Taking advantage of TestNG XML reports

- About TestNG
- TestNG Basic Concepts
- Importing TestNG XML reports
  - Entities
    - Mapping of fields from the report to the Test issue
    - Examples
  - Status
- References

About TestNG

TestNG is a testing framework for Java, mostly focused on unit testing but not only.

Similar to JUnit, TestNG is also used for writing integration and acceptance tests, making use of other libraries such as Selenium.

TestNG XML reports are normally creating in the context of Java applications; however, so other languages and test automation frameworks may also be able to generate this kind of reports.

TestNG is quite feature-complete as may be seen in its online documentation.

Supported versions

Xray supports TestNG 6.14 XML reports.

TestNG Basic Concepts

In TestNG, you have Test methods, Parameterized Test methods, Test classes, Test groups, suites.

- A Test method is a test case;
- Parameterized Tests are a way of specifying input values for a given Test (similar to an example in a Cucumber Scenario Outline). Parameters may come from the XML configuration file used by TestNG or from a data provider (i.e. a method that generates a set of values);
- Tests code is implemented within Test classes;
- Test groups are somehow similar to using Jira labels in Jira issues, in the way that they're used to mark the Test methods as belonging to some "category"; these groups can be used afterwards to more easily select the Test to be run;
- A suite is a configuration file (i.e. a XML file) that is used to enumerate all the Tests to be run, based on the Test classes, Test groups, packages, Test methods.

TestNG uses "attributes" in order to ascribe behavior/characteristics to certain parts of your automated test code; these attributes can be set in the Test method's code.

Learn in practice

Please look at the basic Java example: Testing using TestNG in Java.

Importing TestNG XML reports

Below is a simplified example of a TestNG XML report containing a Test Suite with some Test Cases.

