

# Creating new Resource Types

## Module 8

Twelfth GATE Training Course  
June 2019

© 2019 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/>)

# Outline

- 1** CREOLE Metadata
  - CREOLE Recap
- 2** Creating CREOLE Resources
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource
  - Ready Made Applications
- 3** Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI

# Outline

- 1** CREOLE Metadata
  - CREOLE Recap
- 2** Creating CREOLE Resources
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource
  - Ready Made Applications
- 3** Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI

# CREOLE

The GATE component model is called CREOLE (**C**ollection of **RE**usable **O**bjects for **L**anguage **E**ngineering).

CREOLE uses the following terminology:

- **CREOLE Plugins:** contain definitions for a set of resources.
- **CREOLE Resources:** Java objects with associated configuration.
- **CREOLE Configuration:** the metadata associated with Java classes that implement CREOLE resources.

# CREOLE Plugins

CREOLE is organised as a set of plugins.

Each CREOLE plugin:

- is identified by a “loction” at which it can be found
- contains a special file called `creole.xml`;
- contains the definitions for a set of CREOLE resources.

# CREOLE Plugins

A plugin can be located by...

- a directory URL (the old way)
- Maven coordinates (the new way)
- custom (outside scope of this talk)

# CREOLE Plugins

A typical `creole.xml`:

```
1 <CREOLE-DIRECTORY>
2     <REQUIRES
3         GROUP="uk.ac.gate.plugins"
4         ARTIFACT="annie"
5         VERSION="8.6"/>
6 </CREOLE-DIRECTORY>
```

## CREOLE Resources

A CREOLE resource is a Java Bean with some additional metadata.

### A CREOLE resource class:

- must implement the `gate.Resource` interface (or one of its sub-interfaces `LanguageResource`, `ProcessingResource`, `VisualResource` or `Controller`);
- must be annotated with `@CreoleResource` to identify it as a resource class;
- must provide accessor methods for its parameters.



## CREOLE Annotations: @CreoleResource

The `@CreoleResource` annotation takes attributes giving metadata:

- name** (String) the name of the resource.
- comment** (String) a descriptive comment about the resource
- helpURL** (String) a URL to a help document on the web for this resource.
- icon** (String) the icon to use to represent the resource in GATE Developer.

### Example

```
1 @CreoleResource (name = "Document Stats",  
2                 comment = "Calculates document statistics.")  
3 public class DocStats extends AbstractLanguageAnalyser {  
4     ...  
5 }
```

## CREOLE Annotations: @CreoleResource

### Attributes for Visual Resources

If the resource being configured is a Visual Resource, you can also use the following attributes:

**guiType** (GuiType enum) the type of GUI this resource defines.

**resourceDisplayed** (String) the class name of the resource type that this VR displays, e.g. "gate.Corporus".

**mainViewer** (boolean) is this VR the *most important* viewer for its displayed resource type?

# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources
  - Your First Language Analyser
    - Best Practice
    - Your First Visual Resource
    - Ready Made Applications
- 3 Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI

## Exercise 1: Create an Empty Processing Resource

### Create a new GATE plugin containing a sample PR

```
1 mvn archetype:generate \  
2     -DarchetypeGroupId=uk.ac.gate \  
3     -DarchetypeArtifactId=gate-pr-archetype \  
4     -DarchetypeVersion=8.6
```

- follow the on screen prompts
- make sure it compiles; `mvn install`
- make sure you can load it into GATE
  - use the group ID, artifact ID, and version number you specified to add it to the plugin manager

## Exercise 1 (part 2): Implementation

Implement:

```
1 public Resource init()  
2   throws ResourceInstantiationException { }
```

... to print out a message;

Implement:

```
1 public void execute() throws ExecutionException { }
```

... to count the number of Token annotations in the input document,  
and set the value as a feature on the document.

## Exercise 1: Solution

Try not to use this!

