mylyn release 3.6.2

### m2e 1.0.100

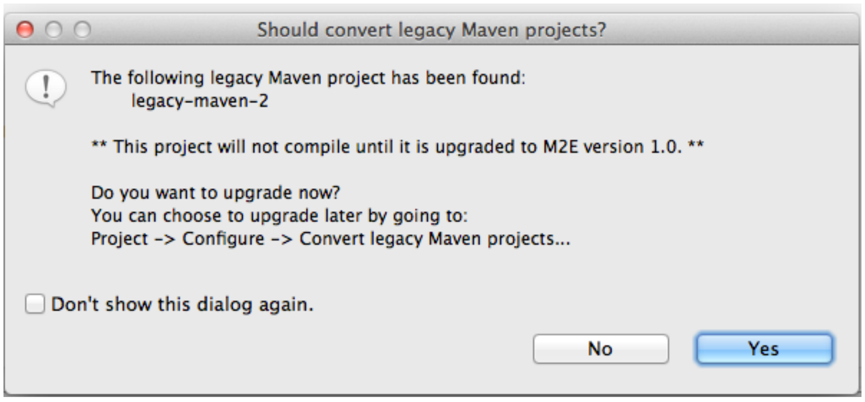
STS now ships with the latest m2e version 1.0.100 that comes as part of Eclipse Indigo SR1. The pre-build STS packages also contain compatible add-ons for WTP, AJDT and other useful add-ons.

Since existing Maven projects that were created using previous versions of STS don't work anymore with the new m2e version, we provide an automatic update for those projects.

STS now detects when you have legacy Maven projects in your workspace and gives you an opportunity to migrate them to a version compatible with STS. Some background: the m2eclipse project released a new version in June, along with the Eclipse Indigo release train. The project was rebranded as m2e and is now hosted on Eclipse.org. Unfortunately, the project broke backwards compatibility with this move. At this point, a legacy m2eclipse project in a workspace that has m2e installed will not compile and will not have any maven functionality turned on for it. Simply right-clicking and converting to a Maven project will not work since there will still be some legacy data in the classpath and project configuration that will cause problems.

Inside of STS, we are providing an easy way of helping users to migrate to the new version of m2e:

1. On startup, STS will search for any legacy Maven projects in the workspace. If any are found, the following dialog will appear:



Should convert legacy Maven projects?

The following legacy Maven project has been found:
legacy-maven-2

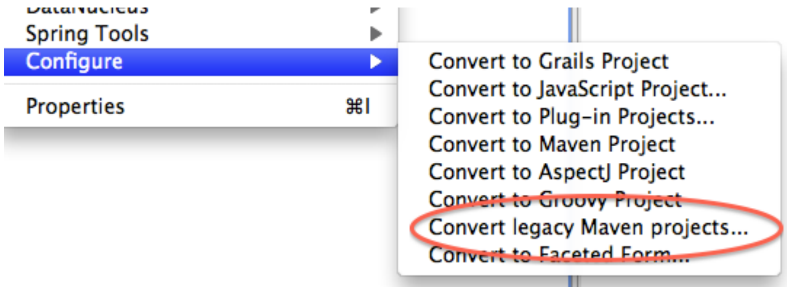
** This project will not compile until it is upgraded to M2E version 1.0. **

Do you want to upgrade now?
You can choose to upgrade later by going to:
Project -> Configure -> Convert legacy Maven projects...

☐ Don't show this dialog again.

No Yes

2. Additionally, this dialog will appear whenever a legacy Maven project is imported into the workspace.
3. If you no longer want this dialog to appear, then you can tick the "Don't show this dialog again" checkbox.
4. You can manually invoke this operation by selecting a legacy Maven project: Right click > Configure > Convert legacy Maven Projects...


DataNucleus
Spring Tools
Configure
Properties ⌘I
Convert to Grails Project
Convert to JavaScript Project...
Convert to Plug-in Projects...
Convert to Maven Project
Convert to AspectJ Project
Convert to Groovy Project
Convert legacy Maven projects...
Convert to Faceted Form...

Note that after the migration, old launch configurations and some custom settings will no longer be working.

If a team is working with different versions of STS (for example, some members on STS 2.8.0 and some on STS 2.7.2), care should be taken as once the projects have been migrated to the latest m2e version, they won't work with the older STS which will still be using the old version of m2e. In this situation it is recommended that those users on STS 2.8.0 convert the legacy projects so that they can be worked on but do **not** commit the updated metadata to a shared repository (.project/.classpath). In that way the users on the older version of STS will not pick it up by accident.

## vFabric tc Server 2.6.1

STS ships now with the latest vFabric tc Server Developer Edition 2.6.1.

## Maven 3.0.3

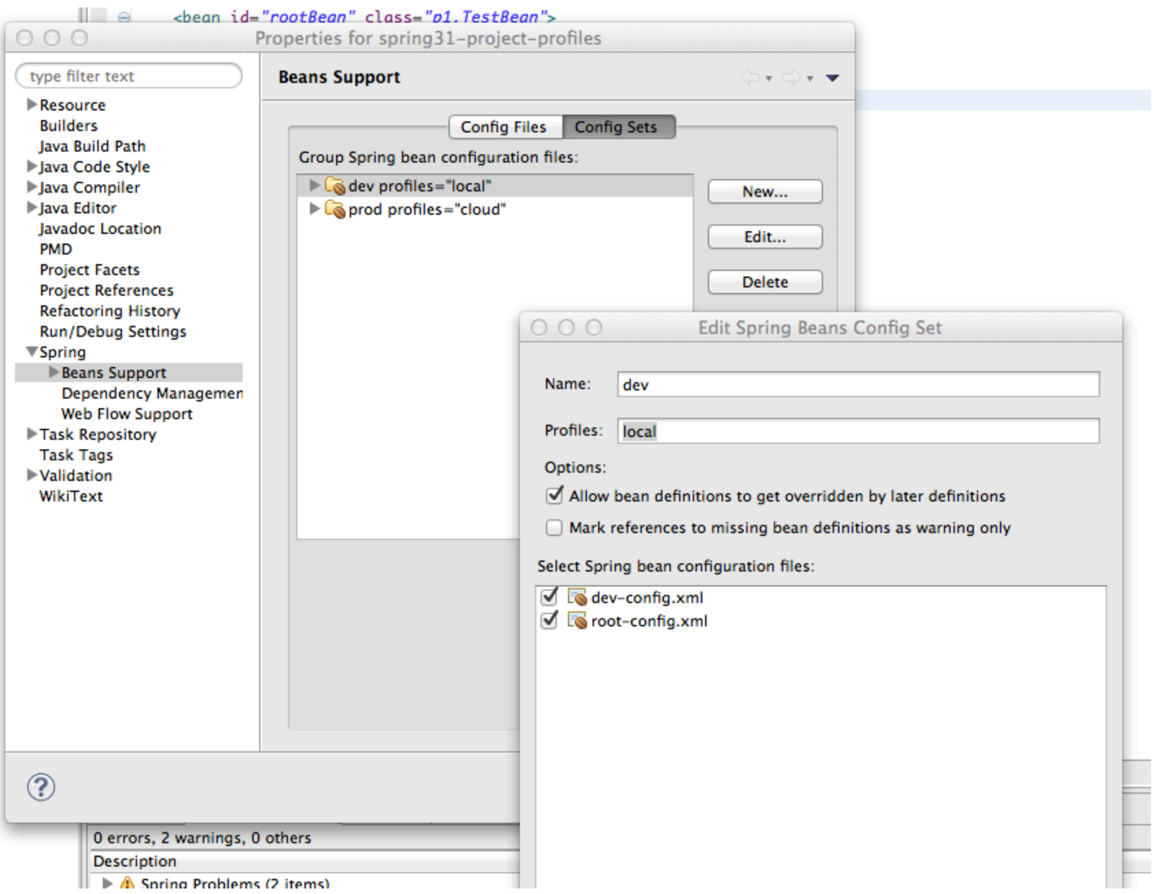
The bundled Maven version got updated to Maven 3.0.3.

## Spring Development Tools

### Spring 3.1 profiles support

STS 2.8.0 provides support for parsing and validating Spring bean definition files based on Spring 3.1, including the new profiles feature.

It is now possible to define the active bean profile for a bean config set to enable profile-aware validation:



```
<bean id="rootBean" class="p1.TestBean">
```

Group Spring bean configuration files:

- dev profiles="local"
- prod profiles="cloud"

Name: dev

Profiles: local

- [x] Allow bean definitions to get overridden by later definitions
- [ ] Mark references to missing bean definitions as warning only

Select Spring bean configuration files:

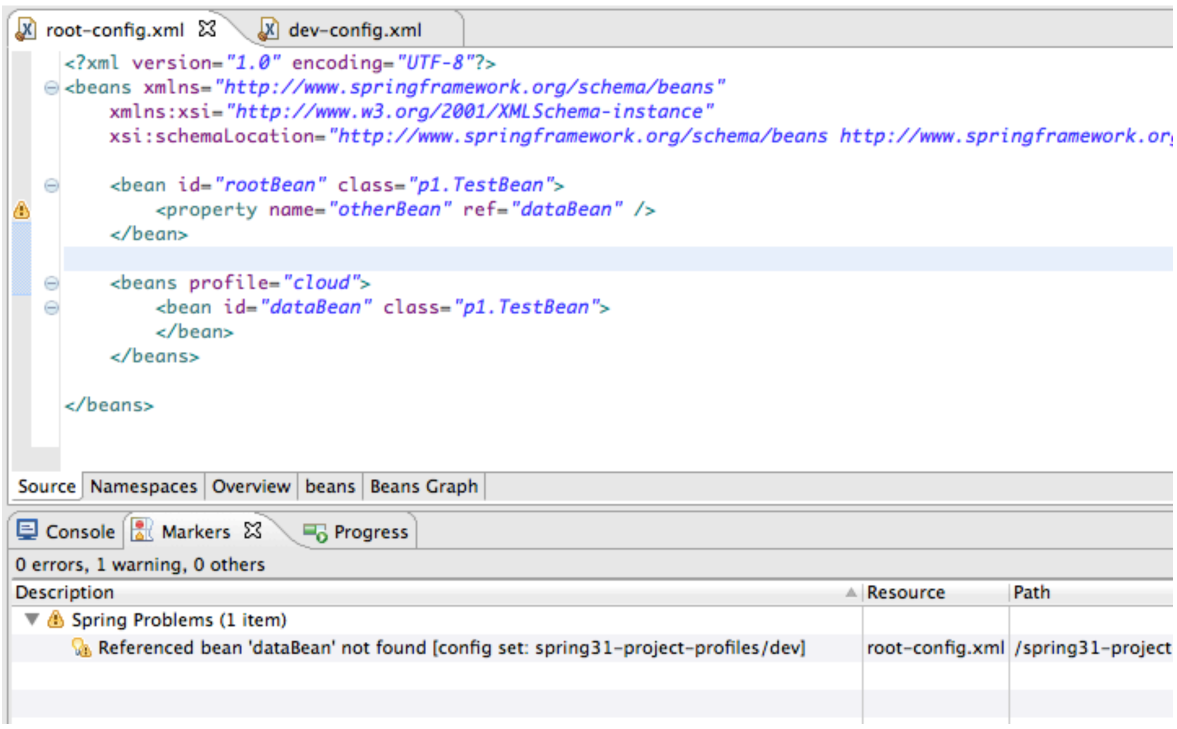
- [x] dev-config.xml
- [x] root-config.xml

0 errors, 2 warnings, 0 others

Description

Spring Problems (2 items)

In this example we have two config sets defined for two different bean profiles. If there is a validation problem in one of those sets, because a referenced bean is not defined for a specific profile, STS will produce an appropriate warning:



root-config.xml | dev-config.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.or

    <bean id="rootBean" class="p1.TestBean">
        <property name="otherBean" ref="dataBean" />
    </bean>

    <beans profile="cloud">
        <bean id="dataBean" class="p1.TestBean">
        </bean>
    </beans>

</beans>
```

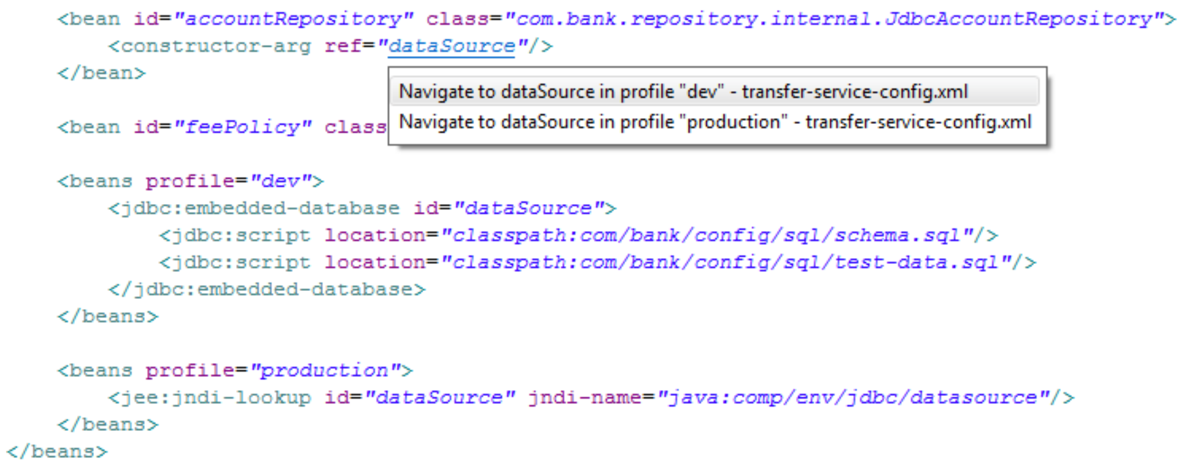
Source | Namespaces | Overview | beans | Beans Graph

Console | Markers | Progress

0 errors, 1 warning, 0 others

| Description | Resource | Path |
|---|---|---|
| Spring Problems (1 item) | | |
| Referenced bean 'dataBean' not found [config set: spring31-project-profiles/dev] | root-config.xml | /spring31-project |

We also added more advanced hyperlinking for profile-based bean definitions. When a bean is defined inside multiple profiles, the Spring Config Editor now shows multiple hyperlinks so user can select the correct bean to navigate to.



