Japanese Unit Testing

The Groovy Advantage

Groovy simplifies JUnit testing, making it more Groovy, in several ways, including:

- JUnit is built into the groovy runtime, so you can script JUnit tests for your Groovy and Java classes using Groovy syntax.
- Groovy provides many additional JUnit assertion statements (see below)
- Groovy unit tests are easily scriptable with Ant / Maven (see below)
- Groovy provides Groovy Mocks

See also:

- Unit test your java code with groovy
  ⚠ Note: Groovy doesn't support testing for Java code at all! One block issue is impossibility mocking or stubbing aggregated object attribute in the tested Java class.
- testngroove project page
- Using JUnit 4 with Groovy

Example

To write unit tests in Groovy, you have to create a class extending groovy.util.GroovyTestCase.

```
import groovy.util.GroovyTestCase

class MyTest extends GroovyTestCase {
    void testSomething() {
        assert 1 == 1
        assert 2 + 2 == 4 : "We're in trouble, arithmetic is broken"
    }
}
```

Groovy Test Assertions

Apart from the default assertion methods inherited from the JUnit framework's TestCase class, GroovyTestCase also offers additional test assertions:

- assertArrayEquals(Object[] expected, Object[] value)
- assertLength(int length, char[] array)
- assertLength(int length, int[] array)
- assertLength(int length, Object[] array)
- assertContains(char expected, char[] array)
- assertContains(int expected, int[] array)
- assertToString(Object value, String expected)
- assertInspect(Object value, String expected)
- assertScript(final String script) // assert that a script runs without exceptions
- shouldFail(Closure code) // assert that an exception was thrown in that closure
- shouldFail(Class clazz, Closure code) // the same but for a class

Details

By default Groovy unit test cases generate java bytecode and so are just the same as any other Java unit test cases. One thing to watch is often Ant / Maven look for *.java files to find unit tests with pattern matching, rather than *.class files.
There's an option in Maven to ensure you search for classes (and so find any Groovy unit test cases) via this property

```
maven.test.search.classdir = true
```

Once you've got this enabled you can use Maven goals to run individual test cases like this

```
maven test:single -Dtestcase=foo.MyGroovyTest
```

Running GroovyTestCases on the command-line

Since beta-6, you can also run your groovy tests (extending GroovyTestCase) on the command-line. It has simple as launching any other Groovy script or class:

```
groovy MyTest.groovy
```

Running GroovyTestCases in IDEs

Most IDEs support JUnit but maybe don't yet handle Groovy shame!  .
Firstly if you compile the groovy code to bytecode, then it'll just work in any JUnit IDE just fine.

Sometimes though you want to just hack the unit test script and run from in your IDE without doing a build.
If you're IDE doesn't automatically recompile Groovy for you then there's a utility to help you run Groovy unit test cases inside any JUnit IDE without needing to run your Ant / Maven build.

The GroovyTestSuite class is a JUnit TestSuite which will compile and run a GroovyUnit test case
from a command line argument (when run as an application) or from the *_test_* system property when run as a JUnit test suite.
To run the GroovyUnitTest as an application, just do the equivalent of this in your IDE

```
java groovy.util.GroovyTestSuite src/test/Foo.groovy
```

Or to run the test suite inside your IDE, just run the GroovyTestSuite test with this system property defined

```
-Dtest=src/test/Foo.groovy
```

Either of the above can really help improve the development experience of writing Groovy unit test cases in IDEs that don't yet support Groovy natively.

Running a TestSuite containing GroovyTestCase scripts directly in Eclipse

You can take advantage of GroovyTestSuite's ability to compile GroovyTestCase scripts into classes to build a TestSuite which can be run from Eclipse's JUnit runner directly.
The suite() method of TestSuite creates and returns a Test. Within your TestSuite's suite() method, you can create a GroovyTestSuite and use to to compile groovy scripts into Class instances, and add them to a TestSuite that you are building using TestSuite.addSuite(Class).

Here's a TestSuite that contains some GroovyTestCase scripts:

```
public class MyTestSuite extends TestSuite {
     // Since Eclipse launches tests relative to the project root,
     // declare the relative path to the test scripts for convenience
     private static final String TEST_ROOT = "src/test/com/foo/bar/";
     public static TestSuite suite() throws Exception {
         TestSuite suite = new TestSuite();
         GroovyTestSuite gsuite = new GroovyTestSuite();
         suite.addTestSuite(FooTest.class);  // non-groovy test cases welcome, too.
         suite.addTestSuite(gsuite.compile(TEST_ROOT + "BarTest.groovy"));
         suite.addTestSuite(gsuite.compile(TEST_ROOT + "FooFactoryTest.groovy"));
         suite.addTestSuite(gsuite.compile(TEST_ROOT + "BaazTest.groovy"));
         return suite;
     }
}
```

This TestSuite subclass can then be launched as a normal TestSuite in Eclipse. For example, right-click, Run As -> JUnit Test.

