



FUEGO

The background of the slide features a complex diagram of interconnected circles and arrows. At the top left, a circle with a crosshair is connected by arrows to a circle on the left and a circle on the right. The left circle is connected to a central circle, which in turn connects to a circle at the bottom. The right circle is connected to a circle at the top right, which connects to a circle in the middle right, which then connects to a circle at the bottom right. A horizontal line runs across the middle of the slide, passing through the central circle and the top right circle.

USING JUNIT IN FUEGO

RUNNING AND CREATING JUNIT TESTS WITH FUEGO
FUEGO v5

Pablo Victory
pvictory@fuego.com

August 11, 2004

Contents

1	Introduction	3
2	Using JUnit	4
2.1	Cataloging JUnit	4
2.2	Cataloging JUnit Tests	6
2.3	Executing the Tests	9
3	Developing JUnit Tests in Fuego	10
3.1	Cataloging JUnit	10
3.2	Cataloging FuegoJUnit	10
3.3	Creating JUnit Tests in Fuego	13
4	Generating a Test Suite	14
A	SimpleTest class	17
B	References	18

1 Introduction

JUnit is a simple test framework for building and executing unit tests. The purpose of this article is to show how Fuego can be integrated with JUnit¹ to provide unit testing, either because you want to build a process around your unit tests or because you want to test a new Fuego process as it is being developed.

There are two ways of specifying the tests to run. The first and more common way is to have the tests written in Java classes which are then executed from within Fuego.

The second approach is to create the same tests as Fuego Objects that can then be evaluated within the JUnit framework. If your objective is to perform unit tests of your process components then this last option is the one you should adopt.

¹For every example included in this article JUnit version 3.8.1 was used.

2 Using JUnit

The most simple way of integrating Fuego with JUnit is to invoke JUnit tests from a process activity or a Fuego Object method while writing the tests themselves in Java outside Fuego. You can then check the tests results from within Fuego and take appropriate action depending on these results.

2.1 Cataloging JUnit

The first step is to catalog the JUnit jar in Fuego. Create a new *External Resource* of type *Java Class Library*. Name it appropriately (we named it 'JUnit') and then add the `junit.jar` file which comes with JUnit to the library.

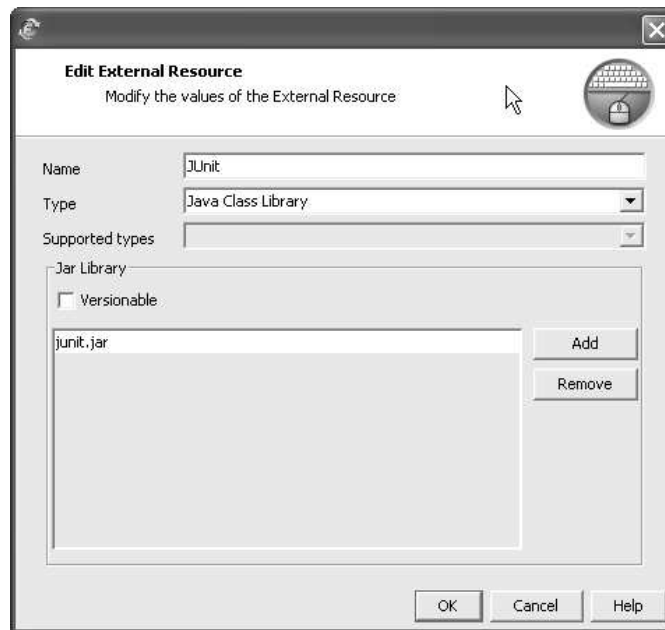


Figure 1: Catalog JUnit jar

Next we have to catalog the JUnit classes needed by Fuego. We have created a new module named 'JUnit' and by right clicking on top of it we bring up the popup menu. We select *Catalog component* and then *Java* from it.