```
    <bean id="accountRepository" class="com.bank.repository.internal.JdbcAccountRepository">
        <constructor-arg ref="dataSource"/>
    </bean>

    <bean id="feePolicy" class

    <beans profile="dev">
        <jdbc:embedded-database id="dataSource">
            <jdbc:script location="classpath:com/bank/config/sql/schema.sql"/>
            <jdbc:script location="classpath:com/bank/config/sql/test-data.sql"/>
        </jdbc:embedded-database>
    </beans>

    <beans profile="production">
        <jee:jndi-lookup id="dataSource" jndi-name="java:comp/env/jdbc/datasource"/>
    </beans>
</beans>
```

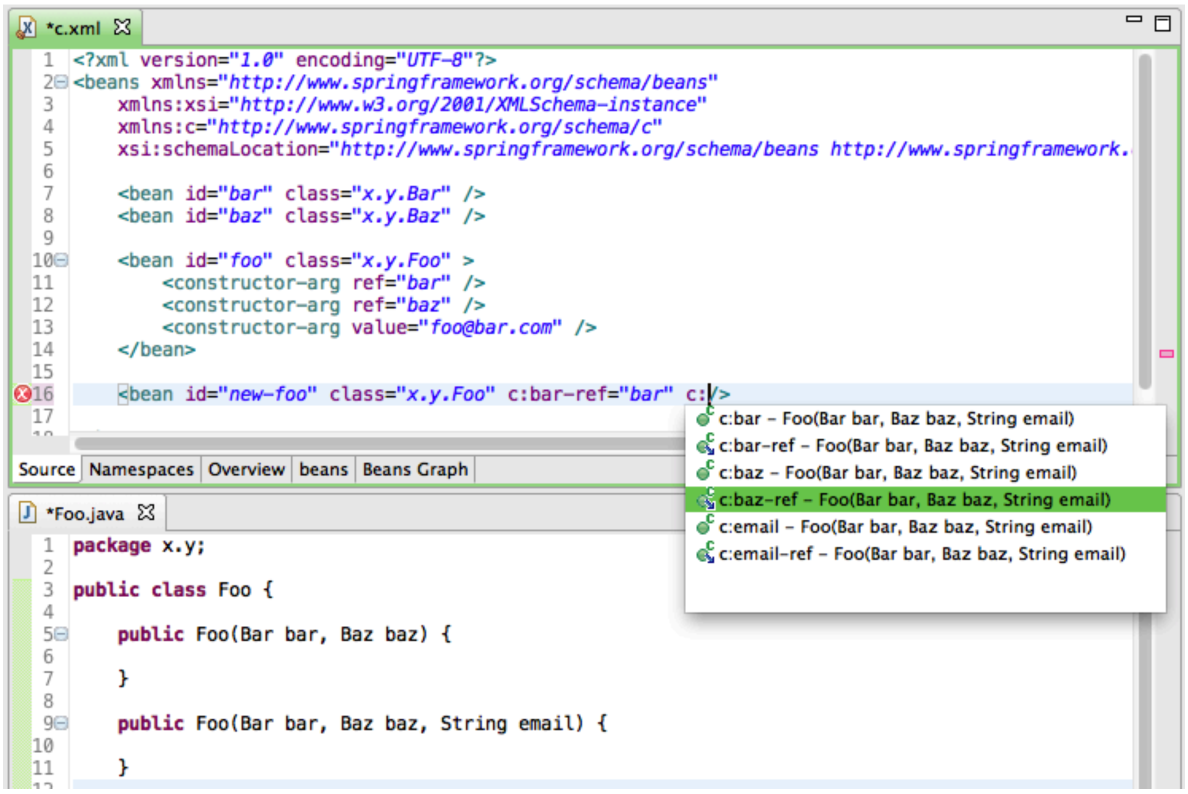
Navigate to dataSource in profile "dev" - transfer-service-config.xml
Navigate to dataSource in profile "production" - transfer-service-config.xml

## Spring 3.1 c-namespace support

STS 2.8.0 adds support for the new c-namespace introduced in Spring 3.1

(http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/beans.html#beans-c-namespace).

Content assist has been added for c-namespace parameter names, as well as for ref-values. Editor validation will quickly inform you if your constructor has the right number of parameters, or if your parameter names are unrecognized. Bean ID refactorings will be picked up by c-namespace ref-values as well.



```
<?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:c="http://www.springframework.org/schema/c"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.

    <bean id="bar" class="x.y.Bar" />
    <bean id="baz" class="x.y.Baz" />

    <bean id="foo" class="x.y.Foo" >
        <constructor-arg ref="bar" />
        <constructor-arg ref="baz" />
        <constructor-arg value="foo@bar.com" />
    </bean>

    <bean id="new-foo" class="x.y.Foo" c:bar-ref="bar" c:/>
```

```
c:bar – Foo(Bar bar, Baz baz, String email)
c:bar-ref – Foo(Bar bar, Baz baz, String email)
c:baz – Foo(Bar bar, Baz baz, String email)
c:baz-ref – Foo(Bar bar, Baz baz, String email)
c:email – Foo(Bar bar, Baz baz, String email)
c:email-ref – Foo(Bar bar, Baz baz, String email)
```

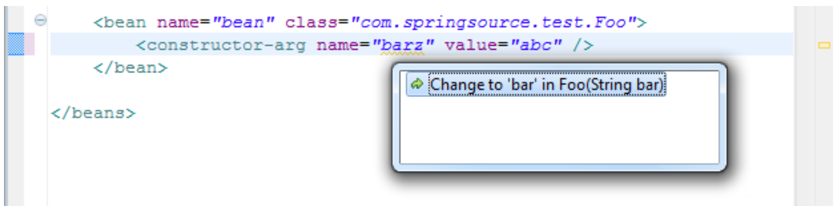
```
package x.y;

public class Foo {

    public Foo(Bar bar, Baz baz) {

    }

    public Foo(Bar bar, Baz baz, String email) {

    }
```

## JDK 1.7.0 and Spring Roo

There was a problem running Spring Roo on top of JDK 1.7.0. While this got fixed in Roo for the upcoming 1.2.0.M1 version of Roo, we managed to fix this within STS as well, so that you can now not only run Spring Roo 1.2.0.M1 on top of JDK 1.7.0, but also the latest Spring Roo 1.1.5 release.

## As-you-type validation and quick fixes for constructor-arg

In this milestone we added as-you-type validation and quick fixes for the name attribute in a constructor-arg element. A warning (yellow) line appears under a name attribute that does not match with any constructor parameter names, and Ctrl+1 brings up a list of possible parameter names to rename to.



```
<bean name="bean" class="com.springsource.test.Foo">
    <constructor-arg name="barz" value="abc" />
</bean>

</beans>
```

## Groovy Eclipse

### Compiler switching

#### Per workspace compiler selection – via command line argument

Groovy-Eclipse now makes it easier to support multiple compiler levels on one installation. To specify a compiler level on the command line, add arguments:

```
sts <OTHER_ARGUMENTS> -groovy.compiler.level 18
```

(For Groovy 1.8.2. Use -groovy.compiler.level 17 for Groovy 1.7.10.)

It is possible to launch two workspaces simultaneously that use different compiler levels if they are specified on the command line. Note that you must explicitly install Groovy 1.8.2 from the update site or the STS dashboard. For more information, please see the docs.

http://docs.codehaus.org/display/GROOVY/Compiler+Switching+within+Groovy-Eclipse#CompilerSwitchingwithinGroovy-Eclipse-Specifyingcompilerlevelonthecommandline

### Groovy 1.8.2

Groovy Eclipse now has Groovy 1.8.2 available as an optional install. See the compiler switching link above for more information on this.

### Groovy-Eclipse configurator for m2e v1.0

The Groovy-Eclipse configurator for m2e (http://www.eclipse.org/m2e) is now available from the Groovy-Eclipse update site:

http://dist.springsource.org/snapshot/GRECLIPSE/e3.7

This configurator supports m2e version 1.0 and provides first class support for your Maven and Groovy project. This configurator supports both GMaven and the groovy-eclipse-compiler. For more information on Groovy-Eclipse's maven support see the docs:

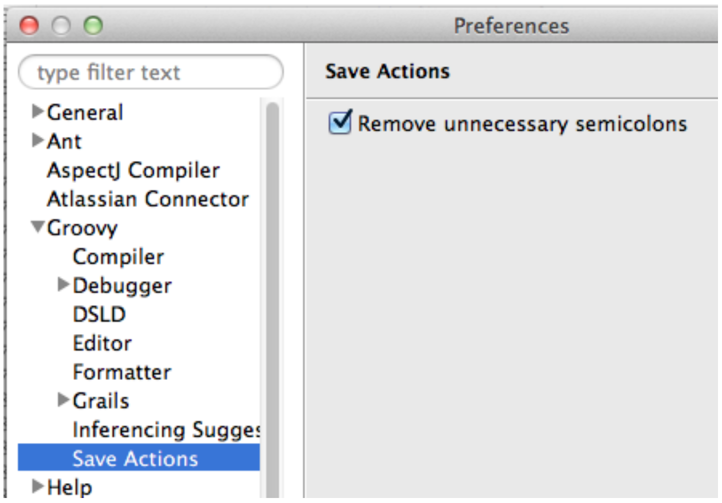
http://docs.codehaus.org/display/GROOVY/Groovy-Eclipse+compiler+plugin+for+Maven

Recent improvements in the groovy-eclipse-compiler plugin for Maven include:

- Uses Groovy 1.8.2 by default (can be configured to use 1.7.10)
- The Java compiler is now Java 7 based (from Eclipse 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 automatically to the build if they exist.
- Updated groovy-eclipse-compiler archetype project. This archetype is available at: https://nexus.codehaus.org/content/repositories/snapshots/

## Remove trailing whitespace and semi-colons save actions

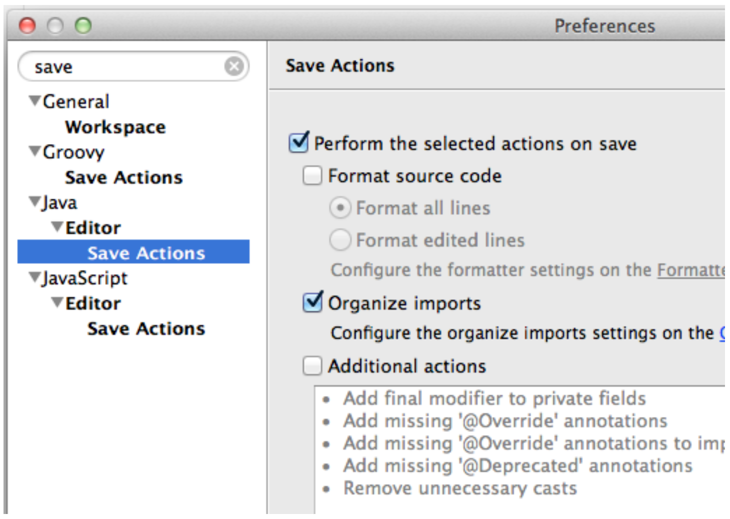
It is now possible to configure Groovy-Eclipse to automatically remove trailing whitespace and semi-colons on file-save. Go to **Preferences > Groovy > Save** actions to enable it:



Save Actions

☑ Remove unnecessary semicolons

It is also necessary to enable save actions for Java editors, which you can do can do by going to **Preferences > Java > Editor > Save** actions:



Save Actions

☑ Perform the selected actions on save

☐ Format source code

● Format all lines

○ Format edited lines

Configure the formatter settings on the Formatte

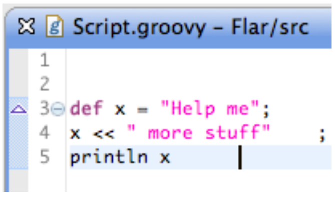
☑ Organize imports

Configure the organize imports settings on the

☐ Additional actions

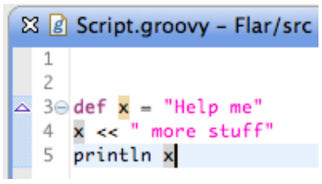
- Add final modifier to private fields
- Add missing '@Override' annotations
- Add missing '@Override' annotations to im
- Add missing '@Deprecated' annotations
- Remove unnecessary casts

Enabling the save action will convert this file:


Script.groovy – Flar/src

```


def x = "Help me";
x << " more stuff"     ;
println x
```

into this file:


Script.groovy – Flar/src

```


def x = "Help me"
x << " more stuff"
println x
```

## Enhanced type inferencing

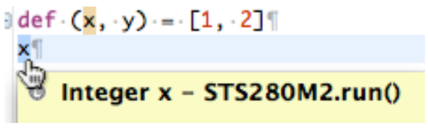
We have added many small enhancements with the Groovy-Eclipse inferencing engine. A list of all inferencing issues addressed is available here.

http://jira.codehaus.org/secure/IssueNavigator.jspa?reset=true&jqlQuery=project+%3D+GRECLIPSE+AND+fixVersion+%3D+17434+AND+component+%3D+%22Inferencing+Engine%22+AND+status+in+%28Resolved%2C+Closed%29+ORDER+BY+priority+DESC

Here are some highlights:

### Support for multiple declaration statements

Now, multiple declaration statements are correctly inferred:

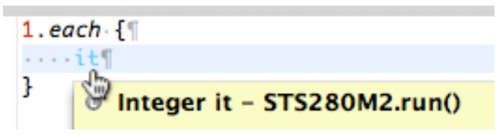