Sample TestNG 6.14 XML report

```
<?xml version="1.0" encoding="UTF-8"?>
<testng-results skipped="0" failed="2" ignored="0" total="8" passed="6">
  <reporter-output>
  </reporter-output>
  <suite name="TestAll" duration-ms="25" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
    <groups>
    </groups>
    <test name="calculator" duration-ms="25" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:
48Z">
```

```
      <class name="com.xpand.java.CalcTest">
        <test-method status="PASS" signature="setUp()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]" name="
setUp" is-config="true" duration-ms="5" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
          <reporter-output>
          </reporter-output>
        </test-method> <!-- setUp -->
        <test-method status="PASS" signature="CanAddNumbers()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]"
name="CanAddNumbers" duration-ms="2" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
          <reporter-output>
          </reporter-output>
          <attributes>
            <attribute name="test">
              <![CDATA[CALC-2]]>
            </attribute> <!-- test -->
            <attribute name="requirement">
              <![CDATA[CALC-1234]]>
            </attribute> <!-- requirement -->
            <attribute name="labels">
              <![CDATA[core addition]]>
            </attribute> <!-- labels -->
          </attributes>
        </test-method> <!-- CanAddNumbers -->
        <test-method status="PASS" signature="CanAddNumbersFromGivenData(int, int, int)[pri:0, instance:com.
xpand.java.CalcTest@36d4b5c]" name="CanAddNumbersFromGivenData" duration-ms="0" started-at="2018-03-06T17:22:
48Z" data-provider="ValidDataProvider" finished-at="2018-03-06T17:22:48Z">
          <params>
            <param index="0">
              <value>
                <![CDATA[1]]>
              </value>
            </param>
            <param index="1">
              <value>
                <![CDATA[2]]>
              </value>
            </param>
            <param index="2">
              <value>
                <![CDATA[3]]>
              </value>
            </param>
          </params>
          <reporter-output>
          </reporter-output>
          <attributes>
            <attribute name="test">
              <![CDATA[CALC-1]]>
            </attribute> <!-- test -->
            <attribute name="requirement">
              <![CDATA[CALC-1234]]>
            </attribute> <!-- requirement -->
            <attribute name="labels">
              <![CDATA[]]>
            </attribute> <!-- labels -->
          </attributes>
        </test-method> <!-- CanAddNumbersFromGivenData -->
        <test-method status="FAIL" signature="CanAddNumbersFromGivenData(int, int, int)[pri:0, instance:com.
xpand.java.CalcTest@36d4b5c]" name="CanAddNumbersFromGivenData" duration-ms="1" started-at="2018-03-06T17:22:
48Z" data-provider="ValidDataProvider" finished-at="2018-03-06T17:22:48Z">
          <params>
            <param index="0">
              <value>
                <![CDATA[2]]>
              </value>
            </param>
            <param index="1">
              <value>
                <![CDATA[3]]>
              </value>
            </param>
            <param index="2">
```

```
              <value>
                <![CDATA[4]]>
              </value>
            </param>
          </params>
          <exception class="java.lang.AssertionError">
            <message>
              <![CDATA[expected [4] but found [5]]]>
            </message>
            <full-stacktrace>
              <![CDATA[java.lang.AssertionError: expected [4] but found [5]
        at org.testng.Assert.fail(Assert.java:93)
        at org.testng.Assert.failNotEquals(Assert.java:512)
        at org.testng.Assert.assertEqualsImpl(Assert.java:134)
        at org.testng.Assert.assertEquals(Assert.java:115)
        at org.testng.Assert.assertEquals(Assert.java:388)
        at org.testng.Assert.assertEquals(Assert.java:398)
        at com.xpand.java.CalcTest.CanAddNumbersFromGivenData(CalcTest.java:40)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.testng.internal.MethodInvocationHelper.invokeMethod(MethodInvocationHelper.java:108)
        at org.testng.internal.Invoker.invokeMethod(Invoker.java:661)
        at org.testng.internal.Invoker.invokeTestMethod(Invoker.java:869)
        at org.testng.internal.Invoker.invokeTestMethods(Invoker.java:1193)
        at org.testng.internal.TestMethodWorker.invokeTestMethods(TestMethodWorker.java:126)
        at org.testng.internal.TestMethodWorker.run(TestMethodWorker.java:109)
        at org.testng.TestRunner.privateRun(TestRunner.java:744)
        at org.testng.TestRunner.run(TestRunner.java:602)
        at org.testng.SuiteRunner.runTest(SuiteRunner.java:380)
        at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:375)
        at org.testng.SuiteRunner.privateRun(SuiteRunner.java:340)
        at org.testng.SuiteRunner.run(SuiteRunner.java:289)
        at org.testng.SuiteRunnerWorker.runSuite(SuiteRunnerWorker.java:52)
        at org.testng.SuiteRunnerWorker.run(SuiteRunnerWorker.java:86)
        at org.testng.TestNG.runSuitesSequentially(TestNG.java:1301)
        at org.testng.TestNG.runSuitesLocally(TestNG.java:1226)
        at org.testng.TestNG.runSuites(TestNG.java:1144)
        at org.testng.TestNG.run(TestNG.java:1115)
        at org.apache.maven.surefire.testng.TestNGExecutor.run(TestNGExecutor.java:283)
        at org.apache.maven.surefire.testng.TestNGXmlTestSuite.execute(TestNGXmlTestSuite.java:75)
        at org.apache.maven.surefire.testng.TestNGProvider.invoke(TestNGProvider.java:120)
        at org.apache.maven.surefire.booter.ForkedBooter.invokeProviderInSameClassLoader(ForkedBooter.java:373)
        at org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:334)
        at org.apache.maven.surefire.booter.ForkedBooter.execute(ForkedBooter.java:119)
        at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:407)
]]>
            </full-stacktrace>
          </exception> <!-- java.lang.AssertionError -->
          <reporter-output>
          </reporter-output>
          <attributes>
            <attribute name="test">
              <![CDATA[CALC-1]]>
            </attribute> <!-- test -->
            <attribute name="requirement">
              <![CDATA[CALC-1234]]>
            </attribute> <!-- requirement -->
            <attribute name="labels">
              <![CDATA[]]>
            </attribute> <!-- labels -->
          </attributes>
        </test-method> <!-- CanAddNumbersFromGivenData -->
        <test-method status="PASS" signature="CanAddNumbersFromGivenData(int, int, int)[pri:0, instance:com.
xpand.java.CalcTest@36d4b5c]" name="CanAddNumbersFromGivenData" duration-ms="0" started-at="2018-03-06T17:22:
48Z" data-provider="ValidDataProvider" finished-at="2018-03-06T17:22:48Z">
          <params>
            <param index="0">
              <value>
                <![CDATA[-1]]>
```

