Advanced GATE Embedded: Using Groovy Module 5

Seventh GATE Training Course June 2014

© 2014 The University of Sheffield

This material is licenced under the Creative Commons

Attribution-NonCommercial-ShareAlike Licence

(http://creativecommons.org/licenses/by-nc-sa/3.0/)

< □ > < 同 > < 回 > < 回 > < 回 >



- Introduction to Groovy
- Scripting GATE Developer
- Groovy Scripting for PRs and Controllers
- Writing GATE Resource Classes in Groovy

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Outline

1 GATE and Groovy

- Introduction to Groovy
- Scripting GATE Developer
- Groovy Scripting for PRs and Controllers
- Writing GATE Resource Classes in Groovy

Groovy

- Dynamic language for the JVM
- Groovy scripts and classes compile to Java bytecode fully interoperable with Java.
- Syntax very close to regular Java
- Explicit types optional, semicolons optional
- Dynamic dispatch method calls dispatched based on runtime type rather than compile-time.
- Can add new methods to existing classes at runtime using metaclass mechanism
- Groovy adds useful extra methods to many standard classes in java.io, java.lang, etc.

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

def keyword declares an untyped variable

イロト 不得 トイヨト イヨト

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- def keyword declares an untyped variable
- but dynamic dispatch ensures the get call goes to the right class (AnnotationSet).

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- def keyword declares an untyped variable
- but dynamic dispatch ensures the get call goes to the right class (AnnotationSet).
- findAll and collect are methods added to Collection by
 Groovy

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- def keyword declares an untyped variable
- but dynamic dispatch ensures the get call goes to the right class (AnnotationSet).
- findAll and collect are methods added to Collection by
 Groovy
 - http://groovy.codehaus.org/groovy-jdk has the details.

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- def keyword declares an untyped variable
- but dynamic dispatch ensures the get call goes to the right class (AnnotationSet).
- findAll and collect are methods added to Collection by Groovy
 - http://groovy.codehaus.org/groovy-jdk has the details.
- is the safe navigation operator if the left hand operand is null it returns null rather than throwing an exception

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

■ =~ for regular expression matching

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- =~ for regular expression matching
- unified access to JavaBean properties it.startNode shorthand for it.getStartNode()

イロト 不得 トイヨト イヨト

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- =~ for regular expression matching
- unified access to JavaBean properties it.startNode shorthand for it.getStartNode()
- and Map entries anchor.features.href shorthand for anchor.getFeatures().get("href")

イロト 不得 トイヨト イヨト

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3 anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- =~ for regular expression matching
- unified access to JavaBean properties it.startNode shorthand for it.getStartNode()
- and Map entries anchor.features.href shorthand for anchor.getFeatures().get("href")

```
Map entries can also be accessed like arrays, e.g.
features ["href"]
```

Closures

Parameter to collect, findAll, etc. is a *closure*

- like an anonymous function (JavaScript), a block of code that can be assigned to a variable and called repeatedly.
- Can declare parameters (typed or untyped) between the opening brace and the ->
- If no explicit parameters, closure has an implicit parameter called it.
- Closures have access to the variables in their containing scope (unlike Java inner classes these do not have to be final).
- The return value of a closure is the value of its last expression (or an explicit return).
- Closures are used all over the place in Groovy

イロト 不得 トイヨト イヨト

More Groovy Syntax

- Shorthand for lists: ["item1", "item2"] declares an ArrayList
- Shorthand for maps: [foo:"bar"] creates a HashMap mapping the key "foo" to the value "bar".
- Interpolation in *double-quoted* strings (like Perl):

"There are \${anns.size()} annotations of type \${annType}"

- Parentheses for method calls are optional (where this is unambiguous): myList.add 0, "someString"
 - When you use parentheses, if the last parameter is a closure it can go outside them: this is a method call with two parameters someList.inject(0) { last, cur -> last + cur }
- "slashy string" syntax where backslashes don't need to be doubled: /C:\Program Files\Gate/ equivalent to

```
'C:\\Program Files\\Gate'
```

Operator Overloading

- Groovy supports operator overloading cleanly
- Every operator translates to a method call
 - x == y becomes x.equals(y) (for reference equality, use x.is(y))
 - x + y becomes x.plus(y)
 - x << y becomes x.leftShift(y)
 - full list at http://groovy.codehaus.org
- To overload an operator for your own class, just implement the method.
- e.g. List implements leftShift to append items to the list:

['a', 'b'] << 'c'== ['a', 'b', 'c']

Groovy in GATE

- Groovy support in GATE is provided by the Groovy plugin.
- Loading the plugin
 - enables the Groovy scripting console in GATE Developer
 - adds utility methods to various GATE classes and interfaces for use from Groovy code
 - provides a PR to run a Groovy script.
 - provides a scriptable controller whose execution strategy is determined by a Groovy script.

