Module – Using Groovy Processors

Overview

This document will describe how to write a 'Groovy' script that uses a 'YAML' lookup.

Once complete you will be able to create your own scripts to do just about anything.

YAML

Samples (separated by '---'):

```
- 1
- 2
- 3
- 4
---
- 1: 1
  2: 2
- 1: 1
  2: 2
---
1:
 - 1
 - 2
2:
 - 1
 - 2
---
1:
 1: 1
 2: 2
2:
 1: 1
 2: 2
```

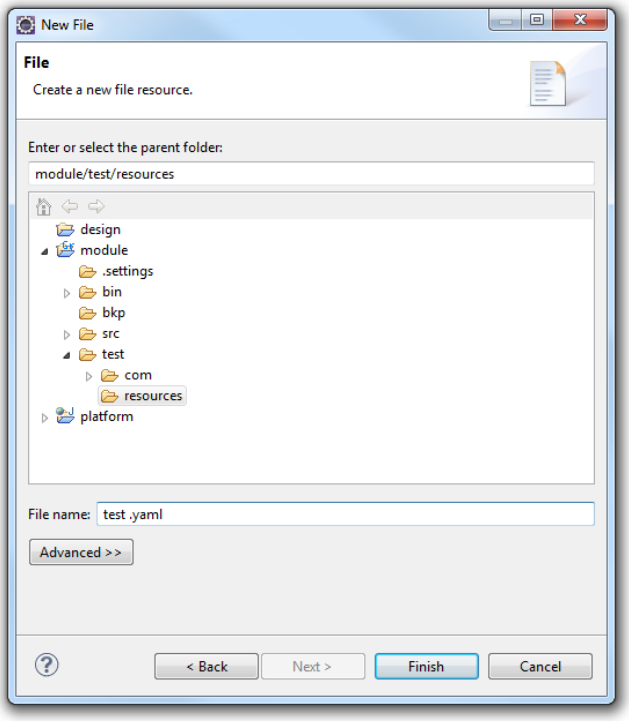
Output (separate items in a collection):

[1, 2, 3, 4] // List
[{1=1, 2=2}, {1=1, 2=2}] // List with Map as members
{1=[1, 2], 2=[1, 2]} // Map with List as members
{1={1=1, 2=2}, 2={1=1, 2=2}} // Map with Map as members

For more information on YAML visit this website: http://yaml.org

Lookup

Start by creating a new file called "test.yaml" in the /module/test/resources folder:



The contents of the file will look like the text below:

```
txt : text
xml : xml
pdf : pdf
xls : excel
```

When the code above is processed it will end up producing a simple HashMap.

Groovy

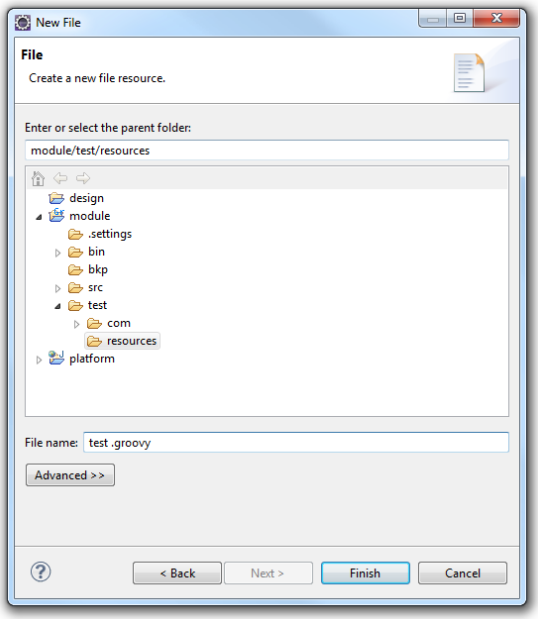
To work with Groovy scripts you will need to install the Groovy-Eclipse plugin, the relevant update sites are shown below:

- 4.3 (Kepler)
  - http://dist.springsource.org/snapshot/GRECLIPSE/e4.3/
- 4.2 and 3.8 (Juno)
  - http://dist.springsource.org/release/GRECLIPSE/e4.2/
- 3.7 (Indigo)
  - http://dist.springsource.org/release/GRECLIPSE/e3.7/
- 3.6 (Helios)
  - http://dist.springsource.org/release/GRECLIPSE/e3.6/
- 3.5 (Galileo)
  - http://dist.springsource.org/release/GRECLIPSE/e3.5/

For more information on the Groovy scripting language see here: http://groovy.codehaus.org/

Script

Next create a new file called “test.groovy” in the /module/test/resources folder:



The 'Groovy Processor' will actually inject in the contents of the YAML file in a variable called 'yaml'. It will also inject the TransactionData object into a variable called 'transactionData'.

The 'yaml' object is actually a collection of objects, each items type will be determined based on the YAML file. In this case it will be a HashMap.

We are going to compare the "com.pilotfish.FileExtension" attribute to the keys in our map and create a new attribute "com.training.FileDescription" based on the result.