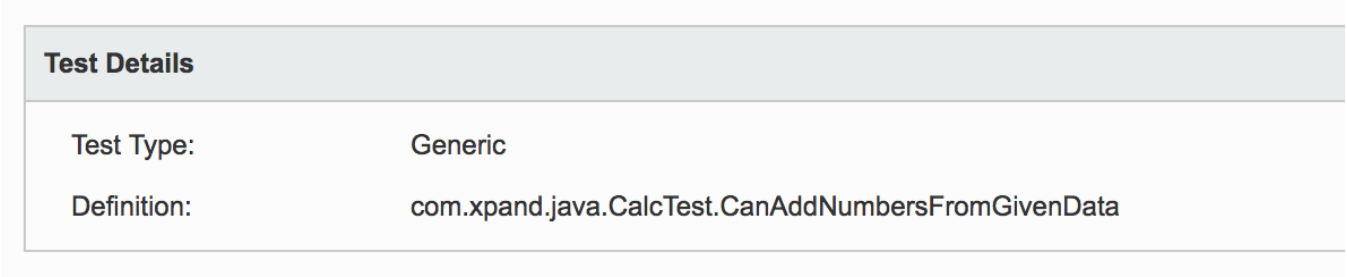
```
            </value>
          </param>
          <param index="1">
            <value>
              <![CDATA[1]]>
            </value>
          </param>
          <param index="2">
            <value>
              <![CDATA[0]]>
            </value>
          </param>
        </params>
        <reporter-output>
        </reporter-output>
        <attributes>
          <attribute name="test">
            <![CDATA[CALC-1]]>
          </attribute> <!-- test -->
          <attribute name="requirement">
            <![CDATA[CALC-1234]]>
          </attribute> <!-- requirement -->
          <attribute name="labels">
            <![CDATA[]]>
          </attribute> <!-- labels -->
        </attributes>
      </test-method> <!-- CanAddNumbersFromGivenData -->
      <test-method status="PASS" signature="CanDivide()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]"
name="CanDivide" duration-ms="1" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
        <reporter-output>
        </reporter-output>
        <attributes>
          <attribute name="test">
            <![CDATA[]]>
          </attribute> <!-- test -->
          <attribute name="requirement">
            <![CDATA[CALC-1237]]>
          </attribute> <!-- requirement -->
          <attribute name="labels">
            <![CDATA[]]>
          </attribute> <!-- labels -->
        </attributes>
      </test-method> <!-- CanDivide -->
      <test-method status="PASS" signature="CanMultiplyX()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]"
name="CanMultiplyX" duration-ms="0" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
        <reporter-output>
        </reporter-output>
        <attributes>
          <attribute name="test">
            <![CDATA[]]>
          </attribute> <!-- test -->
          <attribute name="requirement">
            <![CDATA[CALC-1236]]>
          </attribute> <!-- requirement -->
          <attribute name="labels">
            <![CDATA[]]>
          </attribute> <!-- labels -->
        </attributes>
      </test-method> <!-- CanMultiplyX -->
      <test-method status="FAIL" signature="CanDoStuff()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]"
name="CanDoStuff" duration-ms="0" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
        <exception class="java.lang.AssertionError">
          <message>
            <![CDATA[null]]>
          </message>
          <full-stacktrace>
            <![CDATA[java.lang.AssertionError: null
      at org.testng.Assert.fail(Assert.java:93)
      at org.testng.Assert.assertNotEquals(Assert.java:897)
      at org.testng.Assert.assertNotEquals(Assert.java:902)
      at com.xpand.java.CalcTest.CanDoStuff(CalcTest.java:86)
```

```
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.testng.internal.MethodInvocationHelper.invokeMethod(MethodInvocationHelper.java:108)
        at org.testng.internal.Invoker.invokeMethod(Invoker.java:661)
        at org.testng.internal.Invoker.invokeTestMethod(Invoker.java:869)
        at org.testng.internal.Invoker.invokeTestMethods(Invoker.java:1193)
        at org.testng.internal.TestMethodWorker.invokeTestMethods(TestMethodWorker.java:126)
        at org.testng.internal.TestMethodWorker.run(TestMethodWorker.java:109)
        at org.testng.TestRunner.privateRun(TestRunner.java:744)
        at org.testng.TestRunner.run(TestRunner.java:602)
        at org.testng.SuiteRunner.runTest(SuiteRunner.java:380)
        at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:375)
        at org.testng.SuiteRunner.privateRun(SuiteRunner.java:340)
        at org.testng.SuiteRunner.run(SuiteRunner.java:289)
        at org.testng.SuiteRunnerWorker.runSuite(SuiteRunnerWorker.java:52)
        at org.testng.SuiteRunnerWorker.run(SuiteRunnerWorker.java:86)
        at org.testng.TestNG.runSuitesSequentially(TestNG.java:1301)
        at org.testng.TestNG.runSuitesLocally(TestNG.java:1226)
        at org.testng.TestNG.runSuites(TestNG.java:1144)
        at org.testng.TestNG.run(TestNG.java:1115)
        at org.apache.maven.surefire.testng.TestNGExecutor.run(TestNGExecutor.java:283)
        at org.apache.maven.surefire.testng.TestNGXmlTestSuite.execute(TestNGXmlTestSuite.java:75)
        at org.apache.maven.surefire.testng.TestNGProvider.invoke(TestNGProvider.java:120)
        at org.apache.maven.surefire.booter.ForkedBooter.invokeProviderInSameClassLoader(ForkedBooter.java:373)
        at org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:334)
        at org.apache.maven.surefire.booter.ForkedBooter.execute(ForkedBooter.java:119)
        at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:407)
]]>
            </full-stacktrace>
          </exception> <!-- java.lang.AssertionError -->
          <reporter-output>
          </reporter-output>
        </test-method> <!-- CanDoStuff -->
        <test-method status="PASS" signature="CanSubtract()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]"
name="CanSubtract" duration-ms="0" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:22:48Z">
          <reporter-output>
          </reporter-output>
          <attributes>
            <attribute name="test">
              <![CDATA[]]>
            </attribute> <!-- test -->
            <attribute name="requirement">
              <![CDATA[CALC-1235]]>
            </attribute> <!-- requirement -->
            <attribute name="labels">
              <![CDATA[core]]>
            </attribute> <!-- labels -->
          </attributes>
        </test-method> <!-- CanSubtract -->
        <test-method status="PASS" signature="tearDown()[pri:0, instance:com.xpand.java.CalcTest@36d4b5c]"
name="tearDown" is-config="true" duration-ms="0" started-at="2018-03-06T17:22:48Z" finished-at="2018-03-06T17:
22:48Z">
          <reporter-output>
          </reporter-output>
        </test-method> <!-- tearDown -->
      </class> <!-- com.xpand.java.CalcTest -->
    </test> <!-- calculator -->
  </suite> <!-- TestAll -->
</testng-results>
```