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Scripting GATE Developer

- Groovy provides a Swing-based console to test out small snippets of code.
- The console is available in the GATE Developer GUI via the Tools menu. To enable, load the Groovy plugin.



Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Imports and Predefined Variables

The GATE Groovy console imports the same packages as JAPE RHS actions:

gate, gate.annotation, gate.util, gate.jape and gate.creole.ontology

The following variables are implicitly defined:

corpora a list of loaded corpora LRs (Corpus)

docs a list of all loaded document LRs (DocumentImpl)

prs a list of all loaded PRs

apps a list of all loaded Applications (AbstractController)

Exercise 1: The Groovy Console

- Start the GATE Developer GUI
- Load the Groovy plugin
- $\blacksquare \ Select \ {\sf Tools} \to {\sf Groovy} \ {\sf Tools} \to {\sf Groovy} \ {\sf Console}$
- Experiment with the console
- For example to tokenise a document and find how many "number" tokens it contains:

```
1 doc = Factory.newDocument(new URL('http://gate.ac.uk'))
2 tokeniser = Factory.createResource('gate.creole.tokeniser.
DefaultTokeniser')
3 tokeniser.document = doc
4 tokeniser.execute()
5 tokens = doc.annotations.get('Token')
6 tokens.findAll { it.features.kind == 'number' }.size()
```

Exercise 1: The Groovy Console

- Variables you assign in the console (without a def or a type declaration) remain available to future scripts in the same console.
- So you can run the previous example, then try more things with the doc and tokens variables.
- Some things to try:
 - Find the names and sizes of all the annotation sets on the document (there will probably only be one named set).
 - List all the different kinds of token
 - Find the longest word in the document

Exercise 1: Solution

Some possible solutions (there are many...)

```
1 // Find the annotation set names and sizes
2 doc.namedAnnotationSets.each { name, set ->
    println "${name} has size ${set.size()}"
3
4
  }
5
  // List the different kinds of token
6
  tokens.collect { it.features.kind }.unique()
7
8
  // Find the longest word
9
10 tokens.findAll {
    it.features.kind == 'word'
11
12 }.max { it.features.length.toInteger() }
```

Groovy Categories

- In Groovy, a class declaring static methods can be used as a category to inject methods into existing types (including interfaces)
- A static method in the category class whose first parameter is a Document:

public static SomeType foo(Document d, String arg)

- ... becomes an instance method of the Document class:
 public SomeType foo(String arg)
- The use keyword activates a category for a single block
- To enable the category globally:

```
TargetClass.mixin(CategoryClass)
```

Utility Methods

- The gate.Utils class (mentioned in the JAPE module) contains utility methods for documents, annotations, etc.
- Loading the Groovy plugin treats this class as a category and installs it as a global mixin.
- Enables syntax like:

```
1 tokens.findAll {
2 it.features.kind == 'number'
3 }.each {
4 println "${it.type}: length = ${it.length()}, "
5 println " string = ${doc.stringFor(it)}"
6 }
```

Utility Methods

- The Groovy plugin also mixes in the GateGroovyMethods class.
- This extends common Groovy idioms to GATE classes
 - e.g. implements each, eachWithIndex and collect for Corpus to do the right thing when the corpus is stored in a datastore
 - defines a withResource method on Resource, to call a closure with a given resource as a parameter, and ensure the resource is deleted when the closure returns:

```
1 Factory.newDocument(someURL).withResource { doc ->
2 // do something with the document
3 }
```

Utility Methods

Also overloads the subscript operator [] to allow:

- annSet["Token"] and annSet["Person", "Location"]
- annSet [15..20] to get annotations within given span
- doc.content[15..20] to get the DocumentContent within a given span
- See src/gate/groovy/GateGroovyMethods.java in the Groovy plugin for details.