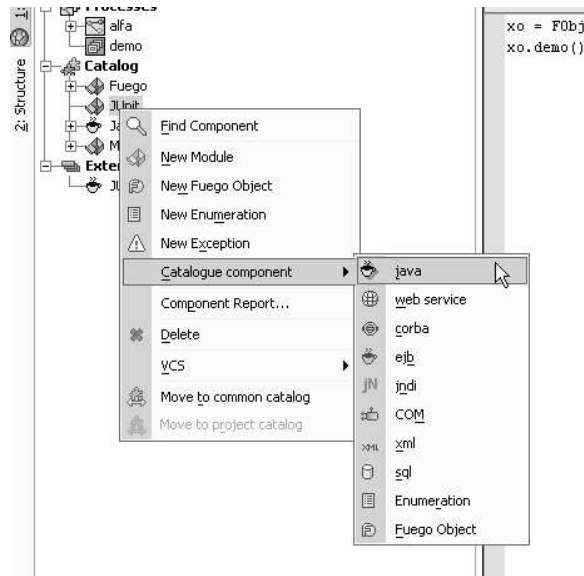


Figure 2: Catalog JUnit classes

A catalog wizard will show up, we select the '**JUnit**' configuration we created before and click 'Next'.

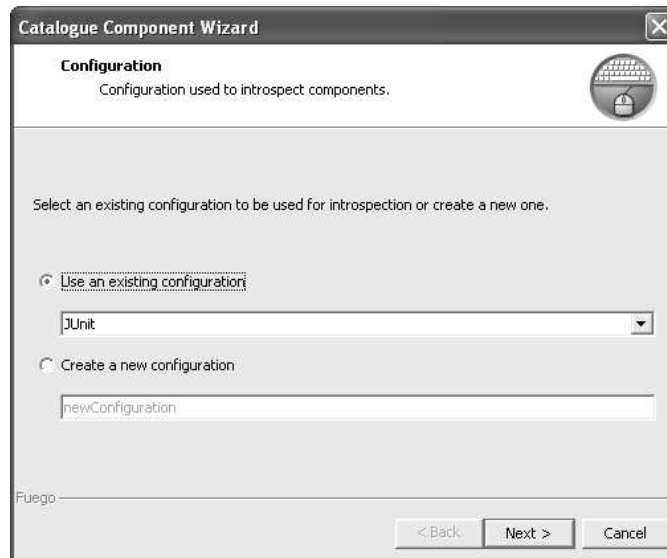


Figure 3: Select JUnit configuration

The wizard then displays a list of available packages and classes included in the java library. We need only the framework package so we select it from the tree.

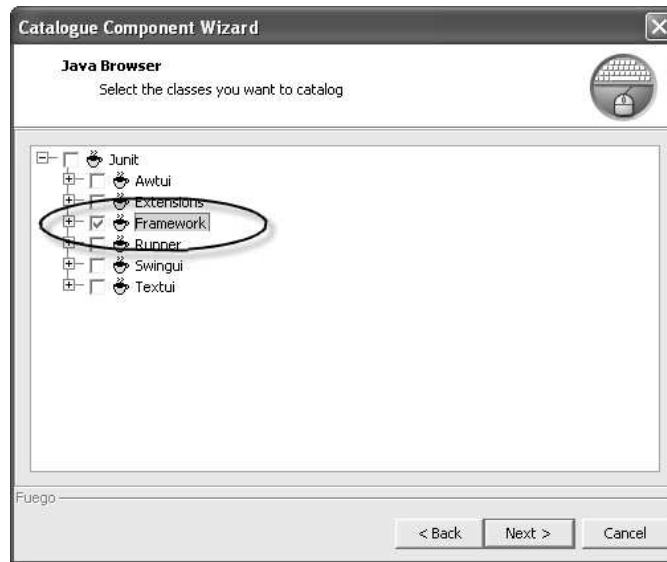


Figure 4: Select necessary classes

2.2 Cataloging JUnit Tests

Our Junit tests have been developed in java and have been compiled and packaged into a jar file². We need to catalog this jar file in order to have access to this tests.

Our first step is to catalog this jar file which includes our test classes. We need to create a new *External Resource* also of type *Java Class Library*. We named our new configuration '**JUnit Tests**'.

