Taking advantage of JUnit XML reports

- About JUnit
- JUnit Basic Concepts
- Importing JUnit XML reports
    - Entities
    - Status
    - Xray extended JUnit format
        - Test issue id/key
        - Test requirements
        - Test summary
        - Test description
        - Test labels
        - Test run comment
        - Test run evidence
        - Test run custom fields
        - Test run start date
        - Test run finish date
- References

About JUnit

blocked URL

JUnit is a testing framework for Java, mostly focused for unit testing.

It is also used for writing integration and acceptance tests, making use of other libraries such as Selenium.

JUnit was massively used by the Java community and thus, its XML test result reports have become a de facto standard for test result reporting.

JUnit XML reports may be created by many different testing frameworks for Java, JavaScript, Ruby, Python, or any other language.

JUnit Basic Concepts

In JUnit, you have Tests and (Test) Suites. A Suite is a way of aggregating a group of tests, along with their results. This applies not just to the original Java's JUnit but also for other implementations that generate the JUnit XML report.

In Java, Tests are created within a Test Case class, which contain the Tests, implemented as class methods (and properly annotated).

The Test Case classes may be grouped in Test Suites.

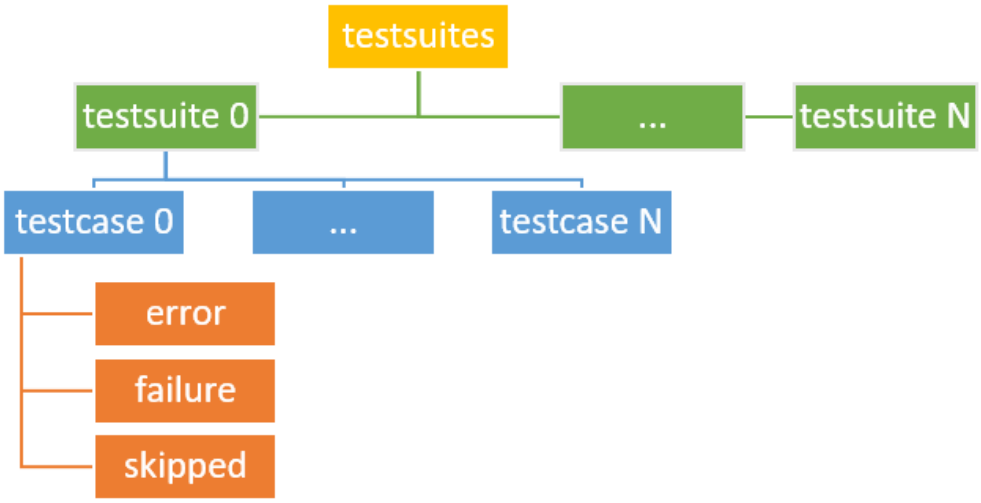
JUnit provides way more concepts (Test Runners, Test Fixtures, Categories, etc.) although they are not relevant in this context.