The code should look something like below:

```
String filetype =
(String)transactionData.getAttributes().getAttribute(
         "com.pilotfish.FileExtension");
for (Object data : yaml) {
          Map<String, String> map = (Map<String, String>) data;
          String filedesc = map.get(filetype);
          if (filedesc != null) {
               transactionData.getAttributes().setAttribute(
               "com.training.FileDescription", filedesc);
          } else {

               // populate filedesc with filetype if no match
found
transactionData.getAttributes().setAttribute(
               "com.training.FileDescription", filetype);
          }
}
```

Testing

To test this new module we will need to create a new Unit Test.

Create a new class called “com.training.test.GroovyProcessorTest”.

The code in the test() method should look like this:

```
GroovyProcessor processor = new GroovyProcessor();
processor.getConfigurationManager().setValue("GroovyFile",
"./test/resources/test.groovy");
processor.getConfigurationManager().setValue("YamlFile",
"./test/resources/test.yaml");
processor.processData(createTransactionData());
```

Unless you copied a previous test you should have a compilation error on the createTransactionData() method. To solve this problem create a new method that sets the ‘com.pilotfish.FileExtension’ attribute and returns a TransactionData object. Sample implementation shown below:

```
TransactionData data = new TransactionData();

InputStream inputStream = new
FileInputStream("./test/resources/input.txt");

data.setDataStream(inputStream);

data.getAttributes().setAttribute("com.pilotfish.FileExtension",
"xls");

return data;
```

Because we will be asking the Processor to load files we will need to provide an implementation of com.pilotfish.eip.ResourceLoader.

Create a class that implements the ResourceLoader interface. The only method you actually need to implement is the getFile() method shown below:

```
@Override
public File getFile(String paramString) {
      return new File(paramString);
}
```

Last step is to set the ResourceLoader on to our GroovyProcessor. The following line should be added prior to the call to processData():

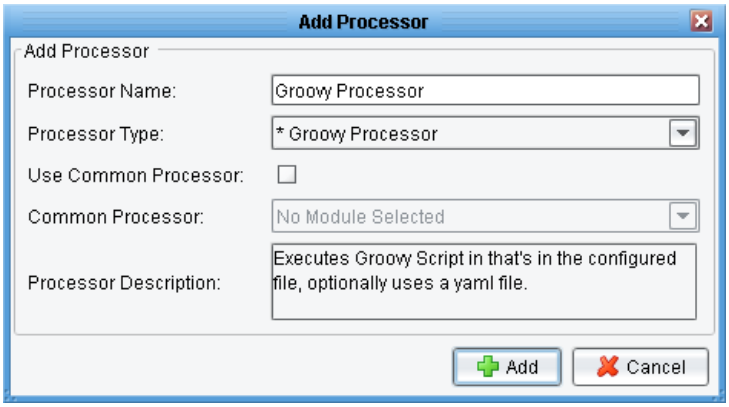
```
processor.setResourceLoader(new TestResourceLoader());
```

We can now execute our test and validate the output.

Console

Open the first 'Simple' route you created.

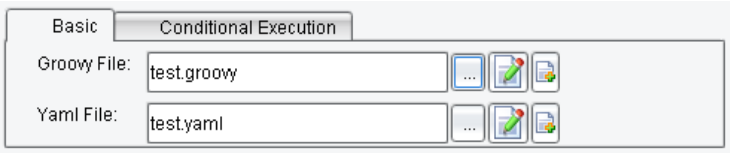
Add a new '* Groovy Processor' after the listener stage:



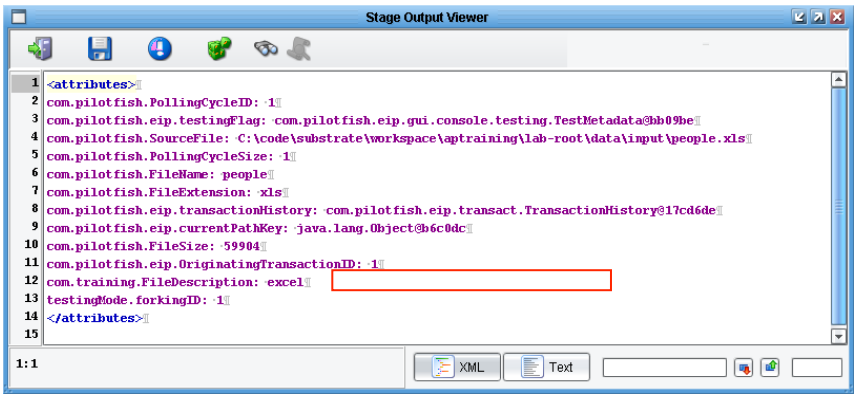
Configure your processor by navigating to the Groovy script and YAML lookup.

If prompted to copy the files to the route directory just click 'Yes'.

The final configuration should look like this:



When you test the route you should see your new attribute being created:



Debug

You can remotely debug a process by adding the following configuration to your launch script:

```
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8765 -Djaxp.debug=true
```