From there, the behavior of the JUnit test runner is the same; hierarchy view of all tests and individual methods, their results, etc.

Using normal scripts as unit test cases

You can write scripts like this

```
x = [1, 2, 3]
assert x.size() == 3
```

and use these scripts as unit test cases if you use the GroovyTestSuite class to run them as described below.

When the above script is compiled, it doesn't actually implement JUnit's TestCase and so needs a special runner so that it can be used inside a JUnit test framework. This is what GroovyTestSuite does, it detects scripts like the above and wraps them in a JUnit Test adapter so you can run scripts like the above as a unit test case inside your IDE.
Example of a test suite to test Groovy scripts

```
// RunAllScriptsTests.groovy
import groovy.util.GroovyTestSuite
import junit.framework.Test
import junit.textui.TestRunner
import org.codehaus.groovy.runtime.ScriptTestAdapter

class AllTests {
    static Test suite() {
        def allTests = new GroovyTestSuite()

        allTests.addTest(new
ScriptTestAdapter(allTests.compile("GroovyOK.groovy"), [] as String[]))

        allTests.addTest(new
ScriptTestAdapter(allTests.compile("GroovyOK2.groovy"), [] as String[]))

        return allTests
    }
}

TestRunner.run(AllTests.suite())
```

Sample output

```
groovy RunAllScriptsTests.groovy
..
Time: 0,015

OK (2 tests)
```

New AllTestSuite

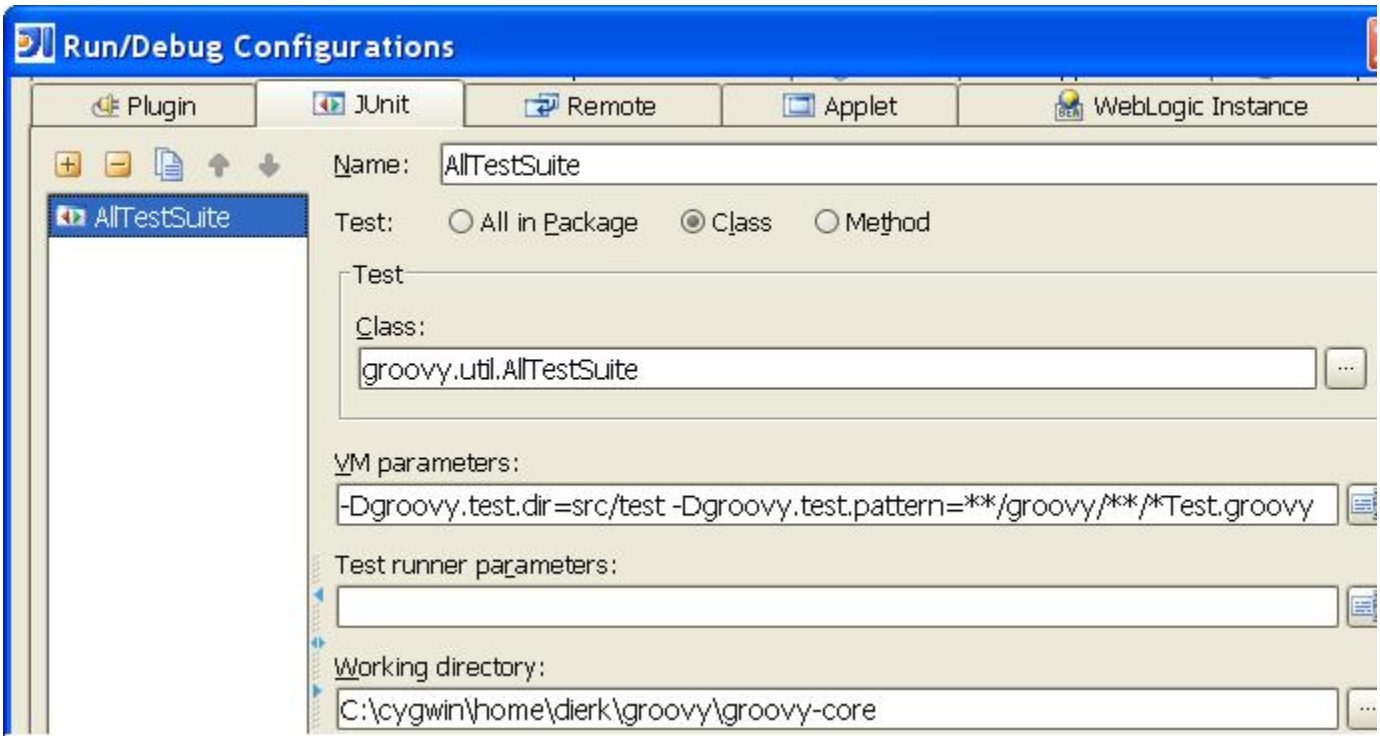
There is a new class groovy.util.AllTestSuite that transparently cares for all the above.