²The code for our simple test (**SimpleTest** class) is included in the appendix.

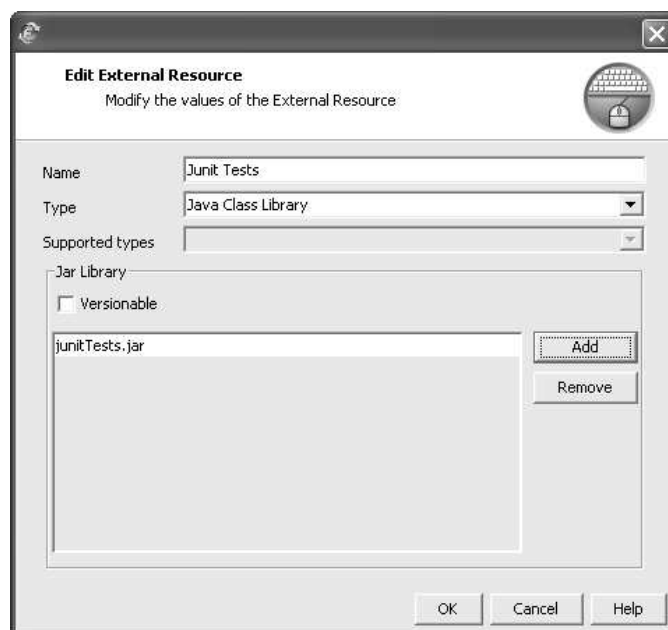


Figure 5: Catalog JUnit Tests

Then we need to catalog the java components included in this jar file. Again we right click on the module named '**JUnit**' and from the popup menu we choose to catalog a component of type *Java*. A catalog component wizard will show up. We select the new '**JUnit Tests**' configuration from it.

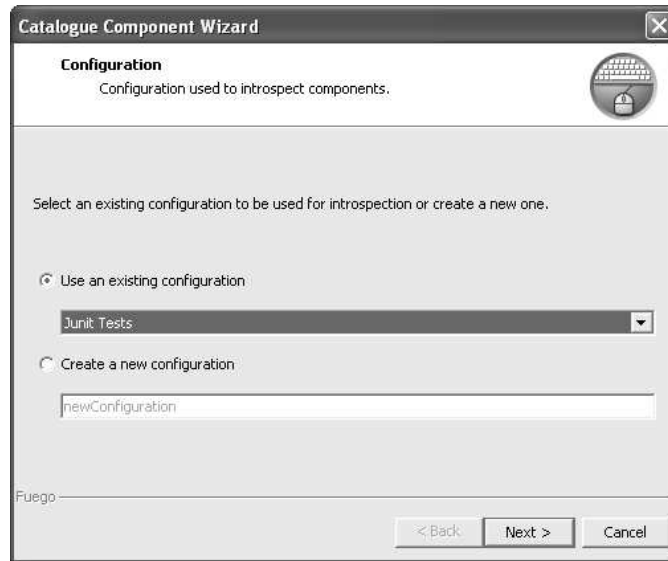


Figure 6: Choose the appropriate configuration

The wizard will then display a tree with all the classes included in the selected configuration. In our jar file we only have one class (**SimpleTest**). We should select all the test classes we wish to execute from within Fuego.