```
def (x, y) = [1, 2]
x
```

Integer x – STS280M2.run()

### 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:

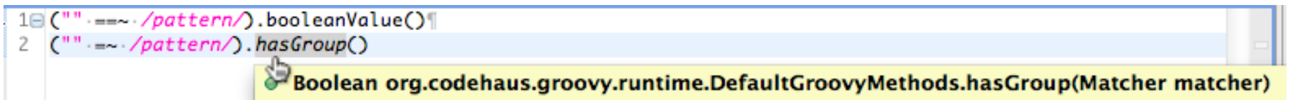


```
1.each {
    it
}
```

Integer it – STS280M2.run()

### Type inferencing for match operators

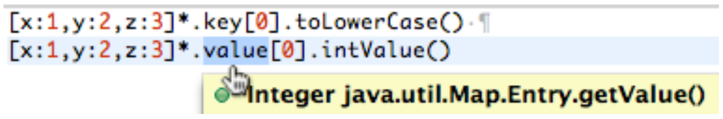
The =~ and ==~ operators now have appropriate inferencing:



```
("" ==~ /pattern/).booleanValue()
("" =~ /pattern/).hasGroup()
```
Boolean org.codehaus.groovy.runtime.DefaultGroovyMethods.hasGroup(Matcher matcher)

### Spread-dot type inferencing

The spread-dot operator is now correctly handled by the inferencing engine:



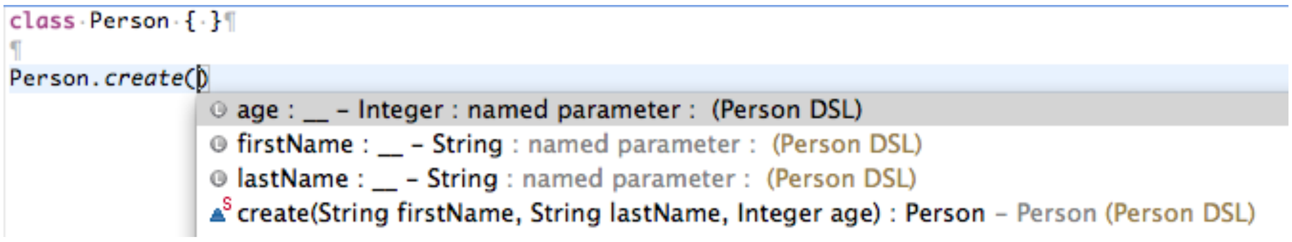
```
[x:1,y:2,z:3]*.key[0].toLowerCase()
[x:1,y:2,z:3]*.value[0].intValue()
```
Integer java.util.Map.Entry.getValue()

## DSL Support

We are continuing to improve on our DSL support in Groovy-Eclipse through our DSL descriptors. Below are the significant enhancements.

### Named parameter content assist

Methods that expect named parameters (typically specified in a DSL descriptor) can have their named parameters filled in via content assist, as shown in this screenshot:

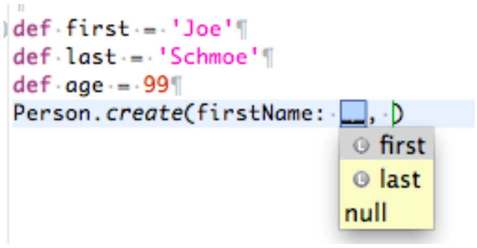


```
class Person { }

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.

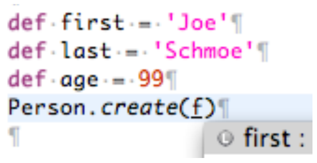
(http://blog.springsource.com/2011/05/08/better-dsl-support-in-groovy-eclipse/)

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)
```

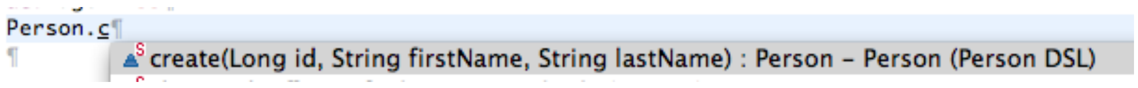
## namedParams and optionalParams for method contributions

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. Their position is after all the regular parameters. Finally, 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.

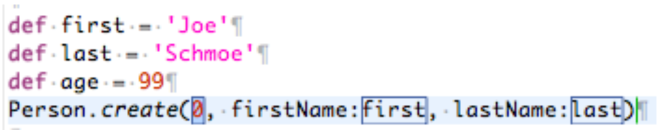
An example should make this clear. Consider the following simple DSLD file that add editing support for the static 'create' method to the Person class:

```
currentType("Person").accept {
     method name:"create", type:"Person", params:[id:Long],
        namedParams:[firstName:String, lastName:String],
        optionalParams:[age:Integer],
        isStatic:true, provider:"Person DSL"
}
```

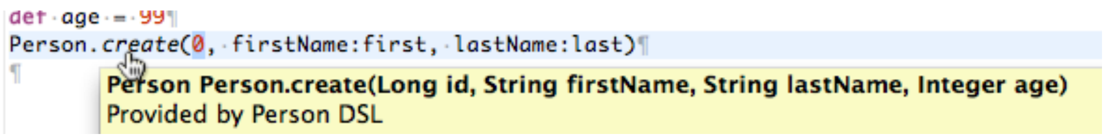
The method contribution defines params, namedParams, and optionalParams. The first thing to see is that when doing content assist, only the regular and named parameters will be shown:


```
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):

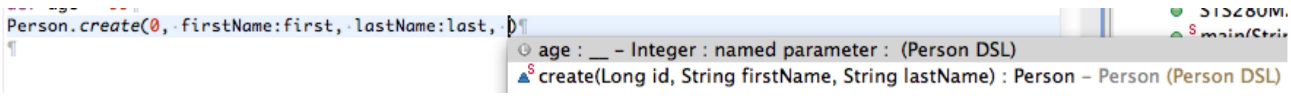

```
def first = 'Joe'
def last = 'Schmoe'
def age = 99
Person.create(0, firstName:first, lastName:last)
```

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

springsource A division of vmware

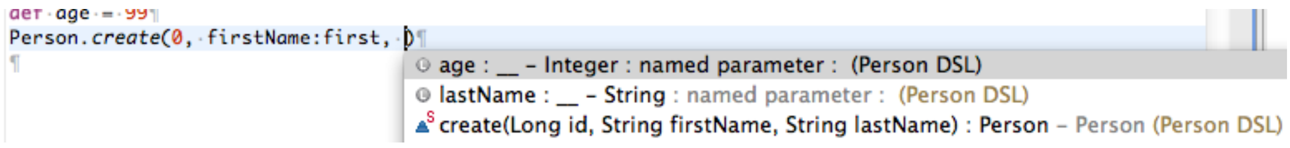
They can also be seen in content assist, when named parameters would be visible:



```
Person.create(0, firstName:first, lastName:last, )
```

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:

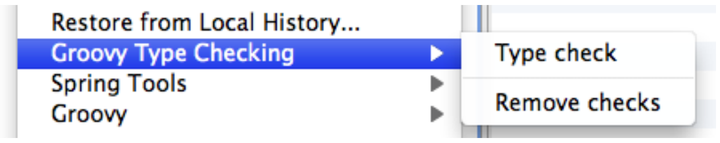


```
def age = 99
Person.create(0, firstName:first, )
```

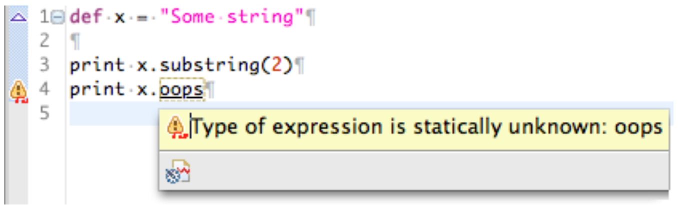
age : __ – Integer : named parameter : (Person DSL)
lastName : __ – String : named parameter : (Person DSL)
create(Long id, String firstName, String lastName) : Person – Person (Person DSL)

## Static type checking

This version of Groovy-Eclipse 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:

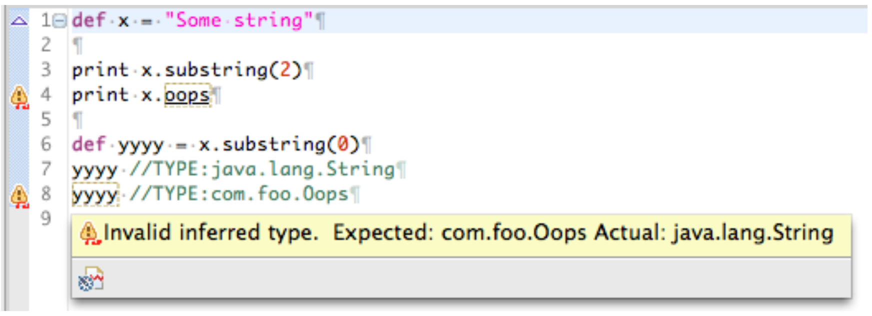


```
def x = "Some string"

print x.substring(2)
print x.oops
```

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):



```
def x = "Some string"

print x.substring(2)
print x.oops

def yyyy = x.substring(0)
yyyy //TYPE:java.lang.String
yyyy //TYPE:com.foo.Oops
```

Invalid inferred type. Expected: com.foo.Oops Actual: java.lang.String

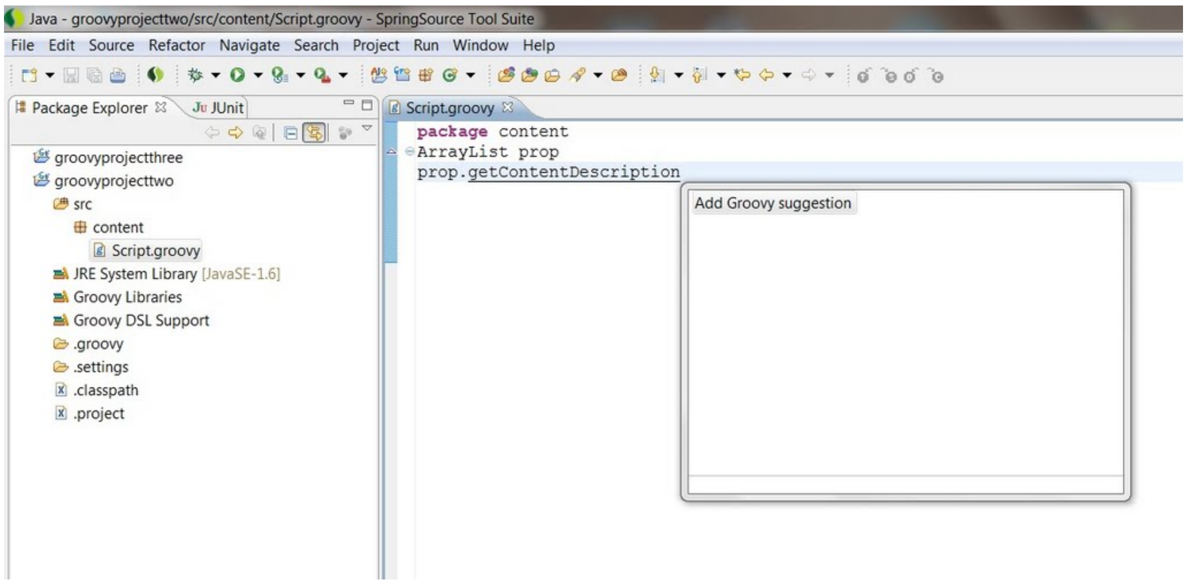
This feature is still experimental and we are looking for feedback on it. If you find this feature useful, if you have any suggestions for it, or if you have found problems with it, please contact us on the STS forum to discuss.