Importing JUnit XML reports

Below is a simplified example of a JUnit XML report containing a Test Suite with one Test Case.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <testcase classname="some.class.name" name="Test1" time="123.345000"/>
    </testsuite>
</testsuites>
```

The simplified tags hierarchy of these reports can be represented in the following diagram:



Entities

JUnit test cases are imported to Xray's ***Generic* test issues**, and the **`classname` and `name` attributes** are concatenated and mapped to the **Generic Test Definition** field of the Generic test.

Test cases can be imported to an already existing test by explicitly defining the test issue id or key or by a matching generic test definition (if a test already exists with the same generic test definition, then a duplicate will not be created).

If no test issue id or key is specified and no test exists with the same generic test definition, then a new test will be created.

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

JUnit's Test Suites are not mapped to any special entity. However, the execution details screen will show the Test Suite related to a specific test result.

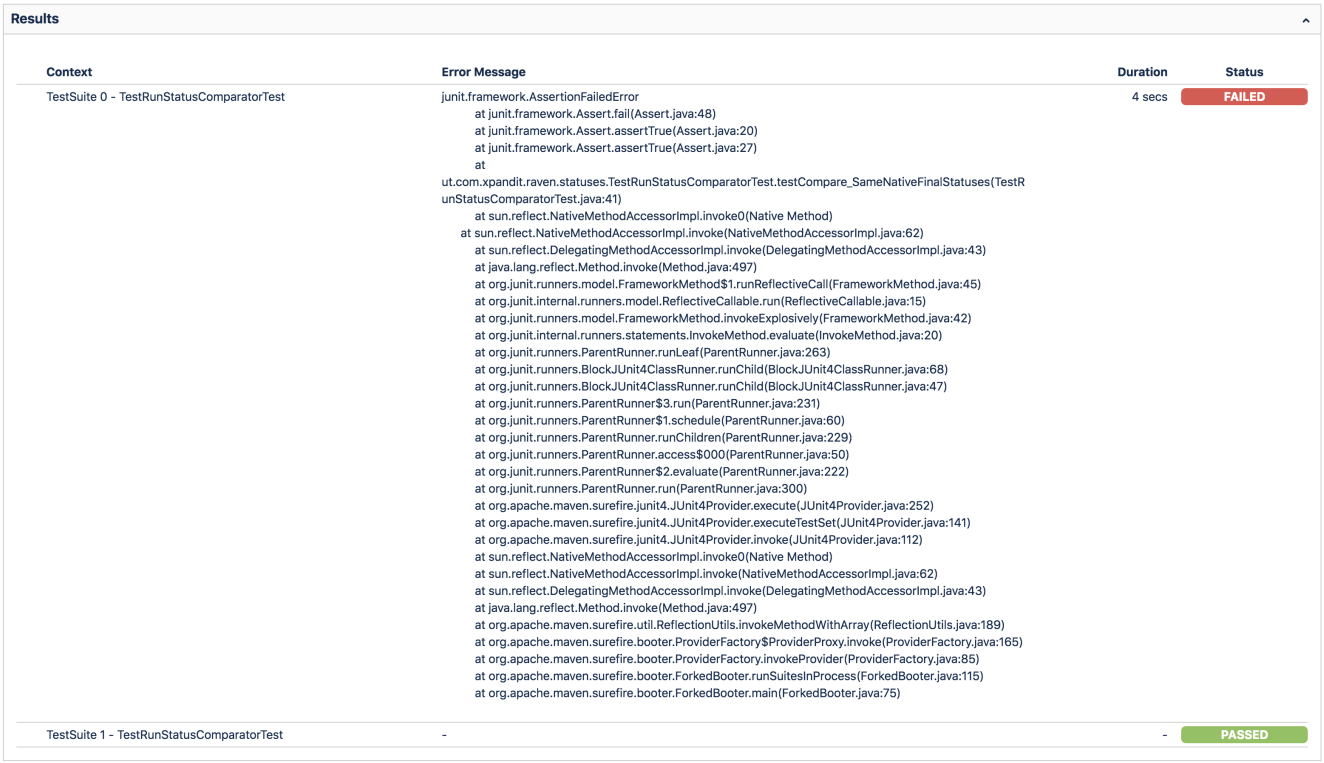
Status

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

Test Cases	Test status
with failures	FAIL
with errors	FAIL
skipped	TODO
without failures, errors, and weren't skipped	PASS

Note: Test Cases with the status FAIL may have an error/failure message, which can be seen in the Test Run screen, under the Results section.

If the same Test Case has been executed on multiple Test Suites, then the result for each Test Suite will be shown.



When a Test Case is executed in multiple Test Suites, 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 were PASS	**PASS**
If any of the mapped results of the Test Case was FAIL	**FAIL**
Other cases	**TODO**

Xray extended JUnit format

Although this is not yet included by JUnit, Xray supports a customized version of the JUnit report format that uses **custom properties/attributes** to import Xray related information.

Test issue id/key

Two scenarios are supported to specify an existing test to import the JUnit test case to:

- A test issue id is passed as a **`test_id` property** on the **`testcase`** element.
- A test issue key is passed as a **`test_key` property** on the **`testcase`** element.

If both properties exist in a `testcase`, the `test_id` will be used. If the given test issue id or key does not exist, an error will be thrown.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <!-- no test explicitly specified, will try to be matched using the default generic test definition -->
        <testcase classname="some.class.name" name="Test1" requirement="CALC-123" time="123.345000"/>

        <testcase classname="some.class.name" name="Test2" time="123.345000">
            <properties>
                                <!-- using a custom "test_id" property -->
                <property name="test_id" value="10001" />
            </properties>
        </testcase>

                <testcase classname="some.class.name" name="Test3" time="123.345000">
            <properties>
                                <!-- using a custom "test_key" property -->
                <property name="test_key" value="CALC-123" />
            </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test requirements

Three scenarios are supported to link a test with requirements:

- A requirement key is passed as a **`requirement` attribute** on the **`testcase`** element.
- A requirement key is passed on a **`requirement` property** element inside the **`testcase`** element.
- Multiple requirement keys are passed on a **`requirements` property** element inside the **`testcase`** element, separated by "," (comma).

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <!-- using a custom "requirement" attribute -->
        <testcase classname="some.class.name" name="Test1" requirement="CALC-123" time="123.345000"/>

        <testcase classname="some.class.name" name="Test2" time="123.345000">
            <properties>
                                <!-- using a custom "requirement" property -->
                <property name="requirement" value="CALC-123" />
            </properties>
        </testcase>

                <testcase classname="some.class.name" name="Test3" time="123.345000">
            <properties>
                                <!-- using a custom "requirements" property -->
                <property name="requirements" value="CALC-123,CALC-456" />
            </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test summary

Use a **`test_summary` property** element inside the **`testcase`** element to explicitly set the issue summary. This summary will be used both to create or update the test.

If importing to a new test and the summary is not explicitly defined, it will **default** to the **`name` attribute** of the **`testcase`**.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <!-- if a new test, summary will default to "Test1" -->
        <!-- if an existing test, summary will stay as is -->
        <testcase classname="some.class.name" name="Test1" time="123.345000"/>