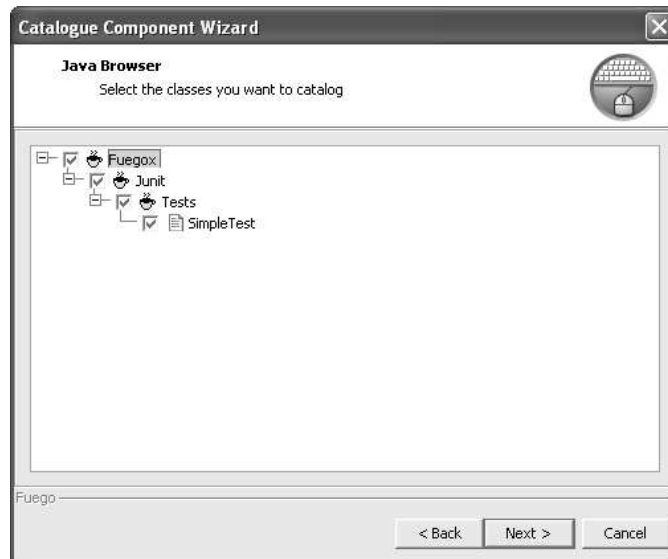


Figure 7: Select the tests

2.3 Executing the Tests

We are now ready to execute the test(s) from within Fuego. The following code will execute the specified test and display the results. This piece of code could be placed inside an activity or as a method in some Fuego Object.

```
1 test = SimpleTest()
2 testResult = run(test)
3
4 display "ERRORS: " + errorCount(testResult)
5 errorsIterator = errors(testResult)
6 while hasMoreElements(errorsIterator) do
7     error = TestFailure(errorsIterator.nextElement())
8     display "ERR: " + exceptionMessage(error)
9 end
10
11 display "FAILURES: " + failureCount(testResult)
12 failsIterator = failures(testResult)
13 while hasMoreElements(failsIterator) do
14     fail = TestFailure(failsIterator.nextElement())
15     display "FAIL: " + exceptionMessage(fail)
16 end
```

- The first line creates a new test of type **SimpleTest**. This is the java component we cataloged and which contains our JUnit test.
- The second line runs the test which returns a **TestResult** object as defined by the JUnit framework.
- Lines 4-9 display the number of errors and what each error represents.
- Lines 11-16 display the number of failures and what each failure represents.

Even though the tests can be executed in Fuego and the results verified and process by Fuego, the tests are written in Java outside Fuego. We will show now how to write your tests using Fuego Objects and execute them in order to test your process functionality.

3 Developing JUnit Tests in Fuego

Fuego also provides a way of writing and executing your tests within Fuego. With this approach we will write the actual tests in a Fuego Object and we will use JUnit to execute them.

3.1 Cataloging JUnit

As with the previous approach the first step is to catalog the JUnit jar in Fuego. Please refer to section [2.1](#) to catalog it.

3.2 Cataloging FuegoJUnit

In order to be able to write and execute the tests Fuego provides a jar file which acts as a bridge between Fuego and JUnit providing a callback mechanism. This jar file provides a **FuegoTestCase** class which will act as a wrapper for the tests we develop in our Fuego Object and will allow JUnit to use the Fuego Objects as **TestCase** objects.

Our first step is to catalog this jar file. We need to create a new *External Resource* also of type *Java Class Library*. We named our new configuration 'JUnitBridge' as it effectively works as a bridge between Fuego and JUnit.

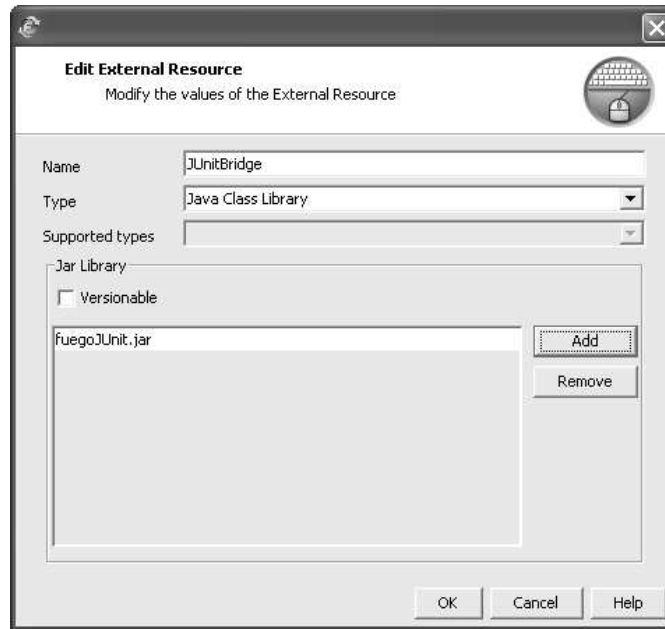


Figure 8: Catalog FuegoJUnit Bridge

Then we need to catalog the java components included in this jar file. Again we right click on the module named '**JUnit**' in our catalog and from the popup menu we choose to catalog a component of type Java. A catalog component wizard will show up. We select the new '**JUnitBridge**' configuration from it.