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Exercise 2: Using a category

In the console, try using some of these new methods:

```
1 tokens = doc.annotations["Token"]
2 tokens.findAll {
3 it.features.kind == 'number'
4 }.each {
5 println "${it.type}: length = ${it.length()}, "
6 println " string = ${doc.stringFor(it)}"
7 }
```

The Groovy Script PR

- The Groovy plugin provides a PR to execute a Groovy script.
- Useful for quick prototyping, or tasks that can't be done by JAPE but don't warrant writing a custom PR.
- PR takes the following parameters:

scriptURL (init-time) The path to a valid Groovy script

inputASName an optional annotation set intended to be used as input by the PR

outputASName an optional annotation set intended to be used as output by the PR

scriptParams optional parameters for the script as a FeatureMap

Script Variables

The script has the following implicit variables available when it is run

doc the current document

corpus the corpus containing the current document

content the string content of the current document

inputAS the annotation set specified by inputASName in the PRs runtime parameters

- outputAS the annotation set specified by outputASName in the PRs runtime parameters
- scriptParams the parameters FeatureMap passed as a runtime parameter

and the same implicit imports as the console.

Corpus-level processing

- Any other variables are treated like instance variables in a PR values set while processing one document are available while processing the next.
- So Groovy script is stateful, can e.g. collect statistics from all the documents in a corpus.
- Script can declare methods for pre- and post-processing:
 - beforeCorpus called before first document is processed.
 - afterCorpus called after last document is processed
 - aborted called if anything goes wrong
- All three take the corpus as a parameter
- scriptParams available within methods, other variables not.

• □ • • @ • • Ξ • • Ξ • ·

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Controller Callbacks Example

Count the number of annotations of a particular type across the corpus

```
1 void beforeCorpus(c) {
    println "Processing corpus ${c.name}"
2
    count = 0
3
4
  }
5
 count += doc.annotations[scriptParams.type].size()
6
7
8 void afterCorpus(c) {
    println "Total ${scriptParams.type} annotations " +
9
        "in corpus ${c.name}: ${count}"
10
11
 }
```

Exercise 3: Using the Script PR

- Write a very simple Goldfish annotator as a Groovy script
 - Annotate all occurrences of the word "goldfish" (case-insensitive) in the input document as the annotation type "Goldfish".
 - Add a "numFish" feature to each Sentence annotation giving the number of Goldfish annotations that the sentence contains.
- Put your script in the file

hands-on/groovy/goldfish.groovy

- To test, load hands-on/groovy/goldfish-app.xgapp into GATE Developer (this application contains tokeniser, sentence splitter and goldfish script PR).
- You need to re-initialize the Groovy Script PR after each edit to goldfish.groovy

Exercise 3: Solution

One of many possible solutions:

```
1 def m = (content =~ /(?i)goldfish/)
2 while (m.find()) {
    outputAS.add((long)m.start(), (long)m.end(),
3
        'Goldfish', [:].toFeatureMap())
4
5
  }
6
7 def allGoldfish = outputAS["Goldfish"]
 inputAS["Sentence"].each { sent ->
8
    sent.features.numFish =
9
        allGoldfish[sent.start()..sent.end()].size()
10
11 }
```

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

The Scriptable Controller

- ConditionalSerialAnalyserController can run PRs conditionally based on the value of a document feature.
- This is useful but limited; Groovy plugin's scriptable controller provides more flexibility.
- Uses Groovy DSL to define the execution strategy.

