M2GL V&V – Maven project preparation and test reports generation

Erwan Bousse (erwan.bousse@irisa.fr)

Last updated September 19, 2013

Abstract

This document describes the procedure to configure your Maven project and to annotate your tests in order to automatically produce test reports.

1 Configuring your Maven project

During all lab sessions, you will have to work with Maven. To make it simpler for you, a Maven configuration file (called `vv-configuration.xml`) is available with all required dependencies and plugins. You will have to use this file as a "parent" artifact within your Maven project. To do so, you must refer to this Maven configuration using `<parent></parent>` . Figure 1 shows a sample `pom.xml` file with a reference to `vv-configuration.xml`. The file `vv-configuration.xml` **must** be put in the project folder, so that your project directly works on your teacher computer.

If you look into `vv-configuration.xml`, you will see a very basic Maven configuration file with:

- The dependency to JUnit (making it unnecessary to add it with Eclipse)
- The plugin `maven-javadoc-plugin`, in order to call `javadoc`. Note that we added multiple customs tags specific to tests comments.
- The plugin `jacoco-maven-plugin` in order to generate test coverage HTML reports.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>my</groupId>
  <artifactId>project</artifactId>
  <version>1</version>

  <parent>
    <groupId>istic</groupId>
    <artifactId>vv-configuration</artifactId>
    <relativePath>vv-configuration.xml</relativePath>
    <version>1</version>
  </parent>

</project>
```

Figure 1: Sample pom.xml file with the parent `vv-javadoc-pom.xml`

2 Generating test reports

2.1 Test descriptions with Javadoc

You will have to comment very carefully each one of your JUnit test method using javadoc tags. Except for the `@see` tag, all of them are course-specific. Section 1 explains how to configure Maven/javadoc to make them available. Figure 2 shows how they should be used. The tags to use are the following:

`@see` The tested method. In order for the link to work in the javadoc, use the following syntax (for which autocompletion works in Eclipse):

```
package.MyClass#methodName(type1 type2)
```

`@type` Either “Functionnal” or “Structural”

`@input` The input of the method (both the calling object and the parameters)

`@oracle` What is expected for the test to pass

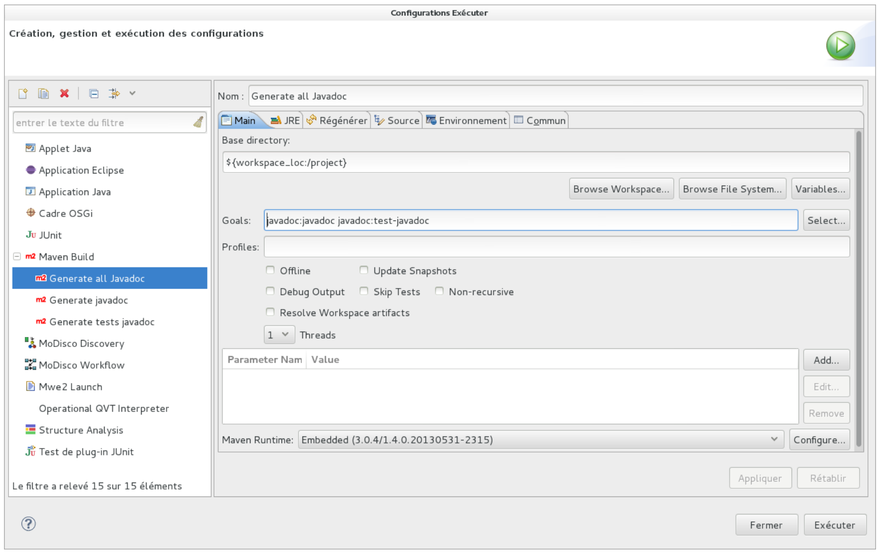
`@passed` Whether the test passed or not

`@correction` If required, the applied patch. Be careful to use `<pre></pre>` to make it readable (see the example Figure 2).

```java
public class testMyClass {

        /**
         * Tests the "doStuff" method normal behavior.
         * @see project.MyClass#doStuff(int)
         * @type Functional
         * @input 5
         * @oracle Must return "true"
         * @passed No
         * @correction
         * <pre>
         * l.9
         * - if (i > 5)
         * + if (i < 5)
         *
         * l.14
         * - value = i;
         * + value = i+2;
         * </pre>
         */
        @Test
        public void testDoStuff() {
                MyClass mc = new MyClass();
                assertFalse(mc.doStuff(5));
        }
}
```

Figure 2: Example of JUnit test case that uses the tags.



Configurations Exécuter

Création, gestion et exécution des configurations

entrer le texte du filtre

- Applet Java
- Application Eclipse
- Application Java
- Cadre OSGi
- JUnit
- Maven Build
  - Generate all Javadoc
  - Generate javadoc
  - Generate tests javadoc
- MoDisco Discovery
- MoDisco Workflow
- Mwe2 Launch
- Operational QVT Interpreter
- Structure Analysis
- Test de plug-in JUnit

Le filtre a relevé 15 sur 15 éléments

Nom : Generate all Javadoc

Main | JRE | Régénérer | Source | Environnement | Commun

Base directory: ${workspace_loc:/project}

Goals: javadoc:javadoc javadoc:test-javadoc

Profiles:

☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts
1 Threads

Parameter Nam	Value

Maven Runtime: Embedded (3.0.4/1.4.0.20130531-2315)

Figure 3: Eclipse run configuration with a Maven build to produce all the javadoc.

To generate the HTML javadoc for the code, you can use the Maven goal `javadoc:javadoc`. For the tests you can use `javadoc:test-javadoc`. Since we make links from the tests towards the code (using `@see`), we need to produce both. A convenient way is to configure a new "Eclipse Run Configuration" of the type "Maven Build" with "`javadoc:javadoc javadoc:test-javadoc`" in the "Goals" field. The output is produced in the folder `target/site/testapidocs`. Figure 3 shows an example of Javadoc run configuration.

2.2 Test coverage report with Jacobo

To generate the jacobo HTML report for the test coverage, you can use the Maven goal `prepare-package`. Again, a convenient way is to configure a new "Eclipse Run Configuration" of the type "Maven Build" with `prepare-package` in the "Goals" field. The output is produced in the folder `target/site/jacobo`.