http://forum.springsource.org/forumdisplay.php?32-SpringSource-Tool-Suite

## User contributed inferencing property and method suggestions

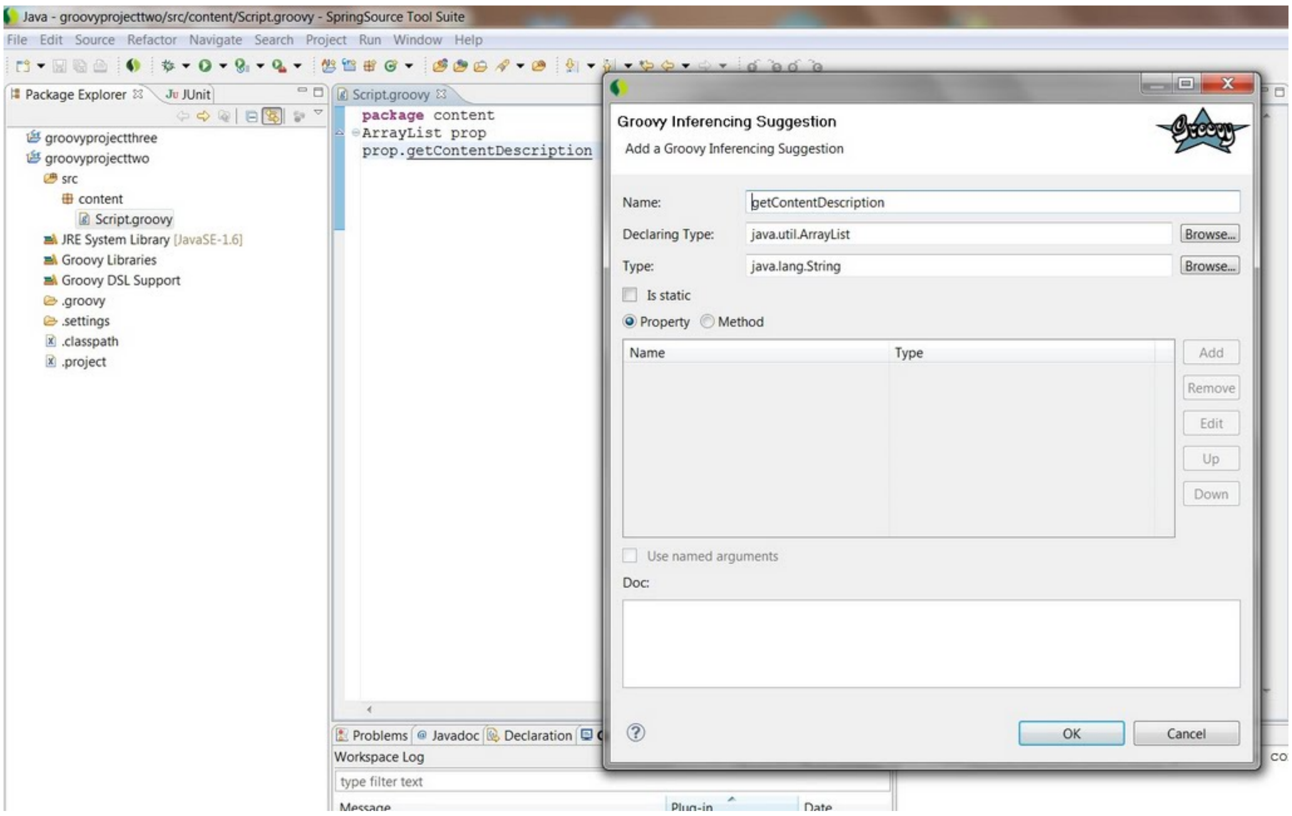
There is now another way to extend Groovy-Eclipse's inferencing engine. Using quick assists and a preferences page, users can now contribute dynamic properties and methods to existing classes for a customized editing experience.

As seen in the image below it is now possible to invoke a Quick-Assist (CTRL-1 or CMD-1 on macs) 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.



```
package content
ArrayList prop
prop.getContentDescription
```

The Groovy Suggestions dialogue then opens, pre-populated with values related to the current selection, like the declaring type, property name and type:



```
package content
ArrayList prop
prop.getContentDescription
```

**Groovy Inferencing Suggestion**

Add a Groovy Inferencing Suggestion

Name: getContentDescription

Declaring Type: java.util.ArrayList [Browse...]

Type: java.lang.String [Browse...]

☐ Is static

[✓ Property] Method

| Name | Type |
|---|---|

Add / Remove / Edit / Up / Down

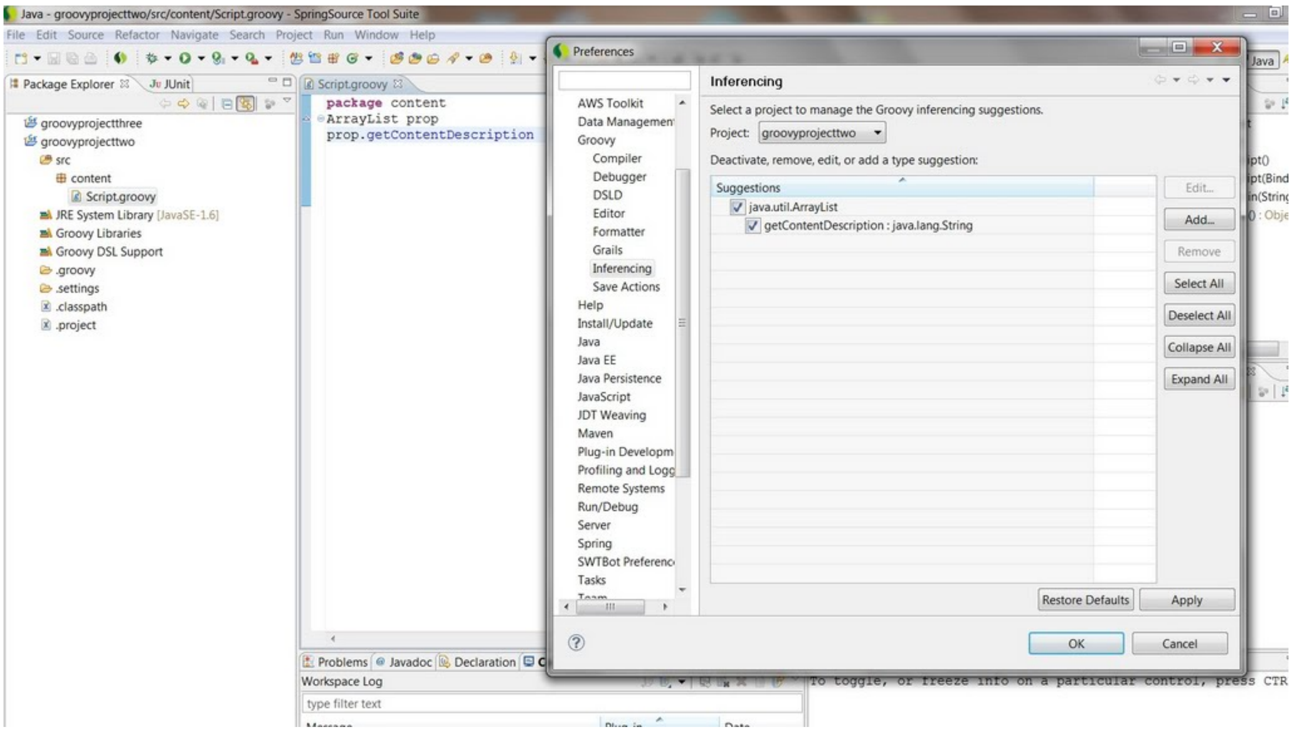
☐ Use named arguments

Doc:

OK Cancel

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.



```
package content
ArrayList prop
prop.getContentDescription
```

Inferencing

Select a project to manage the Groovy inferencing suggestions.

Project: groovyprojecttwo

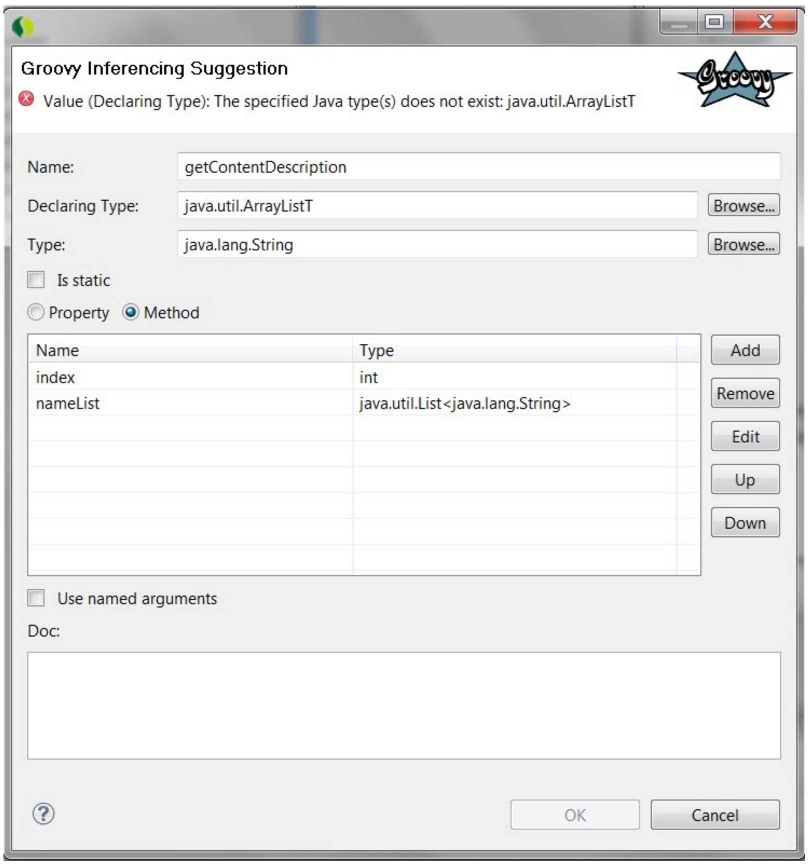
Deactivate, remove, edit, or add a type suggestion:

Suggestions
- [✓] java.util.ArrayList
  - [✓] getContentDescription : java.lang.String

Suggestions are persisted in the /.groovy/suggestions.xdsl file at the root of all Groovy projects. Each Groovy project has it's own

suggestions.xdsl and so they can be persisted in version control for the project. As of M2, 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:



Groovy Inferencing Suggestion

Value (Declaring Type): The specified Java type(s) does not exist: java.util.ArrayListT

Name: getContentDescription

Declaring Type: java.util.ArrayListT

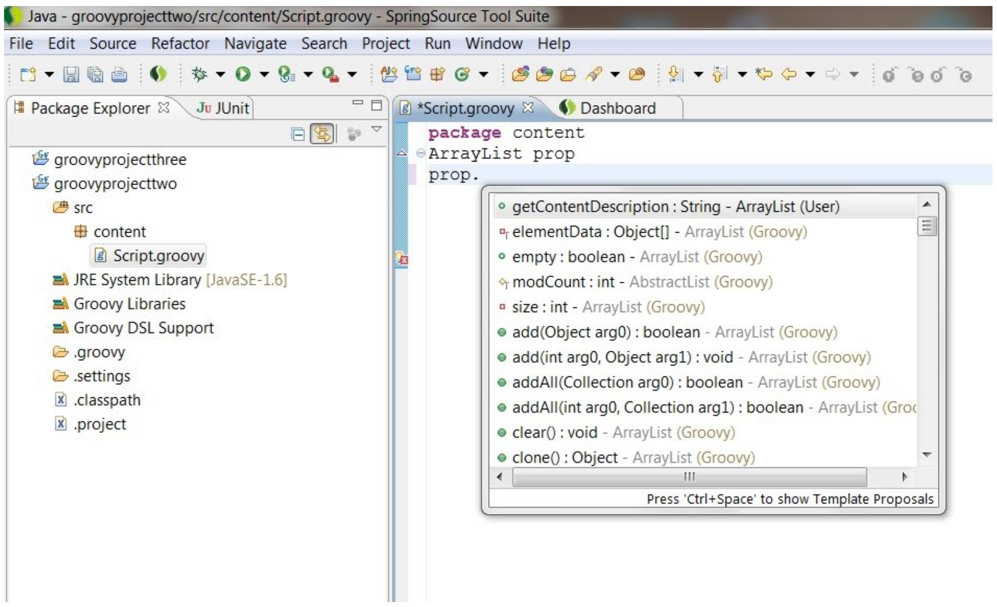
Type: java.lang.String

Is static

Property [✓ Method]

| Name | Type |
|---|---|
| index | int |
| nameList | java.util.List<java.lang.String> |

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:



Java - groovyprojecttwo/src/content/Script.groovy - SpringSource Tool Suite

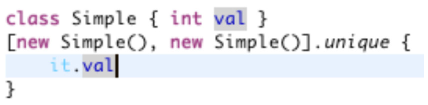
```
package content
ArrayList prop
prop.
```

```
getContentDescription : String - ArrayList (User)
elementData : Object[] - ArrayList (Groovy)
empty : boolean - ArrayList (Groovy)
modCount : int - AbstractList (Groovy)
size : int - ArrayList (Groovy)
add(Object arg0) : boolean - ArrayList (Groovy)
add(int arg0, Object arg1) : void - ArrayList (Groovy)
addAll(Collection arg0) : boolean - ArrayList (Groovy)
addAll(int arg0, Collection arg1) : boolean - ArrayList (Groo
clear() : void - ArrayList (Groovy)
clone() : Object - ArrayList (Groovy)
Press 'Ctrl+Space' to show Template Proposals
```

springsource A division of vmware

## 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
}
```

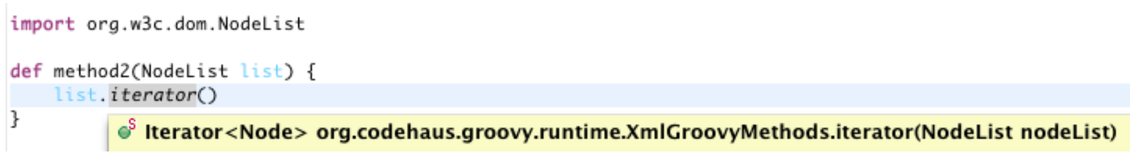
In addition to unique all methods inside the DefaultGroovyMethods class now support inferencing on parameters in closures where possible. A complete list is available in:

http://jira.codehaus.org/browse/GRECLIPSE-1143.

## Support for DateGroovyMethods, SwingGroovyMethods, XmlGroovyMethods, and ProcessGroovyMethods

These classes are now handled just like DefaultGroovyMethods in terms of content assist, hovers, and navigation. For example:

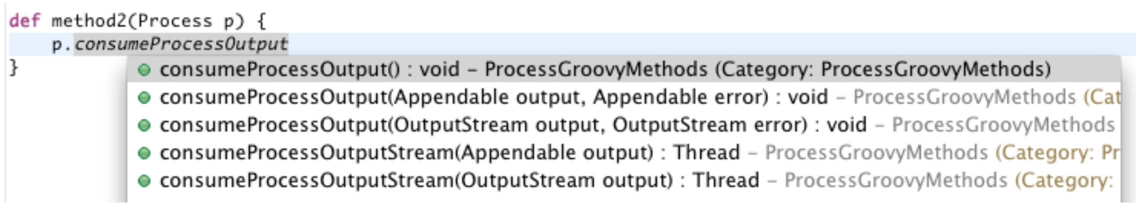
XmlGroovyMethods:



```
import org.w3c.dom.NodeList

def method2(NodeList list) {
    list.iterator()
}
```

Iterator<Node> org.codehaus.groovy.runtime.XmlGroovyMethods.iterator(NodeList nodeList)

ProcessGroovyMethods:



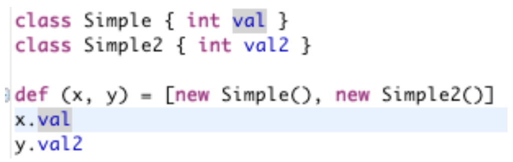
```
def method2(Process p) {
    p.consumeProcessOutput
}
```

consumeProcessOutput() : void – ProcessGroovyMethods (Category: ProcessGroovyMethods)
consumeProcessOutput(Appendable output, Appendable error) : void – ProcessGroovyMethods (Cat
consumeProcessOutput(OutputStream output, OutputStream error) : void – ProcessGroovyMethods
consumeProcessOutputStream(Appendable output) : Thread – ProcessGroovyMethods (Category: Pr
consumeProcessOutputStream(OutputStream output) : Thread – ProcessGroovyMethods (Category:

For more information, see GRECLIPSE-1131, GRECLIPSE-1143, GRECLIPSE-1145, GRECLIPSE-1153, GRECLIPSE-1154 and GRECLIPSE-1155.

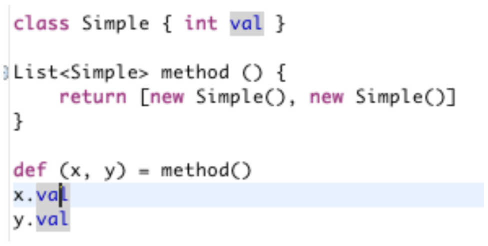
## Multiple assignment statements

The types of variables assigned in multi-assignment statements are 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
```

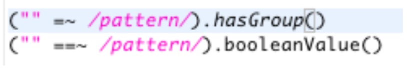
Also, when a method returns a list or array, the type parameter or component type is used as the type of the assigned variable:



```
class Simple { int val }

List<Simple> method () {
    return [new Simple(), new Simple()]
}

def (x, y) = method()
x.val
y.val
```

## Match operator inferencing

Match operator expressions now support inferencing:



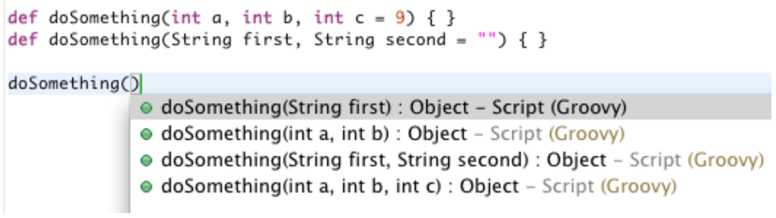
```
("" =~ /pattern/).hasGroup()
("" ==~ /pattern/).booleanValue()
```

For more information, see GRECLIPSE-1159.

## Method context information

Groovy-Eclipse now follows the JDT model of displaying context information for methods when invoking content assist inside of a method call and there is no prefix.

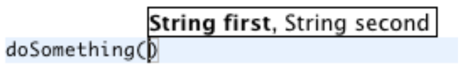
For example, when invoking content assist just after an opening paren, you are only shown a list of known ways to invoke the target method:



```
def doSomething(int a, int b, int c = 9) { }
def doSomething(String first, String second = "") { }

doSomething()
```

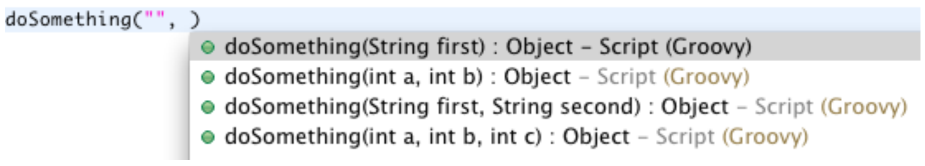
- doSomething(String first) : Object – Script (Groovy)
- doSomething(int a, int b) : Object – Script (Groovy)
- doSomething(String first, String second) : Object – Script (Groovy)
- doSomething(int a, int b, int c) : Object – Script (Groovy)

Selecting one of the proposals will show that methods information in tooltips above the caret location:



```
String first, String second
doSomething()
```

Similarly, invoking content assist after a comma will show context information as well:



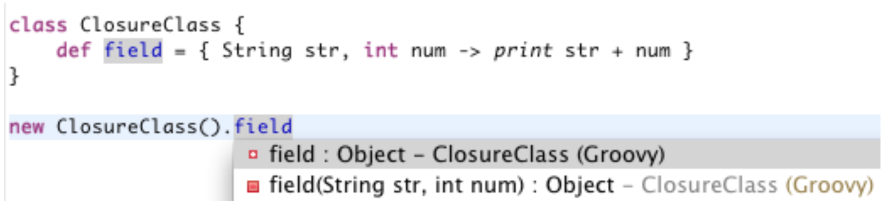
```
doSomething("", )
```

- 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)
```

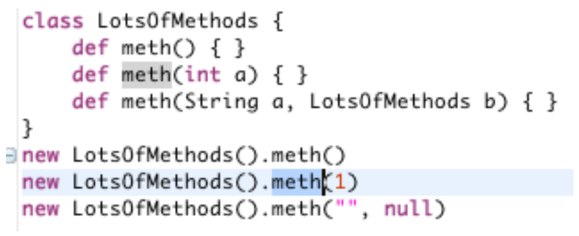
For more information, see GRECLIPSE-1139.

## 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 GRECLIPSE-682 for more information.

## More precise searching for overloaded methods

The number of parameters of method declarations are now used to help more precisely determine the actual declaration of a method reference:

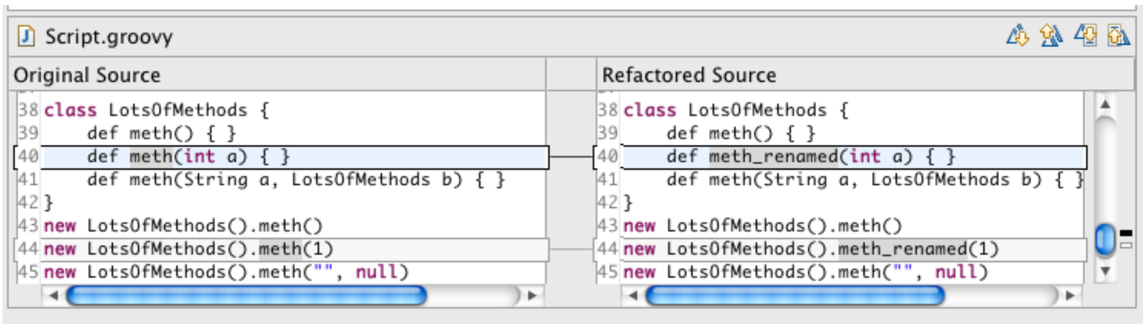


```
class LotsOfMethods {
    def meth() { }
    def meth(int a) { }
    def meth(String a, LotsOfMethods b) { }
}
new LotsOfMethods().meth()
new LotsOfMethods().meth(1)
new LotsOfMethods().meth("", null)
```

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:



Script.groovy

Original Source

```
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)
```

Refactored Source

```
class LotsOfMethods {
    def meth() { }
    def meth_renamed(int a) { }
    def meth(String a, LotsOfMethods b) { }
}
new LotsOfMethods().meth()
new LotsOfMethods().meth_renamed(1)
new LotsOfMethods().meth("", null)
```

### 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.

## Grails Development Tools

### Support for Grails 2.0.0.M2

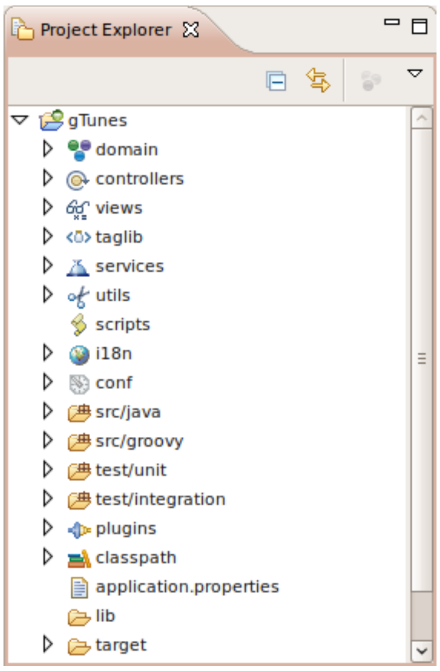
STS 2.8.0 now supports Grails 2.0.0.M2 and has been tested with snapshot builds leading up to the not-yet-released 2.0.0.RC1.

### Grails Project Explorer: Icons and element ordering improved

The Grails project explorer view has been cleaned-up:

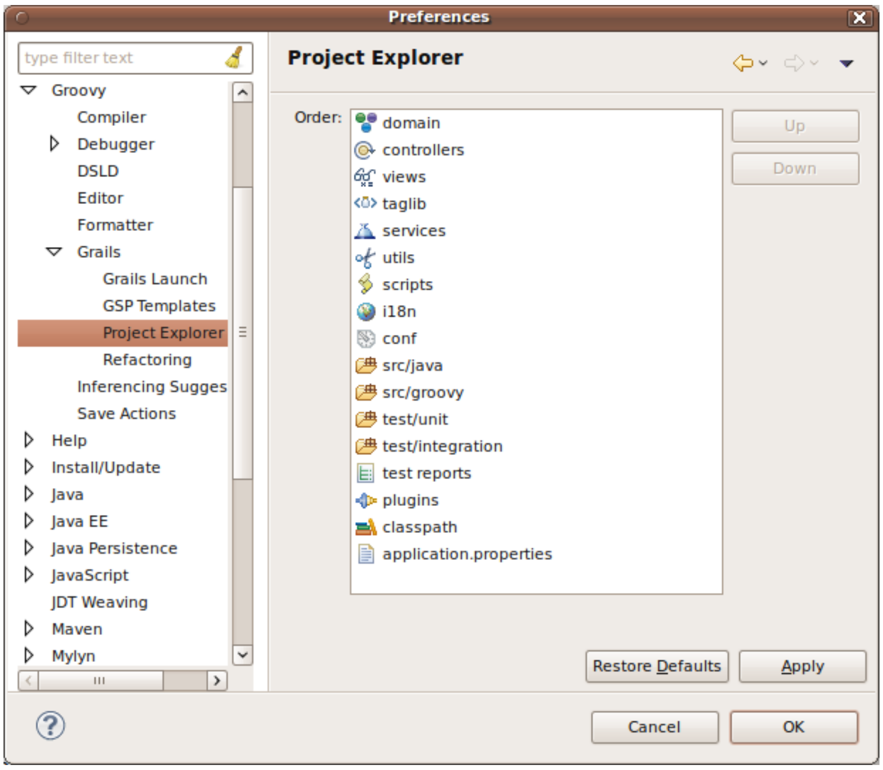
- A new icon added for grails-app/utils
- Default ordering of elements improved based on user suggestions

The default view is now like this:



## Grails Project Explorer: Element order now configurable

The order of elements in the Grails Project Explorer is now a global workspace preference that can be changed via **Windows > Preferences > Groovy > Grails > Project Explorer:**



Order:
- domain
- controllers
- views
- taglib
- services
- utils
- scripts
- i18n
- conf
- src/java
- src/groovy
- test/unit
- test/integration
- test reports
- plugins
- classpath
- application.properties

## Grails pergmen default raised to 192m

Default permgen size for Grails commands launched from STS has been raised to 192m (previously 96m).

The default JVM launch arguments for running Grails commands are now:

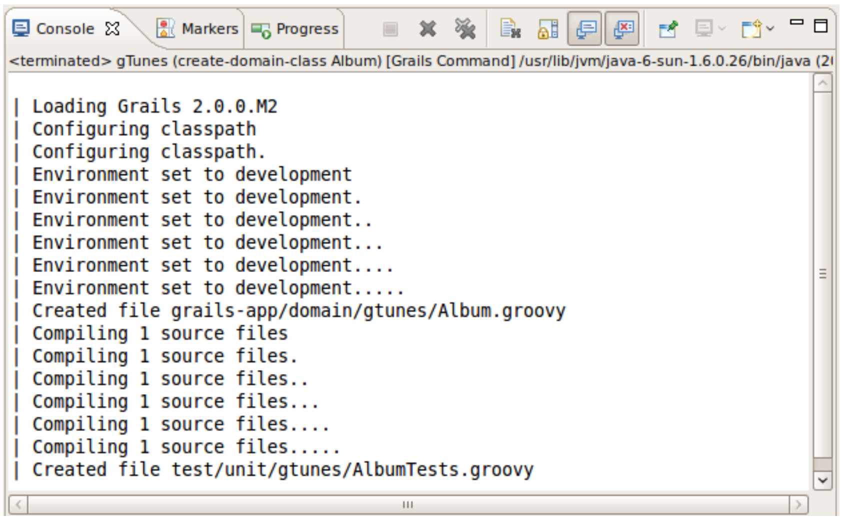
```
-server -Xmx512M -XX:MaxPermSize=192m
```

Note that the arguments are only provided by STS if they are not already set by the existing run-app or test-app launch configuration.

Existing launch configurations and their custom vm arguments can be inspected and changed via the "Run" menu under: **Run > Run Configurations > Grails** see the **Arguments** tab.

## Grails 2.0 output cleanup

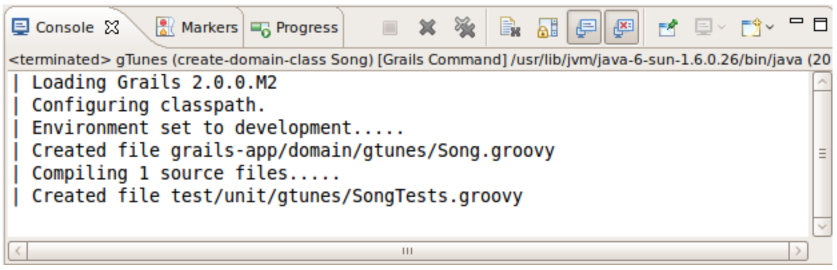
Grails 2.0 uses ANSI control characters to make output look nicer on the command line. Unfortunately, Eclipse Console View does not support this. This makes the Grails 2.0 output not look as nice inside STS. For example, look at the screenshot below and note the repeated status messages:



<terminated> gTunes (create-domain-class Album) [Grails Command] /usr/lib/jvm/java-6-sun-1.6.0.26/bin/java (2

