TestBenchRunner

TestBenchRunner makes it possible to create, compile and run JUnit TestSuites that consist of both JUnit tests and TestBench tests. Tests in the Suite are run in order specified when the test suite is created.

TestBenchRunner compiles the classfiles to `{the current working directory}/build`. This can be overridden by defining the System property **`com.vaadin.testbench.build`** (relative to current working directory. All files will be handled as relative to working directory ${user.dir})

Creating a TestSuite

TestBenchRunner can be run from the command line with:

```
com.vaadin.testbench.runner.TestBenchRunner [-options] <Test files>

<test files>      ',' separated files. Accepted files are Test files (.java,
                  .html) and test suite files (.xml, .html) all files will
                  create one test suite in order files are specified. (.html
                  suite file shoulde be a TestBenchIDE suite file consisting
                  of only TestBench tests. This suite will be run as one test)
-options          -p <Base path to files from here>
                  -path <Base path to files from here>
                  -m, -make   Create test suite java and build.xml
                  -c, -connect
                              Connect html tests that come after one another.
                  -h, -help   Show help
```

(**Note.** If -make is not present the TestSuite will be run as JUnit tests after it is created.)
(**Note.** If -p is not defined current working directory will be used as path to files.)
(**Note.** Using -connect will create new .html files that combine tests using “test_” + hashCode)

TestBenchRunner can also be used from a java program.

The make and connect flags can be set with `setMakeTests(boolean)` and `setConnectTests(boolean)` respectively.

Connected tests will print out to System.out a string “End test for [File Name]” when their part of the test is finished. This will help the debugging when a combined test fails.

TestBenchRunner uses `'-encoding utf8'` during compilation. Target encoding can be changed with `setEncoding(String encoding)` or by defining the optional system property `com.vaadin.testbench.encoding`

Example code for running a TestBenchSuite with the TestBenchRunner:

```
TestBenchRunner runner = new TestBenchRunner();

List<String> tests = new LinkedList<String>();
tests.add("Money_tst.java");
tests.add("Money_tst2.java");
TestBenchSuite suite = runner.createTestSuite(tests, "test", "Suite Test");

boolean success = runner.runTestSuites(suite);
```

Example output success:

```
Browser  : winxp-firefox35
Testsuite: Suite Test
Tests run: 2, Failures: 0, Errors: 0, Time elapsed: 0.015 sec
```

Example output fail:

```
Browser  : winxp-firefox35
Testsuite: Suite Test
Tests run: 2, Failures: 1, Errors: 0, Time elapsed: 0.010 sec
testMoney_tst2(Money_tst2): expected:<8> but was:<10>
```

(**Note.** Test suite files .xml and .html need to use `parseFiles(files[], path)` or `parseTestSuite(file, path)`)

TestBenchRunner saves all created TestBenchSuites in a list in the order they were created and can be requested at any time from the TestBenchRunner object. TestBenchSuites can be gotten as a List<TestBenchSuite> containing all suites for TBR `runner.getTestBenchSuites()` or a specific suite with `runner.getTestBenchSuite(int)`

Results for the latest test run (if run with TBR) can be requested from TestBenchSuite for a specific browser `TestResult = suite.getResult(String)` or for all browsers that have results `Map<String, TestResult> = suite.getResults()`

Running TestSuite as Apache Ant <junit>

The TestSuite example could be run as a normal Apache Ant JUnit test. The java file that is run with ant should look as follows:

TestSuite.java

```
public class TestSuite {
  public static junit.framework.Test suite() throws Exception {
    TestBenchRunner runner = new TestBenchRunner();

    List<String> tests = new LinkedList<String>();
    tests.add("Money_tst.java");
    tests.add("Money_tst2.java");
    TestBenchSuite suite = runner.createTestSuite(tests, "test", "Suite Test");

    return suite.getTestSuite();
  }
}
```

(**Note.** If using multiple browsers use `suite.getTestSuite("BROWSER_NAME");` as `getTestSuite()` will return first test suite in iterator set)

The JUnit target in the ant file should have the following structure:

```
<target name="run-tests">
 <junit>
  <classpath>
   <fileset dir="${tesbench.dir}" includes="vaadin-testbench-*.jar" />
   <!-- ant path has tools.java (linux/windows, osx?) -->
   <path path="${java.class.path}" />
   <pathelement path="${test.classes}" />
  </classpath>
```

```
    <formatter type="plain" usefile="false" />

    <jvmarg value="-Dcom.vaadin.testbench.build=test/build" />

    <batchtest fork="yes" haltonerror="yes" haltonfailure="yes">
     <fileset dir="../src">
      <include name="**/test/*.java" />
     </fileset>
    </batchtest>
   </junit>
  </target>
```

The running of tests will halt if any of the tests in the TestSuite fail.
Example output:

```
[junit] Testsuite: TestSuite
[junit] Tests run: 2, Failures: 0, Errors: 0, Time elapsed: 0,015 sec
[junit] Testcase: testMoney_tst took 0 sec
[junit] Testcase: testMoney_tst2 took 0 sec
```

Getting Ant JUnit to log test failure when running with TestBenchRunner:

ReportingTest.java

```
 public class ReportingTest extends junit.framework.TestCase {
   public void testreport_test() throws Throwable {

     TestBenchRunner tbr = new TestBenchRunner();

     TestBenchSuite tbs = tbr.parseFiles(new String[] { "Money_tst.java",
     "Money_tst2.java" }, "test");

     tbr.runTestSuites(tbs);

     junit.framework.TestResult tr = tbs.getResult("winxp-firefox35");
     if (tr.failureCount() > 0) {
       throw tr.failures().nextElement().thrownException();
     }
     if (tr.errorCount() > 0) {
       throw tr.errors().nextElement().thrownException();
     }
   }
 }
```