Entities

Test Cases are imported to Xray's ***Generic*** **Test issues.** The "classname" and "methodname" attributes are concatenated and mapped to the **Generic Test Definition** field of the Generic Test.

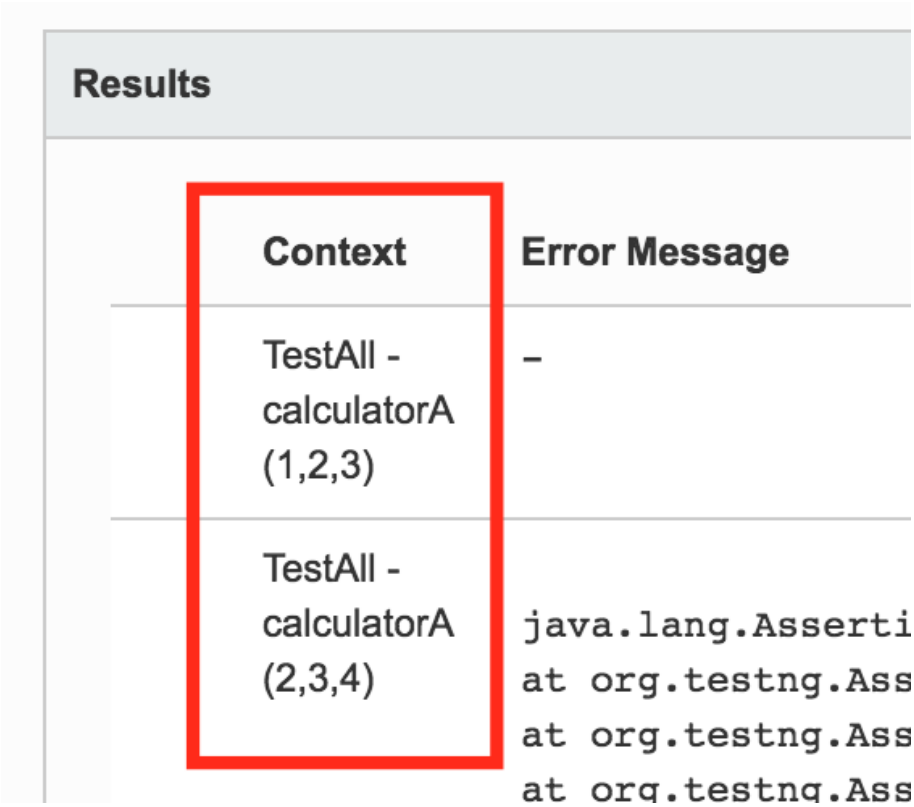
If a Test already exists with the same **Generic Test Definition**, then it is not created again.