```
| Loading Grails 2.0.0.M2
| Configuring classpath
| Configuring classpath.
| Environment set to development
| Environment set to development.
| Environment set to development..
| Environment set to development...
| Environment set to development....
| Environment set to development.....
| Created file grails-app/domain/gtunes/Album.groovy
| Compiling 1 source files
| Compiling 1 source files.
| Compiling 1 source files..
| Compiling 1 source files...
| Compiling 1 source files....
| Compiling 1 source files.....
| Created file test/unit/gtunes/AlbumTests.groovy
```

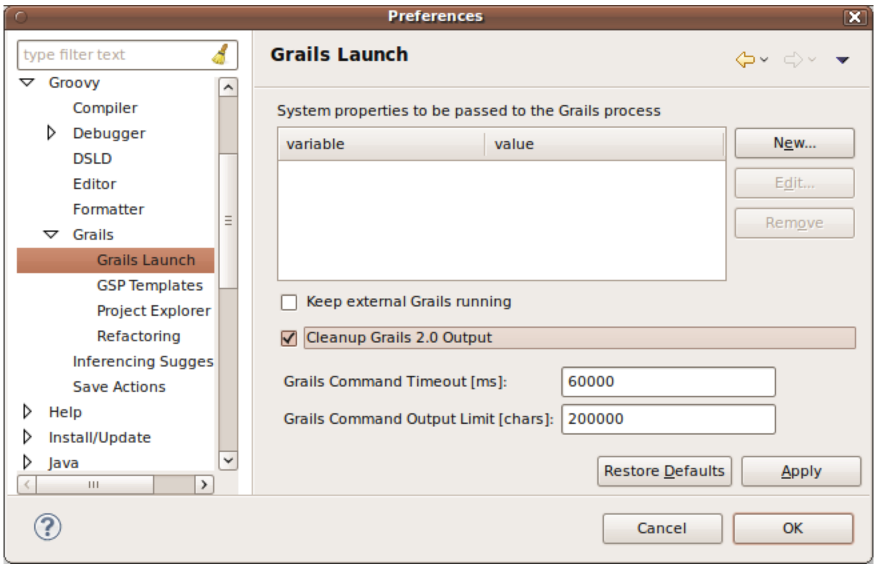
STS now includes an output transformer that attempts to prettify this output. The screenshot below shows what Grails 2.0 output looks like with the output transformer activated:



<terminated> gTunes (create-domain-class Song) [Grails Command] /usr/lib/jvm/java-6-sun-1.6.0.26/bin/java (20

```
| Loading Grails 2.0.0.M2
| Configuring classpath.
| Environment set to development.....
| Created file grails-app/domain/gtunes/Song.groovy
| Compiling 1 source files.....
| Created file test/unit/gtunes/SongTests.groovy
```

Should you wish to disable the output transformer for some reason, it can be found on the Grails launch preferences page:



Grails Launch

System properties to be passed to the Grails process

☐ Keep external Grails running

☑ Cleanup Grails 2.0 Output

Grails Command Timeout [ms]: 60000

Grails Command Output Limit [chars]: 200000

## Grails aware refactoring

### Renaming of Views and Actions

A Grails aware rename participant is triggered when a controller action method or field is renamed. (In Grails 2.0 and later, controllers actions can be defined as methods or fields.)

It performs the following kind of changes in addition to a normal Java method or field rename:

- find and rename references in g:link tags in GSP files in the same project. E.g

  `<g:link controller="song" action="list">`

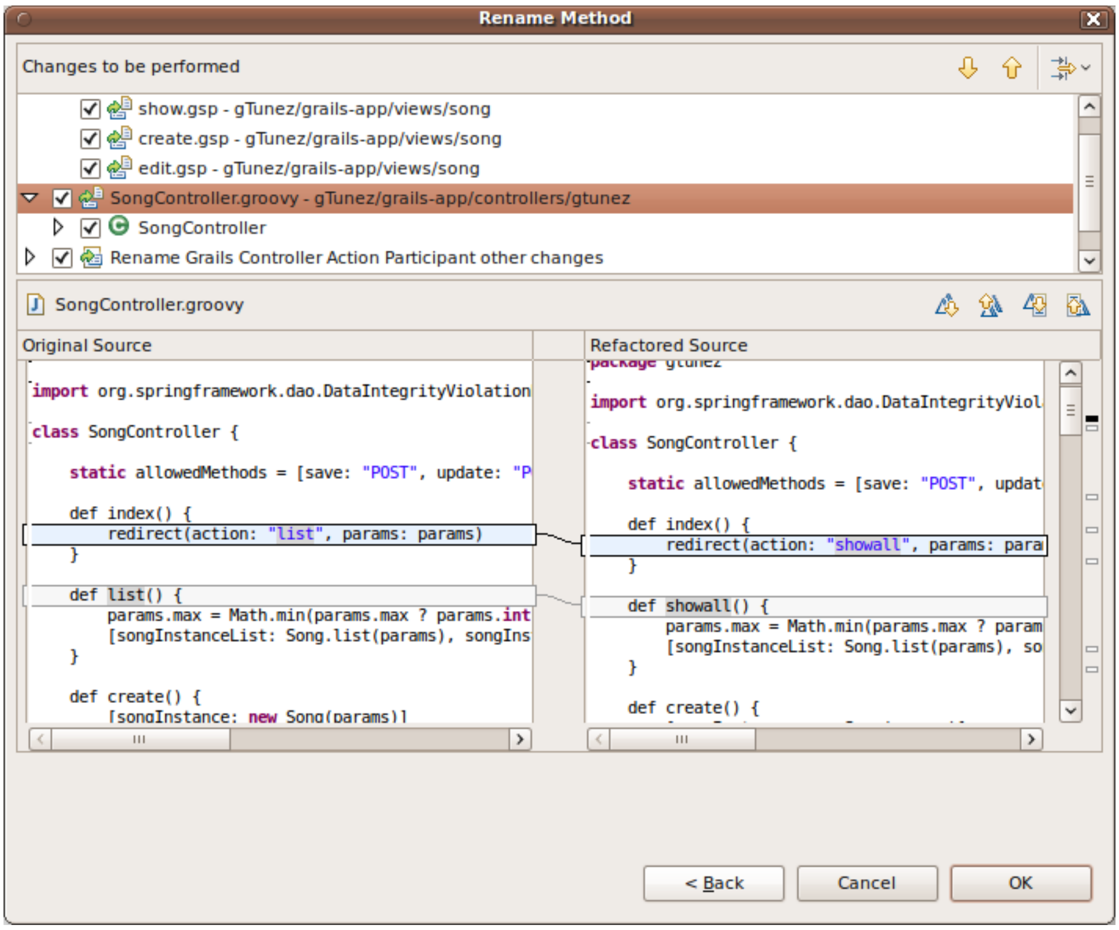
- render and redirect method calls in controller classes E.g

  `render(view: "list" ...)`

- rename the corresponding view (gsp file). E.g.

  rename **views/song/list.gsp to views/song/<new-name>.gsp**

Preview of a method rename refactoring, including some changes added by the Grails Aware Refactoring Participant:



- [✓] show.gsp - gTunez/grails-app/views/song
- [✓] create.gsp - gTunez/grails-app/views/song
- [✓] edit.gsp - gTunez/grails-app/views/song
- [✓] SongController.groovy - gTunez/grails-app/controllers/gtunez
  - [✓] SongController
- [✓] Rename Grails Controller Action Participant other changes

SongController.groovy

Original Source

```
import org.springframework.dao.DataIntegrityViolation

class SongController {

    static allowedMethods = [save: "POST", update: "P

    def index() {
        redirect(action: "list", params: params)
    }

    def list() {
        params.max = Math.min(params.max ? params.int
        [songInstanceList: Song.list(params), songIns
    }

    def create() {
        [songInstance: new Song(params)]
```

Refactored Source

```
package gtunez

import org.springframework.dao.DataIntegrityViol

class SongController {

    static allowedMethods = [save: "POST", updat

    def index() {
        redirect(action: "showall", params: para
    }

    def showall() {
        params.max = Math.min(params.max ? param
        [songInstanceList: Song.list(params), so
    }

    def create() {
```

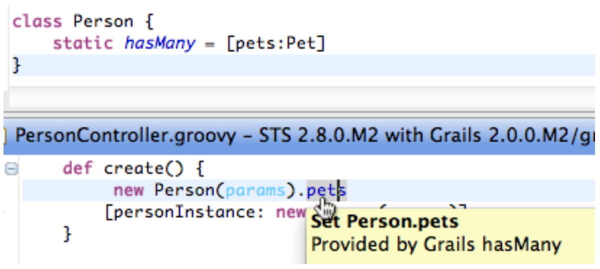
A refactoring participant performing the same kinds of changes is also triggered when a view (gsp file) is renamed.

## Inferencing and navigation improvements for Grails

### belongsTo, hasOne, hasMany inferencing support

References to fields implicitly defined in the belongsTo, hasOne, and hasMany domain class fields are now available to the editor in content assist and type inferencing.  This feature works best in Grails 2.0.0.M2 and later, but is still somewhat supported in 2.0.0.M1.

As an example, the Person domain class has an implicit Pets field, which is visible in the associated PersonController class:



```
class Person {
    static hasMany = [pets:Pet]
}
```

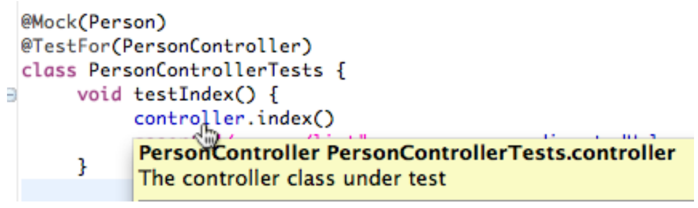
PersonController.groovy – STS 2.8.0.M2 with Grails 2.0.0.M2/g

```
    def create() {
        new Person(params).pets
        [personInstance: new
    }
```

Set Person.pets
Provided by Grails hasMany

### Unit Testing DSL (Grails 2.0.0.M2 or later)

Many parts of the new Grails unit testing DSL and idioms are available for content assist and inferencing.  For example, the type of the value inside the @TestFor annotation is used as the type of the controller field when testing controller classes:



```
@Mock(Person)
@TestFor(PersonController)
class PersonControllerTests {
    void testIndex() {
        controller.index()

    }
```

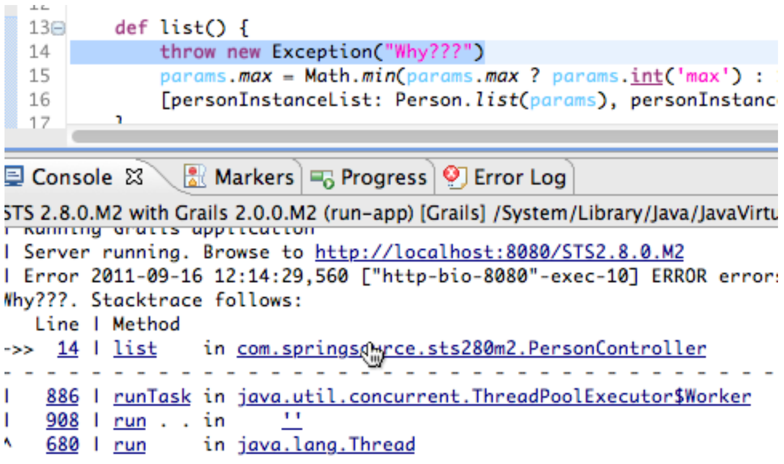
PersonController PersonControllerTests.controller
The controller class under test

## Grails 2.0 stack trace support

The new format for Grails stack traces, as described here:

http://grails.org/doc/2.0.0.M1/guide/introduction.html#developmentEnvironmentFeatures

is now correctly handled in the console. The references are hyperlinked and clicking on them will navigate you to the expected line number in the expected file:

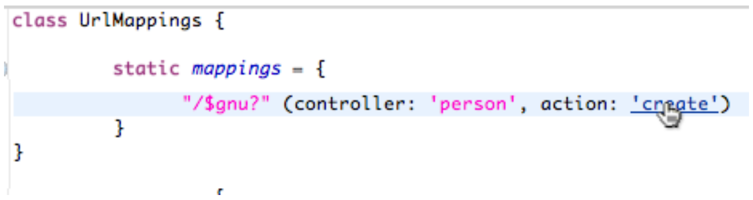