Figure 9: Choose the JUnitBridge configuration

The wizard will then display a tree with all the classes included in the selected configuration. In our jar file we only have one class. We select it from the tree and press 'Next'.

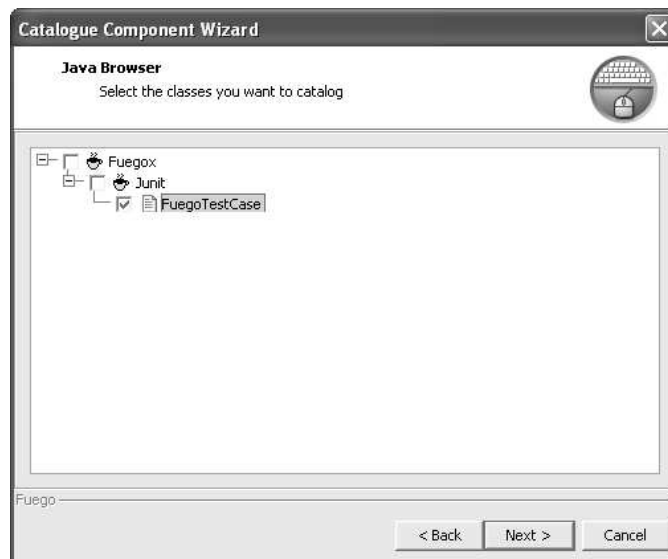


Figure 10: Select the classes

3.3 Creating JUnit Tests in Fuego

To create a JUnit test in Fuego we need to create a Fuego Object which will contain the method or methods which will act as our test units. We have created a simple Fuego Object (named '**FuegoTest**') in our catalog and we have added the following method named '**test**' to it:

```
1 assertEquals Assert
2     using arg1 = 0,
3     arg2 = 1
```

Although simple, the method will represent our test case. This method can be as complex or as simple as you want. Moreover, you could be invoking database components or stored procedures in it to test if they work as expected.

We also create a second Fuego Object named '**Tester**' which will be in charge of executing the tests and displaying the results. In this second Fuego Object we have created a method named `#executeTests()` with the following code in it:

```
1 test = FuegoTestCase()
2 addTestCase test
3     using arg1 = FuegoTest()
4
5 testResult = run(test)
6
7 display "ERRORS: " + errorCount(testResult)
8 errorsIterator = errors(testResult)
9 while hasMoreElements(errorsIterator) do
10     error = TestFailure(errorsIterator.nextElement())
11     display "ERROR: " + exceptionMessage(error)
12 end
13
14 display "FAILURES: " + failureCount(testResult)
15 failsIterator = failures(testResult)
16 while hasMoreElements(failsIterator) do
17     fail = TestFailure(failsIterator.nextElement())
18     display "FAIL: " + exceptionMessage(fail)
19 end
```

- The first line creates an instance of the **FuegoTestCase** bridge object we cataloged previously in section .
- Lines 2 and 3 adds an instance of the **FuegoTest** object where our test resides to the test case we created in line 1. Note that we are

not specifying any particular method in our **FuegoTest** object as the method to be tested. By default the framework will look for a method named `#test()` if no method is explicitly specified.

- Line 5 runs the test.
- Lines 7-19 are similar to the previous example. Here we display any errors or failures encountered during the test.



Debug Warning: Even though both approaches (having created the test in Java or in Fuego) work properly in a runtime environment (either Studio or Enterprise), the second one **can not be used** in Studio in debug mode.

4 Generating a Test Suite

JUnit allows you to build several tests and group them together in a Test Suite. We will implement a Test Suite using Fuego Objects and show how to implement several tests in Fuego.