Test Cases are imported to a new (or a user-specified) Test Execution in the context of some project, along with their respective execution results.

Parameterized Tests are imported to the same Test Run, where each set of parameters is identified by a different context.

The context (shown on the left side of the execution screen details) is with the name of the "suite" plus the name of the "test" as defined in the XML configuration file used by TestNG. If the Test method corresponds to a parameterized test, then the values used for that specific run are appended to the context.



Example of a TestNG configuration file

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="TestAll">

        <test name="calculatorA">
                <classes>
                        <class name="com.xpand.java.CalcTest" />
                </classes>
        </test>

        <test name="calculatorB">
                <classes>
                        <class name="com.xpand.java.CalcTest" />
                </classes>
        </test>


</suite>
```

Mapping of fields from the report to the Test issue

TestNG XML report	Test in Jira
"methodname" attribute	Summary field
"classname" attribute + "." + "methodname" attribute	Generic Test Definition custom field
groups	labels
"labels" attribute under the <attributes> tag, within some <test-method>, containing one or more labels delimited by space	labels
"test" attribute under the <attributes> tag, within some <test-method>, containing a Jira key of the Test issue	identification of Test issue key in Jira
"requirement" attribute under the <attributes> tag, within some <test-method>, containing a Jira key of requirement (s)	link to requirement(s)

Notes:

- linked requirements may be identified by setting an attribute named "requirement" with the respective issue key at the Test method level, using the ITestResult object"; multiple requirements may be specified by using the space delimeter
- the Test to report results against to may be identified by using and attribute named "test";
    - if the "Test" attribute is used explicitly, then the Test must exist or else it will not be imported.

Examples

Consider running some Tests implemented in the following Java class.

excerpt of Java Test class

```
package com.xpand.java;

import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.DataProvider;
import org.testng.Reporter;
import org.testng.reporters.XMLReporter;
import org.testng.ITestResult;
import com.xpand.annotations.Xray;

public class CalcTest {

    @BeforeSuite
    public void setUp() throws Exception {

    }

    @AfterSuite
    public void tearDown() throws Exception {
    }



    @DataProvider
    public Object[][] ValidDataProvider() {
        return new Object[][]{
            {  1, 2, 3 },
            {  2, 3, 4 },  // error or the data itself :)
            { -1, 1, 0 }
        };
    }


    @Test(dataProvider = "ValidDataProvider")
    public void CanAddNumbersFromGivenData(final int a, final int b, final int c)
    {
        Assert.assertEquals(Calculator.Add(a, b), c);
    }


        @Test
    public void CanAddNumbers()
    {
        Assert.assertEquals(Calculator.Add(1, 1),2);
        Assert.assertEquals(Calculator.Add(-1, 1),0);
        ITestResult result = Reporter.getCurrentTestResult();
        result.setAttribute("requirement", "CALC-1234");   // Xray will try to create a link to this
requirement issue
        result.setAttribute("test", "CALC-2");             // Xray will try to find this Test issue and report
result against it
        result.setAttribute("labels", "core addition");    // Xray will add this(ese) label(s) to the
associated Test issue
    }