Simply make a Run Configuration in your IDE for this class, providing the following System Properties:

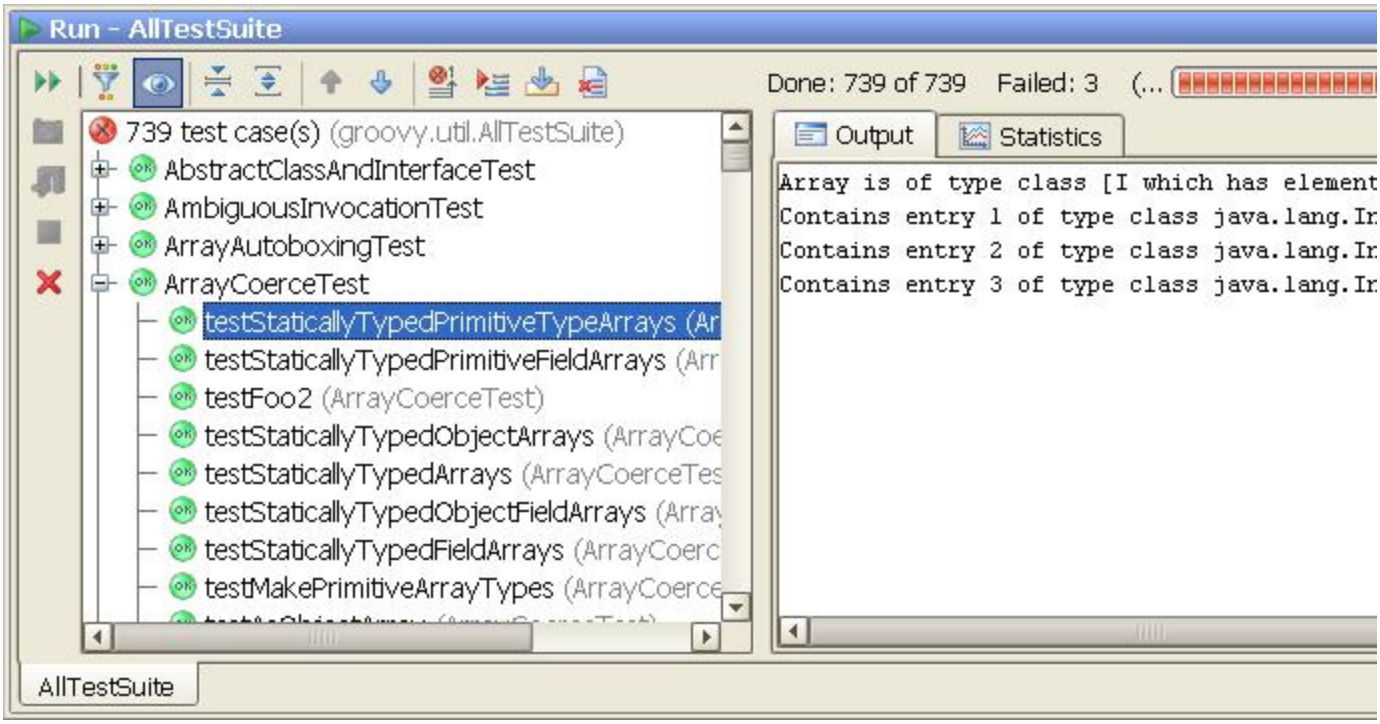
Property name	meaning	default
groovy.test.dir	the directory to search for groovy tests	./test/
groovy.test.pattern	the ant fileset pattern to search below the dir	**/*Test.groovy

See the API documentation of groovy.util.AllTestSuite for more details.

Here is the run config for JetBrains IDEA:



Here is how it looks like when running the AllTestSuite for the the groovy Unit tests of the Groovy project itself:


Run - AllTestSuite
Done: 739 of 739
Failed: 3
739 test case(s) (groovy.util.AllTestSuite)
AbstractClassAndInterfaceTest
AmbiguousInvocationTest
ArrayAutoboxingTest
ArrayCoerceTest
testStaticallyTypedPrimitiveTypeArrays (Ar
testStaticallyTypedPrimitiveFieldArrays (Arr
testFoo2 (ArrayCoerceTest)
testStaticallyTypedObjectArrays (ArrayCoe
testStaticallyTypedArrays (ArrayCoerceTes
testStaticallyTypedObjectFieldArrays (Array
testStaticallyTypedFieldArrays (ArrayCoerc
testMakePrimitiveArrayTypes (ArrayCoerce
Output
Statistics
Array is of type class [I which has element
Contains entry 1 of type class java.lang.In
Contains entry 2 of type class java.lang.In
Contains entry 3 of type class java.lang.In
AllTestSuite