We first create a **Tester** Fuego Object which will be responsible of implementing, executing and displaying the tests cases. We start by adding methods to this new **Tester** object. Each of these methods will represent a test unit. Our first method is called `#testCompany()` and its code is:

```
1 name = "Fuego"
2 company = Company.getCompanyNamed(name)
3
4 assertNotNull Assert
5     using arg1 = company
6
7 assertEquals Assert
8     using arg1 = company.companyname,
9         arg2 = "Fuegox"
```

The method tests that the `#getCompanyNamed()` method in the **Company** Fuego Object performs correctly and answers back the right Company.

We will create a couple more of methods named `#testEmployee()` and `#calculateTax()` so we can include them in our test suite.

We add now an `#executeTests()` method to our **Tester** object which will create our test suite:

```
1 testSuite = TestSuite()
2
3 test = FuegoTestCase()
4 addTestCase test
5     using arg1 = this,
6         arg2 = "getCompanyNamed"
7 addTest testSuite
8     using arg1 = test
9
10 test = FuegoTestCase()
11 addTestCase test
12     using arg1 = this,
13         arg2 = "testEmployee"
14 addTest testSuite
15     using arg1 = test
16
17 test = FuegoTestCase()
18 addTestCase test
19     using arg1 = this,
20         arg2 = "calculateTax"
21 addTest testSuite
22     using arg1 = test
23
24 testResult = TestResult()
25 run testSuite
26     using arg1 = testResult
27
28 displayResults this
29     using testResult = testResult
```

- Line 1 creates a new **TestSuite** object. This **TestSuite** is defined in the JUnit framework catalogued in section 2.1.
- Line 3 creates a new **FuegoTestCase** bridge object which will serve as a wrapper to our **Tester** Fuego Object.
- Lines 4-6 add our **Tester** object to the **FuegoTestCase** we create in the previous line specifying the method which contains the actual test we want to invoke. This is an optional argument. If no method is specified then a method named `#test()` will be executed. If no `#test()` method is present in the Fuego Object then an error is reported.
- Lines 7-8 add the **FuegoTestCase** we just configured to the test suite we created in line 1.

- Lines 10-15 create a second test and add it to the test suite. Note that the Fuego Object which contains the test is the same but we are specifying a different method to test.
- Lines 17-22 create a third test and add it to the test suite.
- Line 24 creates the **TestResult** object which will contain the results of our test suite.
- Line 25-26 run the test suite.
- Line 28-29 display the results by invoking the `#displayResults()` method (see below).

Finally, we need to implement the `#displayResults()` method, also in our **Tester** object, which displays the number of errors and failures encountered during the test. Our implementation is the following:

```
1 if wasSuccessful(testResult) then
2   display "Success"
3 else
4   display "Failed"
5 end
6
7 display "ERRORS: " + errorCount(testResult)
8 errorsIterator = errors(testResult)
9 while hasMoreElements(errorsIterator) do
10   error = TestFailure(errorsIterator.nextElement())
11   display "ERR: " + exceptionMessage(error)
12   display trace(error)
13 end
14
15 display "FAILURES: " + failureCount(testResult)
16 failsIterator = failures(testResult)
17 while hasMoreElements(failsIterator) do
18   fail = TestFailure(failsIterator.nextElement())
19   display "FAIL: " + exceptionMessage(fail)
20   display trace(fail)
21 end
```


A SimpleTest class

The following is the code used for the SimpleTest class.

```
1 package fuegox.junit.tests;
2
3 import junit.framework.TestCase;
4
5 public class SimpleTest extends TestCase {
6
7
8     public SimpleTest()
9     {
10     }
11
12     protected void runTest() throws Throwable
13     {
14         assertEquals(1, 0);
15     }
16 }
```

B References

[JUnit Home Page](http://www.junit.org/index.htm)

<http://www.junit.org/index.htm>

[JUnit Primer](http://www.clarkware.com/articles/JUnitPrimer.html)

<http://www.clarkware.com/articles/JUnitPrimer.html>

[JUnit best practices](http://www.javaworld.com/javaworld/jw-12-2000/jw-1221-junit.html)

<http://www.javaworld.com/javaworld/jw-12-2000/jw-1221-junit.html>