    ...
}
```

In the first Test method, we can see an example of a parameterized test using a data provider. Xray would try to find a Test with the same Generic Test Definition, if it does not find one then it will create a Test issue.

In the second Test method, Xray will try to find an existing Test having the issue key CALC-2. If it does not find one and no Test with same Generic Test Definition already exists then a Generic Test with the summary "CanAddNumbers" would be created.

If there is a requirement with the key "CALC-1234", it would create the link to that requirement.

Additionaly, "core" and "addition" would be added to the Test as labels.

> **Learn more**
>
> It's possible to use Java annotations to more easily mark the Test methods with the linked requirement(s), Test or with the labels you wish to add. Please have a look at the examples shown in Testing using TestNG in Java .

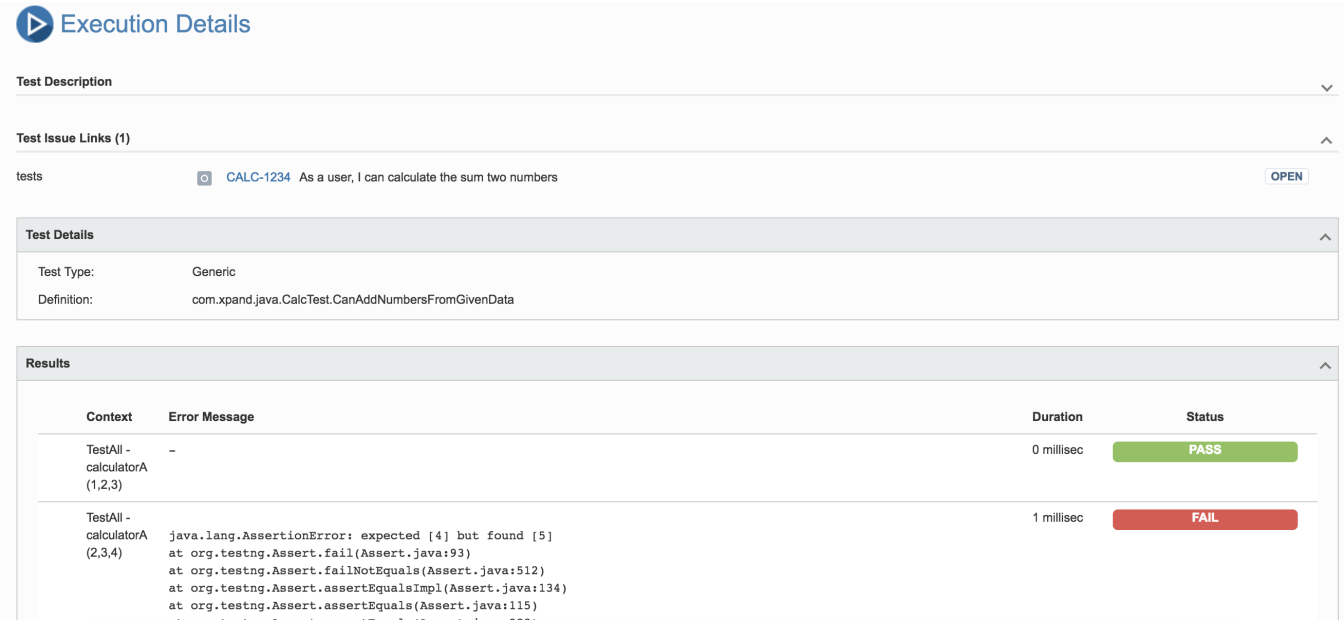
Status

The status of the Test Run will be set based on the Test case result:

Test Cases		Test status
Failed	**Error**	FAIL
Passed	**Success**	PASS
Other Cases	**Other Cases**	TODO

Note: Test cases with the status FAIL may have a failure message displayed in the Test Run screen, under the Results section.

If the same Test case has been executed multiple times, then the result for each context will be shown. A parameterized Test will also appear with its own specific contexts.



Whenever a Test Case is executed in multiple contexts, the overall status of the Test Run will be calculated as a joint value.

Condition	Overall status of the Test Run
If all the mapped results of the Test Case was PASS	**PASS**
If any of the mapped results of the Test Case was FAIL	**FAIL**

Other cases	**TODO**

References

- http://testng.org/doc/
- https://github.com/cbeust/testng
- http://testng.org/doc/documentation-main.html#logging-xml-reports