When a test fails and the failure is thrown it will show up in TeamCity as

ReportTest.testreport_test (com.vaadin.testbench.test)

```
junit.framework.AssertionFailedError: expected:<31> but was:<30>
    at junit.framework.Assert.fail(Assert.java:47)
    at ....
```

and in Apache Ant as

```
Tests run: 1, Failures: 1, Errors: 0, Time elapsed: 0,578 sec
Testcase: testreport_test took 0,469 sec
FAILED
expected:<31> but was:<30>
junit.framework.AssertionFailedError: expected:<31> but was:<30>
 at Money_tst2.testMoney_tst2(Money_tst2.java:11)
 .....
```

Test suite file builds (.xml, .html)

Test suites can not define other test suites inside themselves. When using multiple test suite files for the creation of a TestBenchSuite the test suite files should be defined separately in the order their tests should be run.

.xml test suite

Xml test suite uses the `<title>` node to set the name for the test if used alone.

The `<path>` node defines the common path from this file to where the test files named in this suite are stored. If no path is defined the default path used will be the path to this file.

The `<testfile>` node defines the test file and will be added in the order they are named in the suite file. `<testfile>` can be named as only file name or as path + file name (where path is the absolute path to the test file, path to the test file from this file or as extra addition to `<path>`)

MixedTests.xml

```
<?xml version="1.0"?>
<testsuite>
      <title>MixedTests</title>
      <path>test/files/are/here</path>
      <testfile>Money_tst.java</testfile>
      <testfile>tst3.html</testfile>
</testsuite>
```

.html test suite

Html suite is a suite created by the IDE using a [Save Test Suite] target.

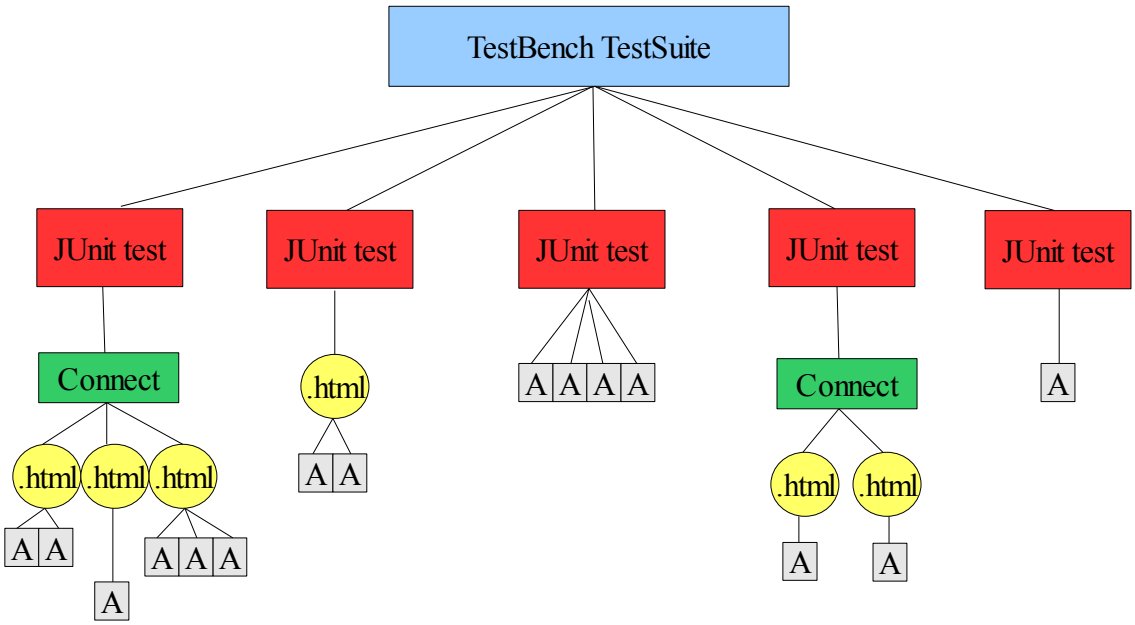
Test_Suite.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta content="text/html; charset=UTF-8" http-equiv="content-type" />
  <title>Test Suite</title>
</head>
<body>
<table id="suiteTable" cellpadding="1" cellspacing="1" border="1"
class="selenium"><tbody>
<tr><td><b>Test Suite</b></td></tr>
<tr><td><a href="tst.html">Sampler_test</a></td></tr>
<tr><td><a href="tst3.html">Test_Sampler</a></td></tr>
</tbody></table>
</body>
</html>
```

To name the html test suite created by the IDE change the first `<tr><td><b>NAME_HERE</b></td></tr>` else the name will be “Test Suite”. If title is not defined a file “Suite.html” will be created.
(**Note!!** Suite.html will be overwritten so **don't** name your own file Suite.html in the directory of the suite file)

Build of a TestBenchSuite

Example illustration of a TestBenchSuite with different test builds.



Each JUnit test will open and close the browser if running a TestBench html test. By using TestSuites html tests can be connected together and test parts can be created as modules that can be reused for other tests.

If the -connect flag was used the firt 2 JUnit tests would be combined to one JUnit test. The order would still stay the same (Number of asserts [2] [1] [3] [2]).

TestBenchRunner system properties

Optional:

`-Dcom.vaadin.testbench.temp` = temporary file dir instead of ”java.io.tmpdir”

`-Dcom.vaadin.testbench.build` = build path where to save .class files.

`-Dcom.vaadin.testbench.browsers` = ',' separated list of browsers to create test suites for.

`-Dcom.vaadin.testbench.encoding` = source file encoding (UTF-8 if none defined)