```
13      def list() {
14          throw new Exception("Why???")
15          params.max = Math.min(params.max ? params.int('max') :
16          [personInstanceList: Person.list(params), personInstanc
```

```
STS 2.8.0.M2 with Grails 2.0.0.M2 (run-app) [Grails] /System/Library/Java/JavaVirtu
| Server running. Browse to http://localhost:8080/STS2.8.0.M2
| Error 2011-09-16 12:14:29,560 ["http-bio-8080"-exec-10] ERROR error:
Why???. Stacktrace follows:
   Line | Method
->>   14 | list     in com.springsource.sts280m2.PersonController
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
|    886 | runTask  in java.util.concurrent.ThreadPoolExecutor$Worker
|    908 | run . .  in     ''
^    680 | run      in java.lang.Thread
```

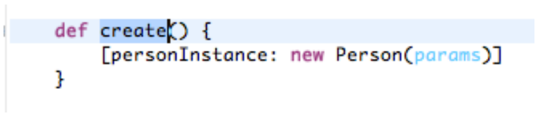
## Navigation support for URL mappings (STS-599)

Control-click navigation is now available in the UrlMappings.groovy file. It is possible to navigate from a mapping declaration to the view, controller, or action that it corresponds to. For example CTRL-Click (or CMD-Click on Macs) on create:



```
class UrlMappings {

        static mappings = {
                "/$gnu?" (controller: 'person', action: 'create')
        }
}
```

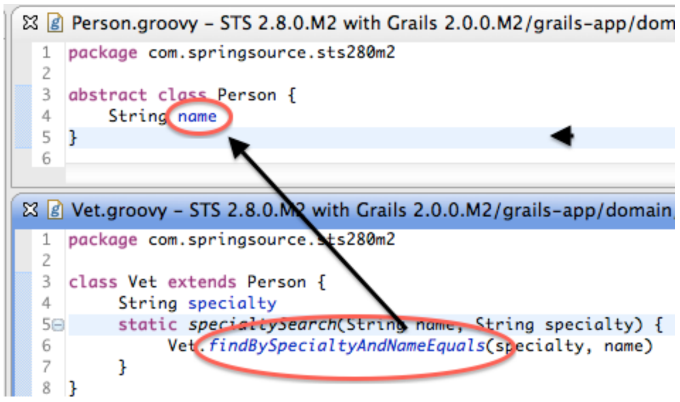
Will navigate you to the create action in the PersonController class:



```
    def create() {
        [personInstance: new Person(params)]
    }
```

### Better content assist for dynamic finders

Validation of Grails dynamic finders will now recognize domain properties coming from abstract domain classes. In the following example, notice how the dynamic finder refers to the name property of the Person domain class:



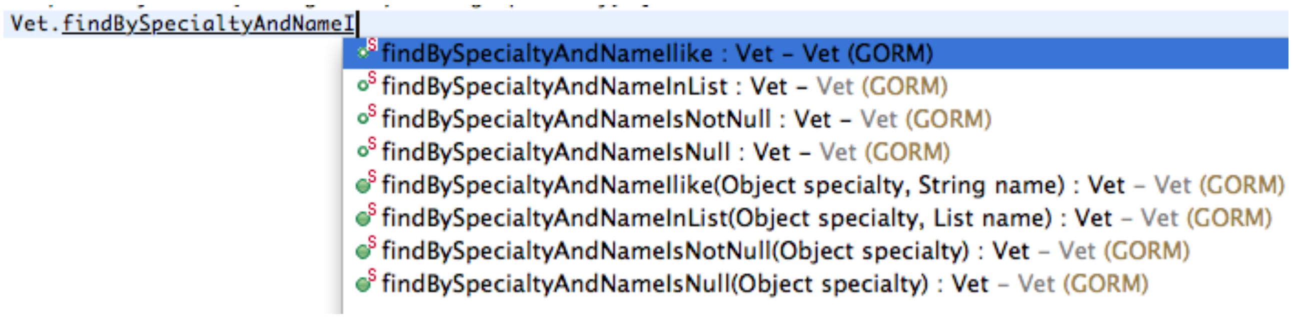
Person.groovy - STS 2.8.0.M2 with Grails 2.0.0.M2/grails-app/dom

```
package com.springsource.sts280m2

abstract class Person {
    String name
}
```

Vet.groovy - STS 2.8.0.M2 with Grails 2.0.0.M2/grails-app/domain

```
package com.springsource.sts280m2

class Vet extends Person {
    String specialty
    static specialtySearch(String name, String specialty) {
        Vet.findBySpecialtyAndNameEquals(specialty, name)
    }
}
```

Also, content assist for dynamic finders will now propose method completions where appropriate:

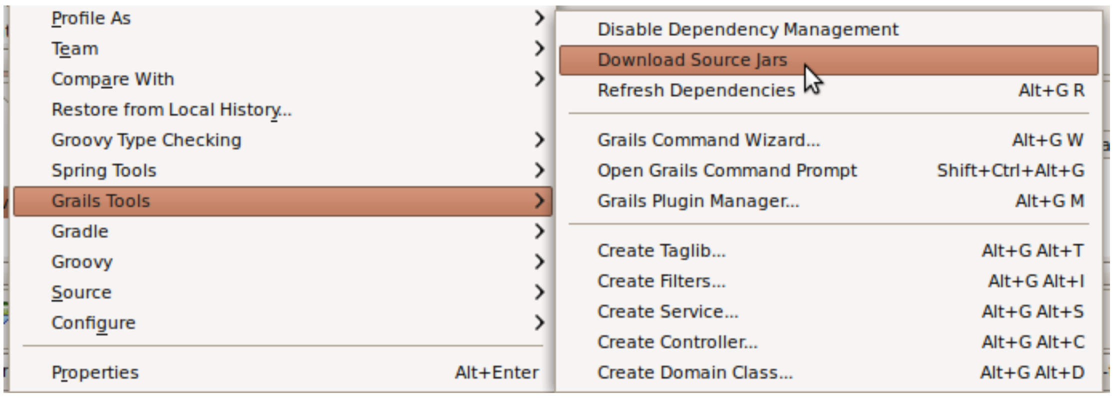


```
Vet.findBySpecialtyAndNameI
```

```
findBySpecialtyAndNameIlike : Vet – Vet (GORM)
findBySpecialtyAndNameInList : Vet – Vet (GORM)
findBySpecialtyAndNameIsNotNull : Vet – Vet (GORM)
findBySpecialtyAndNameIsNull : Vet – Vet (GORM)
findBySpecialtyAndNameIlike(Object specialty, String name) : Vet – Vet (GORM)
findBySpecialtyAndNameInList(Object specialty, List name) : Vet – Vet (GORM)
findBySpecialtyAndNameIsNotNull(Object specialty) : Vet – Vet (GORM)
findBySpecialtyAndNameIsNull(Object specialty) : Vet – Vet (GORM)
```

## Download Source Attachments

Grails 2.0.0.M2 and later provide a command that allows downloading source code for jars in 'Grails Dependencies' (works only if the source jars are indeed resolvable through ivy). STS makes use of this command to provide a 'Download source attachments' menu item.

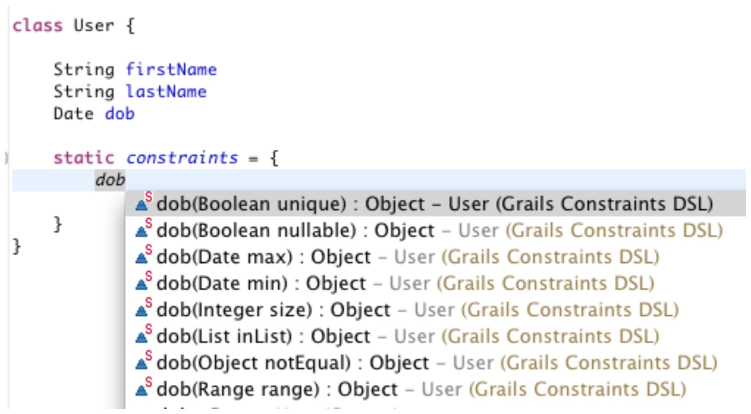

Profile As
Team
Compare With
Restore from Local History...
Groovy Type Checking
Spring Tools
Grails Tools
Gradle
Groovy
Source
Configure
Properties Alt+Enter
Disable Dependency Management
Download Source Jars
Refresh Dependencies Alt+G R
Grails Command Wizard... Alt+G W
Open Grails Command Prompt Shift+Ctrl+Alt+G
Grails Plugin Manager... Alt+G M
Create Taglib... Alt+G Alt+T
Create Filters... Alt+G Alt+I
Create Service... Alt+G Alt+S
Create Controller... Alt+G Alt+C
Create Domain Class... Alt+G Alt+D

## Enhanced DSL support for Grails 2.0.0

There is now a DSL descriptor shipped with Grails 2.0.0. This provides extra editing support when working with Grails artifacts. For example:

### Grails constraints DSL (STS-1184)

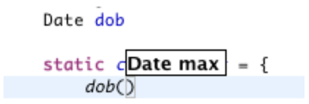
Full content assist for the standard constraints block:



```
class User {

    String firstName
    String lastName
    Date dob

    static constraints = {
        dob
    }
}
```

```
dob(Boolean unique) : Object – User (Grails Constraints DSL)
dob(Boolean nullable) : Object – User (Grails Constraints DSL)
dob(Date max) : Object – User (Grails Constraints DSL)
dob(Date min) : Object – User (Grails Constraints DSL)
dob(Integer size) : Object – User (Grails Constraints DSL)
dob(List inList) : Object – User (Grails Constraints DSL)
dob(Object notEqual) : Object – User (Grails Constraints DSL)
dob(Range range) : Object – User (Grails Constraints DSL)
```

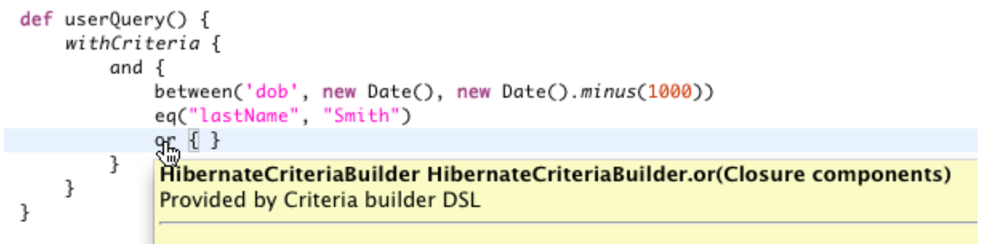
Invoking content assist inside of a constraint method, you can see the method context information displayed:



```
Date dob

static cDate max = {
    dob()
```

### GORM criteria query support (STS-580)

There is now editing support for the criteria query builder DSL in the Groovy Editor:

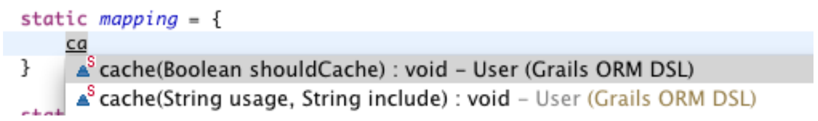


```
def userQuery() {
    withCriteria {
        and {
            between('dob', new Date(), new Date().minus(1000))
            eq("lastName", "Smith")
            or { }
        }
    }
}
```

```
HibernateCriteriaBuilder HibernateCriteriaBuilder.or(Closure components)
Provided by Criteria builder DSL
```

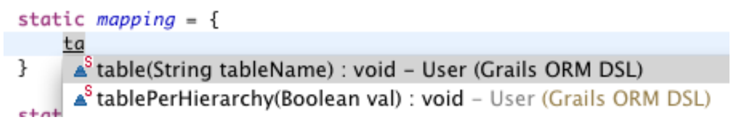
### Grails ORM DSL support

There is now full support for the Grails ORM DSL in the static mapping field of a domain class:



```
static mapping = {
    ca
}
```

```
cache(Boolean shouldCache) : void – User (Grails ORM DSL)
cache(String usage, String include) : void – User (Grails ORM DSL)
```



```
static mapping = {
    ta
}
```

table(String tableName) : void – User (Grails ORM DSL)
tablePerHierarchy(Boolean val) : void – User (Grails ORM DSL)

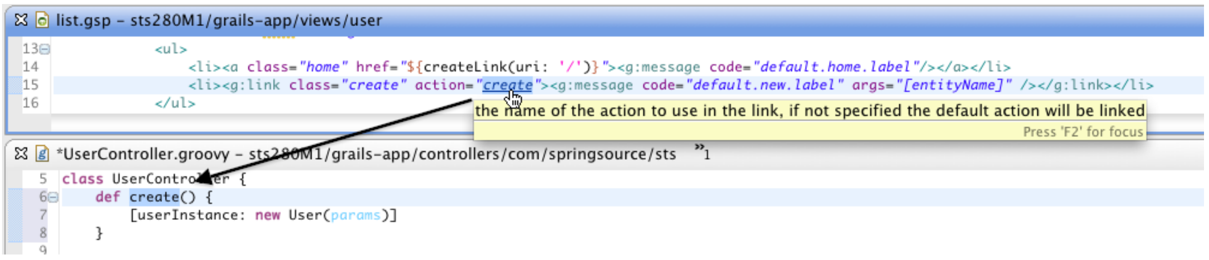
## Enhanced dynamic finder support (STS-1982)

STS now has more complete support for dynamic finders.

## GSP support (STS-1498)

### Navigation to referenced controller or action in g:link tag

Pressing F3 or Ctrl-hover over an action/controller in a g:link tag will navigate you to the definition of that action/controller:



list.gsp – sts280M1/grails-app/views/user