The ScriptableController DSL

- Run a single PR by using its *name* as a method call
 - So good idea to give your PRs identifier-friendly names.
- Iterate over the documents in the corpus using eachDocument
- Within an eachDocument closure, any PRs that implement LanguageAnalyser get their document and corpus parameters set appropriately.
- Override runtime parameters by passing named arguments to the PR method call.
- DSL is a Groovy script, so all Groovy language features available (conditionals, loops, method declarations, local variables, etc.).

http://gate.ac.uk/userguide/sec:api:groovy: controller

ScriptableController example

```
eachDocument
1
    documentReset()
2
    tokeniser()
3
    gazetteer()
4
    splitter()
5
    posTagger()
6
    findLocations()
7
     // choose the appropriate classifier depending how many Locations were found
8
    if(doc.annotations["Location"].size() > 100) {
9
       fastLocationClassifier()
10
11
    else {
12
       fullLocationClassifier()
13
14
15
  }
```

• □ • • @ • • Ξ • • Ξ • ·

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

ScriptableController example

```
eachDocument {
1
     // find all the annotatorN sets on this document
2
    def annotators =
3
       doc.annotationSetNames.findAll {
4
         it ==~ /annotator\d+/
5
6
7
     // run the post-processing JAPE grammar on each one
8
    annotators.each { asName ->
9
       postProcessingGrammar(
10
            inputASName: asName,
11
            outputASName: asName)
12
13
14
     // merge them to form a consensus set
15
    mergingPR(annSetsForMerging: annotators.join(';'))
16
17
```

Robustness and Realtime Features

- When processing large corpora, applications need to be robust.
 - If processing of a single document fails it should not abort processing of the whole corpus.
- When processing mixed corpora or using complex grammars, most documents process quickly but a few may take much longer.
 - Option to interrupt/terminate processing of a document when it takes too long.
 - Particularly useful with pay-per-hour processing such as GATECloud.net

Ignoring Errors

Use an ignoringErrors block to ignore any exceptions thrown in the block.

```
1 eachDocument {
2 ignoringErrors {
3 myTransducer()
4 }
5 }
```

- Exceptions thrown will be logged but will not terminate execution.
- Note nesting
 - ignoringErrors inside eachDocument exception means move to next document.
 - eachDocument inside ignoringErrors exception would terminate processing of corpus.

Limiting Execution Time

Use a timeLimit block to place a limit on the running time of the given block.

```
1 eachDocument {
2 annotateLocations()
3 timeLimit(soft:30.seconds, hard:30.seconds) {
4 classifyLocations()
5 }
6 }
```

- *soft* limit interrupt the running thread and PR
- hard limit Thread.stop()
- Limits are cumulative hard limit starts counting from the expiry of the soft limit.

Limiting Execution Time (2)

- When a block is terminated due to reaching a hard time limit, this generates an exception.
 - So in GATE Developer you probably want to wrap the timeLimit block in an ignoringErrors so it doesn't fail the corpus.
 - But on GATECloud.net each document is processed separately, so you *do* want the exception thrown to mark the offending document as failed.
- Treat timeLimit as a last resort use heuristics to try and avoid long-running PRs (see the "fast" vs. "full" location classifier example).

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Writing Resources in Groovy

- Groovy is more than a scripting language you can write classes (including GATE resources such as ScriptableController) in Groovy and compile them to Java bytecode.
- Compiler available via <groovyc> Ant task in groovy-all JAR.
- In order to use GATE resources written in Groovy (other than those that are part of the Groovy plugin), groovy-all JAR file must go into gate/lib.

Groovy Beans

Recall unified Java Bean property access in Groovy

- x = it.someProp means x = it.getSomeProp()
- it.someProp = x means it.setSomeProp(x)
- Declarations have a similar shorthand: a field declaration with no public, protected or private modifier becomes a private field plus an auto-generated public getter/setter pair.
- But you can provide explicit setter or getter, which will be used instead of the automatic one.
 - Need to do this if you need to annotate the setter (e.g. as a CreoleParameter).
 - Declare the setter private to get a read-only property (but not if it's a creole parameter).

Introduction to Groovy Scripting GATE Developer Groovy Scripting for PRs and Controllers Writing GATE Resource Classes in Groovy

Example: a Groovy Regex PR

```
package gate.groovv.example
1
2
  import gate.*
3
  import gate.creole.*
4
5
  public class RegexPR extends AbstractLanguageAnalyser {
6
7
    String regex
    String annType
8
    String annotationSetName
9
10
    public void execute() {
11
      def aSet = document.getAnnotations(annotationSetName)
12
      def matcher = (document.content.toString() =~ regex)
13
      while(matcher.find())
14
15
         aSet.add(matcher.start(), matcher.end(),
16
             annType, [:].toFeatureMap())
17
18
19
```

イロト イポト イヨト イヨト

э