        <testcase classname="some.class.name" name="Test2" time="123.345000">
            <properties>
                                <!-- if a new test, summary will be "A custom summary" -->
                                <!-- if an existing test, summary will be updated to "A custom summary" -->
                <property name="test_summary" value="A custom summary" />
            </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test description

Use a **`test_description` property** element inside the **testcase** element to set the issue description. This description will be used both to create or update the test.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <testcase classname="some.class.name" name="Test1" time="123.345000">
            <properties>
                                <property name="test_description">
                                        <![CDATA[Some custom description for the test issue.]]>
                                </property>
            </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test labels

Use a **`tags` property** element inside the **`testcase`** element to add labels to the issue. Multiple labels must be separated by "," (comma).

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <testcase classname="some.class.name" name="Test1" time="123.345000">
            <properties>
                                <property name="tags" value="label1,label2" />
            </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test run comment

Use a **`testrun_comment` property** element inside the **`testcase`** element to set the overall comment of the test run.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <testcase classname="some.class.name" name="Test1" time="123.345000">
                        <properties>
                                <property name="testrun_comment">
                                        <![CDATA[Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.]]>
                                </property>
                        </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test run evidence

Use a **`testrun_evidence` property** element inside the **`testcase`** element to add files as global evidence on the test run. Each evidence must be an **`item` element** inside the property, with the filename in the **`name` attribute** and with **Base64 encoded content**.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <testcase classname="some.class.name" name="Test1" time="123.345000">
                        <properties>
                                <property name="testrun_evidence">
                                        <item name="image1.png"
>iVBORw0KGgoAAAANSUhEUgAAADIAAAAyCAYAAAAeP4ixAAAAQ0lEQVR42u3PQREAAAgDINc
/9Mzg14MGZNrOAxERERERERERERERERERERERERERERERERERERERERERERERERERERERERERuVh9TJWdbWjzsQAAAABJRU5ErkJggg==</item>
                                        <item name="image2.png"
>iVBORw0KGgoAAAANSUhEUgAAADIAAAAyCAYAAAAeP4ixAAAAQ0lEQVR42u3PQREAAAgDINc
/9Mzg14MGZNrOAxERERERERERERERERERERERERERERERERERERERERERERERERERERERERERuVh9TJWdbWjzsQAAAABJRU5ErkJggg==</item>
                                </property>
                        </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test run custom fields

Two scenarios are supported to set test run custom fields:

- Each custom field in a **`testrun_customfield` property** element inside the **`testcase`** element. The name of the custom field must appear after the "testrun_customfield:" prefix in the **`name` attribute** and the value should be in the **`value` attribute**.
- Multiple custom fields in a **`testrun_customfields` property** element inside the **`testcase`** element. Each custom field should be an **`item` element** inside the property, with the custom field name in the **`name` attribute** and value in the **element content**.

In both scenarios, multiple select custom fields should have their values separated by ";" (semicolon).

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="1">
        <testcase classname="some.class.name" name="Test1" time="123.345000">
                        <properties>
                                <property name="testrun_customfield:cf1" value="lorem ipsum" />
                                <property name="testrun_customfield:cf2" value="option 1;option 2" /> <!--
multi select must have values separated by ";" -->
                        </properties>
        </testcase>

                <testcase classname="some.class.name" name="Test2" time="123.345000">
            <properties>
                <property name="testrun_customfields">
                                        <item name="cf1">
                                                <![CDATA[lorem ipsum]]>
                                        </item>
                                        <item name="cf2">
                                                <![CDATA[option 1;option 2]]> <!-- multi select must have
values separated by ";" -->
                                        </item>
                                </property>
            </properties>
        </testcase>
    </testsuite>
</testsuites>
```