```
        <ul>
            <li><a class="home" href="${createLink(uri: '/')}"><g:message code="default.home.label"/></a></li>
            <li><g:link class="create" action="create"><g:message code="default.new.label" args="[entityName]" /></g:link></li>
        </ul>
```

the name of the action to use in the link, if not specified the default action will be linked

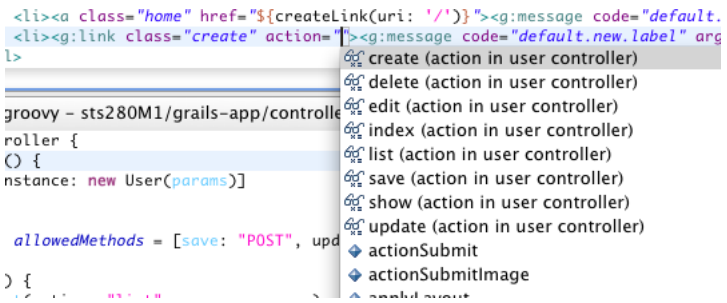
Press 'F2' for focus

*UserController.groovy – sts280M1/grails-app/controllers/com/springsource/sts

```
class UserController {
    def create() {
        [userInstance: new User(params)]
    }
```

### Content assist in g:link tags

There is now content assist for relevant controllers and actions in g:link tags:



```
<li><a class="home" href="${createLink(uri: '/')}"><g:message code="default.
<li><g:link class="create" action=""><g:message code="default.new.label" arg
```

create (action in user controller)
delete (action in user controller)
edit (action in user controller)
index (action in user controller)
list (action in user controller)
save (action in user controller)
show (action in user controller)
update (action in user controller)
actionSubmit
actionSubmitImage

# Gradle Tools

## Selected Gradle project import settings are now persisted with the root project

Certain key import wizard preferences are now persisted inside the Gradle project root .settings folder. The settings that are being persisted are:

- The list of projects imported into STS

- settings related to running specific tasks before/after importing.

The settings file is found under:

```
<root-project-folder>/.settings/com.springsource.sts.gradle.core.import.prefs
```

Though the file is not intended to be edited directly by the user, it should be safe to share among users via source control. The entries in the file are represented in such a way as to retain their meaning relative to the root of the project hierarchy.

Comments and feedback about this feature (specifically ideas from a user's perspective) about whether or not certain preferences should or should not be made shareable in this way are welcome and encouraged. Please raise an STS Jira issue or comment on https://issuetracker.springsource.com/browse/STS-2106 to provide your input.

### Handle common issues caused by running the 'eclipse' tasks before importing

The import wizard now handles many common problems that may occur because of pre-existing and conflicting eclipse configuration files. The wizard only touches those aspects of a project's configuration that are related to what it understands (java nature, jar dependencies, project dependencies, source folders...). Other configuration information, such as other project natures and their related settings are, as much as possible, left intact.

This makes it possible to correctly import project types beyond just the 'pure Java' projects that are supported directly by the current version of the Gradle Tooling API. The generated files, in many cases, can provide the missing configuration information that is needed to make the projects work.

### New options and default option tweaks

The wizard now provides options to run tasks before and/or after projects are imported into the workspace. The intention for these options is to allow the user to customize projects, for example, to make up for lacking support or bugs in the current version of the tooling API or the STS Tool support.

A common use case is to execute the 'cleanEclipse' and 'eclipse' tasks on the project before importing. This works well in combination with the fact that the wizard now is better at dealing with pre-existing Eclipse configuration files.

The before tasks will be executed on all projects that define them, before any projects are imported.

The after tasks will be executed on all projects that define them, after all projects have been imported.

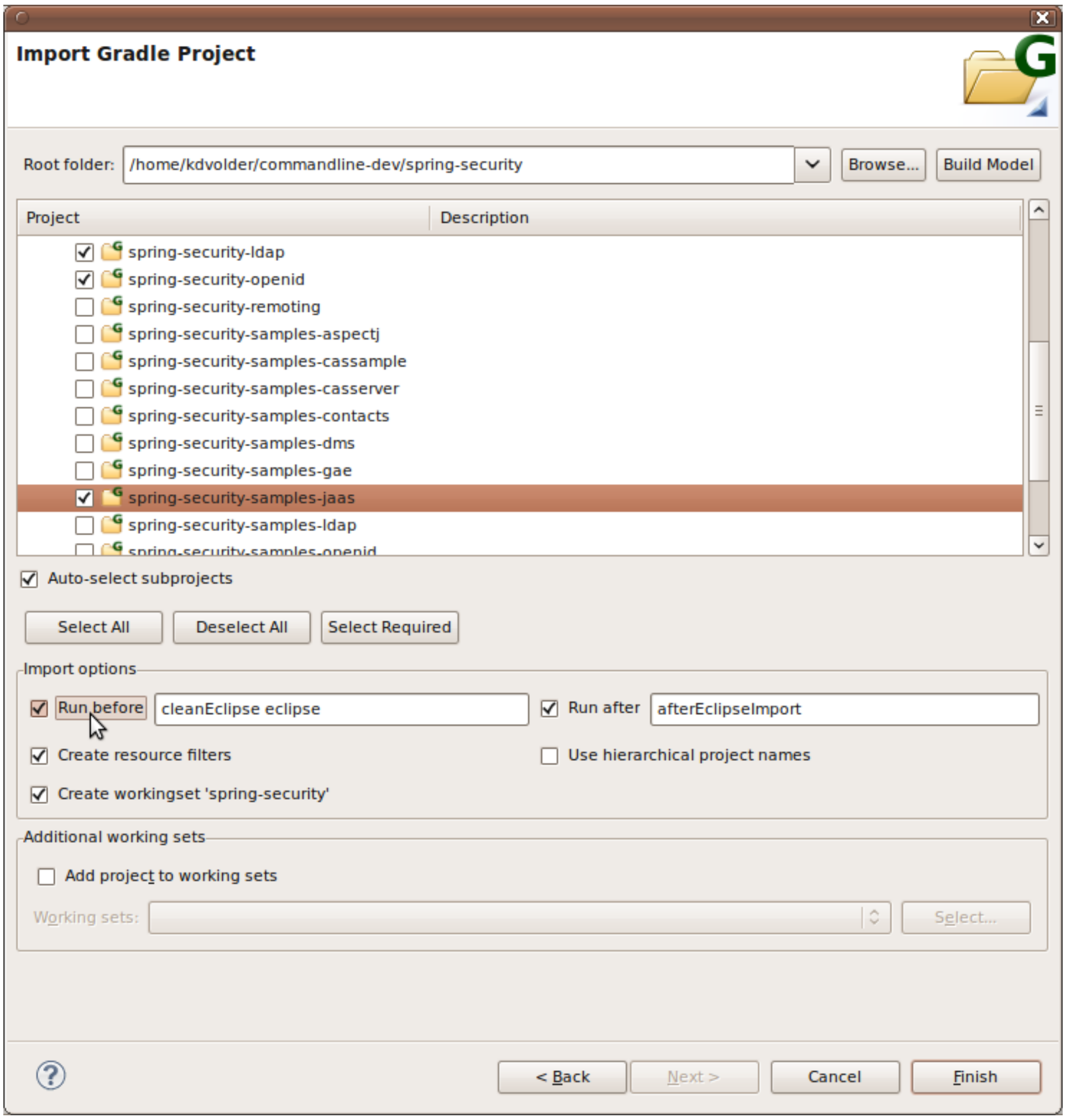
**Warning:** with Gradle M3 it is not possible for STS to determine whether the cleanEclipse and eclipse tasks are defined on a project. See GRADLE-1792 for details. Because of this, the option will be disabled by default for pre M4 projects. You can still use the option with an M3 build by explicitly enabling it. However, you must ensure that all the projects you want to import effectively have the eclipse tasks defined on them. Otherwise execution of the tasks will fail without any tasks being executed.

Note that the default after task, 'afterEclipseImport' is not a built-in Gradle eclipse task. It is purely intended as a hook for you to plugin any custom post processing you may want to do.

Other minor tweaks:

- 'Use Hierachical Project Names' is now deselected by default, because the 'flat' naming scheme seems to correspond more to what most people would expect and want.
- Projects in the tree-viewer are now sorted alphabetically.

The new import wizard looks like this:



**Import Gradle Project**

Root folder: /home/kdvolder/commandline-dev/spring-security

| Project | Description |
|---|---|
| ✓ spring-security-ldap | |
| ✓ spring-security-openid | |
| ☐ spring-security-remoting | |
| ☐ spring-security-samples-aspectj | |
| ☐ spring-security-samples-cassample | |
| ☐ spring-security-samples-casserver | |
| ☐ spring-security-samples-contacts | |
| ☐ spring-security-samples-dms | |
| ☐ spring-security-samples-gae | |
| ✓ spring-security-samples-jaas | |
| ☐ spring-security-samples-ldap | |
| ☐ spring-security-samples-openid | |

✓ Auto-select subprojects

Import options

✓ Run before: cleanEclipse eclipse
✓ Run after: afterEclipseImport
✓ Create resource filters
☐ Use hierarchical project names
✓ Create workingset 'spring-security'

Additional working sets

☐ Add project to working sets
Working sets:

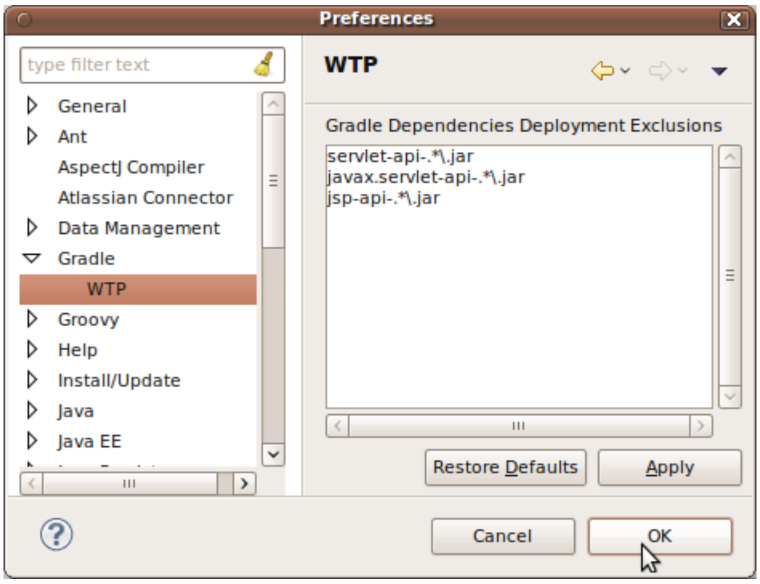
## Simple WTP Support

Although the Gradle Tooling API does not yet provide any support for WTP Web projects, it is possible now to get limited support in STS by relying on running the cleanEclipse and eclipse task before importing projects into STS.

When WTP projects are imported and the eclipse task has been executed beforehand, STS recognizes their WTP nature and performs some customizations specific to WTP projects:

- The classpath container containing the 'Gradle Dependencies' is added to the Deployment Assembly.
- Specific items in the classpath container that should not be deployed are recognized and excluded.

The set of items to be excluded can be configured from a preferences page: Preferences >> Gradle >> WTP :



Gradle Dependencies Deployment Exclusions

```
servlet-api-.*\.jar
javax.servlet-api-.*\.jar
jsp-api-.*\.jar
```

**Note 1:** The default setup avoids common problems that are known to cause failure to deploy a project.

**Note 2:** This is a temporary solution. A proper implementation would exclude dependencies from deployment based on Gradle configuration types (e.g. test and provided dependencies should be excluded). However, the needed information is not provided by the current version of the Gradle Tooling API (see issue GRADLE-1777). In the mean time, this list of exclusions will help to deploy more Gradle WTP projects successfully. Suggestions to add more patterns to the default exclusion list are welcome.

## Automatic fallback to M3

In some cases, a Gradle model build may fail because the tooling API is not able to determine correctly what version of Gradle it should be using for your project. When a model build fails, STS will propose to try using M3 which seems, presently, to be the most stable and widely used version of Gradle. If the option is accepted, this choice will be persisted as a workspace preference so that all further model builds will use M3. (To cancel this, use the Gradle Preferences page to switch back to using the "Gradle Wrapper's Default").

## AspectJ/AJDT

AJDT now includes a release candidate build of AspectJ 1.6.12. In this build two options have been turned to ON by default (minimalModel and typeDemotion). Previously these options defaulted to OFF. These options should lead to a reduction in memory footprint for AspectJ projects (e.g. Roo projects) in your workspace.

If you have any issues after upgrading, the settings can be turned off by specifying

```
-Xset:minimalModel=false,typeDemotion=false
```

in the project properties (in non-standard compiler options).

The release notes for AspectJ 1.6.12 are here:

http://www.eclipse.org/aspectj/doc/released/README-1612.html

## Fixed Bugs and Enhancement Requests

Here is a full list of resolved bugs and enhancement requests for the 2.8.0 release:

STS Issue-Tracker:
https://issuetracker.springsource.com/secure/IssueNavigator.jspa?reset=true&jqlQuery=project+%3D+STS+AND+fixVersion+in+%2811782%2C+11188%2C+11781%2C+11187%2C+11185%2C+11184%29+AND+status+in+%28Resolved%2C+Closed%29

Spring IDE Issue-Tracker:

https://jira.springsource.org/secure/IssueNavigator.jspa?reset=true&jqlQuery=project+%3D+IDE+AND+fixVersion+in+%2812615%2C+12216%2C+12614%2C+12215%2C+12213%2C+12212%29+AND+status+in+%28Resolved%2C+Closed%29