## Exercise 1: Solution

Try not to use this!

```
1 package module8;
2
3 import gate.Resource;
4 import gate.creole.*;
5 import gate.creole.metadata.*;
6
7 @CreoleResource
8 public class DocStats extends AbstractLanguageAnalyser {
9
10     @Override
11     public void execute() throws ExecutionException {
12         int tokens = document.getAnnotations().get("Token").size();
13         document.getFeatures().put("token_count", tokens);
14     }
15
16     @Override
17     public Resource init() throws ResourceInstantiationException {
18         System.out.println(getClass().getName() + " is initialising.");
19         return this;
20     }
21 }
```

# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources
  - Your First Language Analyser
  - **Best Practice**
  - Your First Visual Resource
  - Ready Made Applications
- 3 Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI



## Best Practice: Use Parameters!

- Do not hardcode values, specify them as parameters.
- Values that change internal data structures, built when the PR is created, should be `init-time` parameters. These cannot be changed once the PR was created.
- Values that can be changed between executions should be `run-time` parameters.
- Try to make as many parameters as possible into `run-time` parameters!
- Provide *sensible defaults* for most parameters.
- If you have too many `init-time` parameters, use a config file instead!
- If you have too many `run-time` parameters, provide a Visual Resource!
- Make sure the parameters are well documented!

## Best Practice: Input/Output

### Specify Input/Output!

- If your PR uses annotations, always specify input and output annotation sets:
- use a parameter `inputASName` for the input annotation set name;
- use a parameter `outputASName` for the output annotation set name;

### OR

- use a parameter named `annotationSetName` (if the PR only modifies existing annotations).

## Defining Parameters

Creole parameters are Java Bean properties (a pair of get/set methods), with `@CreoleParameter` annotations on the **setter** method. Main attributes include:

**comment** (String) an optional descriptive comment about the parameter.

**defaultValue** (String) the optional default value for this parameter.

**suffixes** (String) for URL/ResourceReference-valued parameters, a semicolon-separated list of default file suffixes that this parameter accepts.

### Example

```
1
2 @CreoleParameter(
3     comment="The name for the input annotation set.")
4 public void setInputASName(String inputASName) {
5     this.inputASName = inputASName;
```

## CREOLE Annotations: Parameter Types

You can also use the following annotations to mark the type of a CREOLE parameter:

**@Optional** for parameters that are not required.

**@RunTime** for run-time parameters.

### Corrected Example

```
1  @Optional
2  @RunTime
3  @CreoleParameter(
4      comment="The name for the input annotation set.")
5  public void setInputASName(String inputASName) {
6      this.inputASName = inputASName;
7  }
```

**TIP:** More info at <http://gate.ac.uk/userguide/sec:creole-model:config>

## Exercise 2: Develop/Test Cycle

Change the implementation from *Exercise 1* to:

- add proper metadata on the resource class;
- use a parameter for the input annotation set;
- use a parameter for the `Token` annotation type;
- make sure these parameters have good defaults, and documentation;

### Test it!

- Start GATE Developer, load a document, create an instance of the Unicode Tokeniser;
- load your CREOLE plugin, create an instance of your PR; create a Corpus Pipeline and add the two PRs to it;
- run the pipeline over the document and check it works.

## Exercise 3: Better Statistics

Change the implementation from *Exercise 2* to also calculate counts for all **words**, all **nouns**, all **verbs**.

### TIPs:

You will need to run a Sentence Splitter, and POS Tagger after the Tokeniser, in order to get the part-of-speech information.

Definitions:

**word** {Token.kind=="word"}

**noun** POS tag starts with "NN"

**verb** POS tag starts with "VB"

POS tag is `tok.getFeatures().get("category")`

Test it!

# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources**
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource**
  - Ready Made Applications
- 3 Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI

## Visual Resources

- Visual Resources provide UI elements (Swing components) for building user interfaces.
- They are classes that implement the `gate.VisualResource` interface.
- They are associated with a type of resource via CREOLE metadata (which is used as a model for the view represented by the VR).
- The abstract class `gate.creole.AbstractVisualResource` can be used a starting point.



# Visual Resource API

## Visual Resource API

Visual resources extend the `gate.Resource` interface, with :

```
1 /** set the object to be displayed */  
2 public void setTarget(Object target);
```

## AbstractVisualResource

- **extends** `javax.swing.JPanel`;
- implements all the methods required by `gate.Resource`;
- extending classes only need to implement:
  - **public** `Resource` `init()`: initialise the resource (i.e. build the required UI elements);
  - **public void** `setTarget(Object target)`: sets the model for this view.