Test run start date

Use a **`started-at` attribute** on the **`testcase`** element to set the start date of the test run.

The value should be a string representing a date in the ISO8601 date-time format. If the timezone offset is not included in the ISO string, it will default to UTC.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="3">
        <testcase classname="some.class.name" name="Test1" time="123.345000" started-at="2022-02-11T18:30:00.000
Z"> <!-- date-time in UTC -->
        </testcase>

        <testcase classname="some.class.name" name="Test2" time="123.345000" started-at="2022-02-11T20:30:00.000
+02:00"> <!-- date-time in a timezone offset of 2 hours ahead of UTC -->
        </testcase>

        <testcase classname="some.class.name" name="Test3" time="123.345000" started-at="2022-02-11T18:30:
00.000"> <!-- date-time with no timezone offset, will default to UTC -->
        </testcase>
    </testsuite>
</testsuites>
```

Test run finish date

Use a **`finished-at` attribute** on the **`testcase`** element to set the finish date of the test run.

The value should be a string representing a date in the ISO8601 date-time format. If the timezone offset is not included in the ISO string, it will default to UTC.

```
    <?xml version="1.0" ?>
<testsuites>
    <testsuite errors="0" failures="0" id="0" name="my test suite" tests="3">
        <testcase classname="some.class.name" name="Test1" time="123.345000" started-at="2022-02-11T18:30:00.000
Z"> <!-- date-time in UTC -->
        </testcase>

        <testcase classname="some.class.name" name="Test2" time="123.345000" started-at="2022-02-11T20:30:00.000
+02:00"> <!-- date-time in a timezone offset of 2 hours ahead of UTC -->
        </testcase>

        <testcase classname="some.class.name" name="Test3" time="123.345000" started-at="2022-02-11T18:30:
00.000"> <!-- date-time with no timezone offset, will default to UTC -->
        </testcase>
    </testsuite>
</testsuites>
```

References

- https://github.com/junit-team/junit4/wiki
- http://junit.org/junit4/
- https://www.tutorialspoint.com/junit/junit_basic_usage.htm
- https://www.relishapp.com/cucumber/cucumber/docs/formatters/junit-output-formatter