## Visual Resource CREOLE Metadata

- A Visual Resource is associated with a given type of object that it can display (or edit, configure). This association is done via CREOLE metadata on the VR implementation.
- From the API, the VR is populated by calling `setTarget(Object target)`.
- In GATE Developer, the appropriate VR types are instantiated on demand when a resource is double-clicked in the tree. E.g., when a Document is double-clicked, all VR registered as capable of displaying `gate.Document` targets are instantiated.

## VR Metadata Example

### CREOLE Annotations:

```
1 @CreoleResource (name="Statistics Viewer",  
2     comment="Shows document statistics",  
3     resourceDisplayed="gate.Document",  
4     guiType=GuiType.LARGE,  
5     mainViewer=true)  
6 public class StatsViewer extends AbstractVisualResource
```

## Exercise 4: Show the Statistics

- Create a VR that, given a document, can show the statistics produced by the DocStats language analyser.
- add CREOLE metadata to associate the new VR with the interface `gate.Document`;

You can use a simple `JTextPane` to show a `.toString()` value for the document's features.

## Exercise 4: Solution (metadata omitted)

Try not to use this!

## Exercise 4: Solution (metadata omitted)

### Try not to use this!

```
1 package module8;
2 import javax.swing.*;
3 import gate.*;
4 import gate.creole.*;
5 import gate.event.FeatureMapListener;
6 @CreoleResource(name="Statistics Viewer", resourceDisplayed="gate.Document",
7     guiType=GuiType.LARGE, mainViewer=true)
8 public class StatsViewer extends AbstractVisualResource
9     implements FeatureMapListener{
10     private JTextPane textPane;
11     private FeatureMap targetFeatures;
12     public Resource init() throws ResourceInstantiationException {
13         textPane = new JTextPane();
14         add(new JScrollPane(textPane));
15         return this;
16     }
17     public void setTarget(Object target) {
18         if(targetFeatures != null) targetFeatures.removeFeatureMapListener(this);
19         targetFeatures = ((Document)target).getFeatures();
20         targetFeatures.addFeatureMapListener(this);
21         featureMapUpdated();
22     }
23     public void featureMapUpdated() {
24         textPane.setText(targetFeatures.toString());
25     }
26 }
```

# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource
  - **Ready Made Applications**
- 3 Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI

## Ready Made Applications

- Many CREOLE plugins contain one or more example applications
  - they may be used to show how the processing resources can be used
  - some plugins might only contain applications, i.e. the language plugins
- Making these applications easily available through the GUI will make your processing resources easier for others to use
- Example applications can easily be added to the *Ready Made Applications* menu by creating an instance of `gate.creole.PackagedController`



## Packaged Controller API

- Packaged Controllers extend the

`gate.creole.PackagedController` class

```
1 /** the location of the pipeline XGAPP file */  
2 public ResourceReference getPipelineURL();  
3  
4 /** the menu under which the application appears */  
5 public List<String> getMenu();
```

- `gate.creole.PackagedController` is also a GATE resource so we can provide these values using CREOLE annotations

## Example: Chinese IE

```
1 package chinese;
2
3 import gate.creole.PackagedController;
4 import gate.creole.metadata.AutoInstance;
5 import gate.creole.metadata.AutoInstanceParam;
6 import gate.creole.metadata.CreoleParameter;
7 import gate.creole.metadata.CreoleResource;
8
9 import java.net.URL;
10 import java.util.List;
11
12 @CreoleResource(name = "Chinese IE System", icon = "ChineseLanguage",
13     autoinstances = @AutoInstance(parameters = {
14         @AutoInstanceParam(name="pipelineURL", value="resources/chinese.gapp"),
15         @AutoInstanceParam(name="menu", value="Chinese")}))
16 public class ChineseIE extends PackagedController {
17     //all without writing any code!
18 }
```

## Exercise 5: Show Off Your New Plugin

- create, and save, an application that shows how to use your statistics PR
- create a `gate.creole.PackagedController` instance to make the application available through the GUI.

You can use the Chinese IE example as a starting point.

# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource
  - Ready Made Applications
- 3 **Advanced CREOLE**
  - **CREOLE Management**
  - Corpus-level processing
  - Adding actions to the GUI

# The CREOLE and DataStore Registers

## The CREOLE Register

- Stores all CREOLE data, including:
  - which plugins are loaded;
  - which types of CREOLE Resources have been defined;
  - loaded instances of each resource type;
  - which Visual Resources can display any resource type;
- fires events when resources are loaded and deleted;
- forwards all events from the DataStore Register (see below).

## The DataStore Register

- is a `java.util.Set` of DataStore objects.
- fires events when datastores are created, opened and closed.

## CREOLE Register and its Events

```
1 // Obtain a pointer to the CREOLE Register
2 CreoleRegister cReg = Gate.getCreoleRegister();
3 // listen to CREOLE events
4 cReg.addCreoleListener(new CreoleListener() {
5     public void resourceUnloaded(CreoleEvent e) { ... }
6     public void resourceRenamed(Resource resource,
7         String oldName, String newName) { ... }
8     public void resourceLoaded(CreoleEvent e) { ... }
9     public void datastoreOpened(CreoleEvent e) { ... }
10    public void datastoreCreated(CreoleEvent e) { ... }
11    public void datastoreClosed(CreoleEvent e) { ... }
12 });
13 // remove a registered listener
14 cReg.removeCreoleListener(aListener);
```

## Other CREOLE APIs

### Plugins Management

```
1 try {
2     //load a Maven based plugin
3     cReg.registerPlugin(new Plugin.Maven(...));
4     // register a single resource class without using creole.xml
5     cReg.registerPlugin(
6         new Plugin.Component(MyResource.class));
7 } catch(GateException e1) { ... }
8 //get all loaded plugins
9 cReg.getPlugins();
10 //remove a loaded plugin
11 cReg.unregisterPlugin(...);
12 // register to get notified when plugins are loaded or unloaded
13 cReg.addPluginListener(...);
```

## Other CREOLE APIs (continued)

### Find Loaded Resources

```
1 // find all resources of a given type
2 try {
3     cReg.getAllInstances("gate.LanguageAnalyser");
4 } catch (GateException e1) { ... }
```

### Resource Types

```
1 cReg.getPrTypes(); // get PR types (class names)
2 cReg.getLrTypes(); // get LR types (class names)
3 cReg.getVrTypes(); // get VR types (class names)
```



## Other CREOLE APIs (continued)

### CREOLE Metadata

```
1 // Obtain the Resource Data about a resource
2 ResourceData rData = cReg.get("resource.class.name");
3 // get the list of instances
4 List<Resource> instances = rData.getInstantiations();
5 // get the list of parameters
6 ParameterList pList = rData.getParameterList();
7 // get the Init-time / Run-time parameters
8 List<List<Parameter>> someParams;
9 someParams = pList.getRuntimeParameters();
10 someParams = pList.getInitimeParameters();
```

## Exercise 6: CREOLE Metadata

- load the ANNIE application;
- find out which plugins are loaded;
- find out which PR **instances** exist;
- find out which PR types are known to the system;
- find out what parameters they have.

You may find this useful:

```
1 Gate.init();
2 Plugin annie = new Plugin.Maven(
3     "uk.ac.gate.plugins", "annie", "8.6");
4 Gate.getCreoleRegister().registerPlugin(annie);
5
6 ResourceReference xgapp = new ResourceReference(
7     annie, "resources/ANNIE_with_defaults.gapp");
8
9 PersistenceManager.loadObjectFromUri(xgapp.toURI());
```



# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource
  - Ready Made Applications
- 3 Advanced CREOLE
  - CREOLE Management
  - **Corpus-level processing**
  - Adding actions to the GUI

## Corpus-level processing

- When running a PR over a corpus of more than one document, you may want to do some additional pre- and post-processing before the first and after the last document.
- To do this, implement `gate.creole.ControllerAwarePR`
- Three callback methods called at key points in the execution of the *controller that contains the PR*:
  - `controllerExecutionStarted`
  - `controllerExecutionFinished`
  - `controllerExecutionAborted`
- Parameter is the `Controller`.
- “aborted” callback also receives the `Throwable` that caused the abort.

## Corpus-level processing

- So if the controller is a `CorpusController`, these correspond to:
  - before the first document
  - after the last document
  - when something goes wrong

## ControllerAwarePR example

```
1 @CreoleResource(name="Example Analyser")
2 public class ExampleAnalyser
3     extends AbstractLanguageAnalyser
4     implements ControllerAwarePR {
5     public void controllerExecutionStarted(Controller c) {
6         if(c instanceof CorpusController) {
7             System.out.println("Processing corpus " +
8                 ((CorpusController)c).getCorpus().getName());
9         }
10        else {
11            System.out.println(
12                "Running in a simple pipeline");
13        }
14    }
15
16    // controllerExecutionFinished is similar
17 }
```

## Exercise 7: Corpus statistics

Add corpus statistics to your DocStats PR:

- Add private fields to keep a running total count of words (and nouns/verbs).
- Implement `ControllerAwarePR`.
- In the “started” callback, initialize these totals to 0.
- In the “finished” callback
  - check whether you are running in `CorpusController`
  - if so, put the total counts into features on the controller’s `Corpus`.
- You can leave the “aborted” callback empty (or just print a message).

## Exercise 7: Solution

```

1 package module8;
2 // imports omitted for space reasons
3
4 @CreoleResource(name = "Corpus statistics")
5 public class CorpusStats extends AbstractLanguageAnalyser
6     implements ControllerAwarePR {
7     private int totalTokens;
8
9     public void execute() throws ExecutionException {
10         int tokens = document.getAnnotations().get("Token").size();
11         document.getFeatures().put("token_count", tokens);
12         totalTokens += tokens; // keep a running total
13     }
14
15     public void controllerExecutionStarted(Controller c) {
16         totalTokens = 0;
17     }
18
19     public void controllerExecutionFinished(Controller c) {
20         if(c instanceof CorpusController) {
21             ((CorpusController)c).getCorpus().getFeatures()
22                 .put("token_count", totalTokens);
23         }
24     }
25
26     // controllerExecutionAborted omitted
27 }
    
```



# Outline

- 1 CREOLE Metadata
  - CREOLE Recap
- 2 Creating CREOLE Resources
  - Your First Language Analyser
  - Best Practice
  - Your First Visual Resource
  - Ready Made Applications
- 3 Advanced CREOLE
  - CREOLE Management
  - Corpus-level processing
  - Adding actions to the GUI

## Adding actions to the GUI

- Any (language, processing or visual) resource can contribute *actions* to the GATE developer GUI.
- These appear as items on the resource's right-click menu. For example:
  - The “Run” option for controllers comes from the controller editor VR
  - The “Save as...” and “Delete ontology data” options for an ontology LR come from the LR itself.
- This is done by implementing the interface `gate.gui.ActionsPublisher`
- One method, returning a `List` of `javax.swing.Action` objects.

## Exercise 8: ActionsPublisher

Implement cumulative statistics for your DocStats PR:

- keep a running total as before, but rather than resetting it in `controllerExecutionStarted`, provide an action to reset it explicitly.
- provide another action to display the current total.

## Exercise 8: Solution

```

1 package module8;
2 // imports omitted for space reasons
3
4 @CreoleResource(name = "Cumulative statistics")
5 public class CumulativeStats extends AbstractLanguageAnalyser
6     implements ActionsPublisher {
7     // totalTokens and execute() method exactly as in exercise 7
8
9     public List<Action> getActions() {
10         if(actions == null) {
11             actions.add(new AbstractAction("Reset counter") {
12                 public void actionPerformed(ActionEvent e) {
13                     totalTokens = 0;
14                 }
15             });
16
17             actions.add(new AbstractAction("Show current total") {
18                 public void actionPerformed(ActionEvent e) {
19                     JOptionPane.showMessageDialog(
20                         MainFrame.getInstance(), totalTokens + " token(s) since last reset.");
21                 }
22             });
23         }
24         return actions;
25     }
26
27     private List<Action> actions;
28 }
    
```

# Thank you!

## Questions?

More answers at:

- <http://gate.ac.uk> (Our website)
- <http://gate.ac.uk/mail/> (Our mailing list)