Squash TF Java Junit Runner Documentation

squashtest

Apr 01, 2020

Contents

CHAPTER 1

Runner Functions

The runner is transparent and should be able to run any Maven Junit 4 or 5 project **without any need to modify it**. Indeed the Mojos are implemented in such a way that they are fully autonomous and do not need any project specific configuration to run.

1.1 List implemented Junit tests

This Mojo enables one to list as a Json file the available implemented tests. In order to do so one only needs to run the following command :

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-
↪plugin:1.0.0-RELEASE:list
```

The command is structured as follows

- `mvn` : launch Maven
- `clean` : (Optional) One of Maven default goals that enables one to clean everything that has been previously build by Maven.
- `compile` : One of Maven default goals that enables one to compile code that is stored in src/main.
- `test-compile` : One of Maven default goals that enables one to compile code that is stored in src/test.
- `org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-plugin:1.0.0-RELEASE:list` : the actual listing Mojo provided by the **Squash TF** Java Junit Runner. It lists all Junit tests that can be discovered (in the Junit sense) in the code compiled during the "compile" and "test-compile" phases.

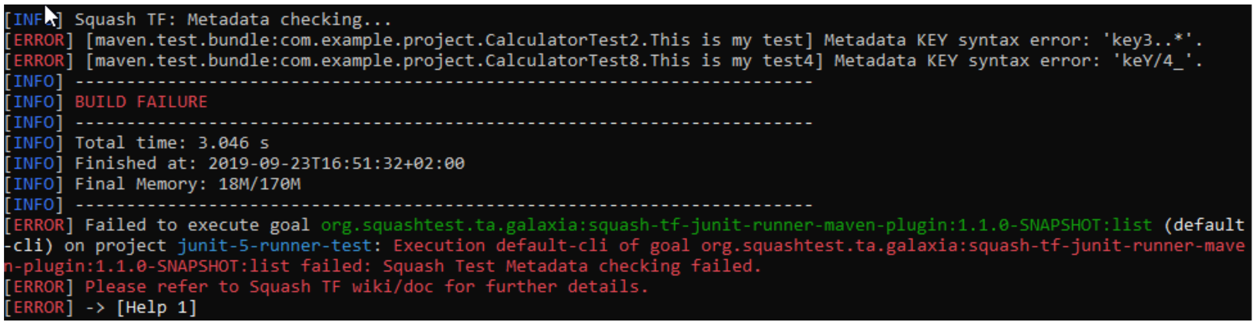
The result should be a Json file named testTree.json located at target/squashTA/test-tree. It is a simple JavaScript table listing available test grouped by "ecosystems". Ecosystem naming follows the convention explained in the *Junit tests reference scheme* section above.

1.1.1 'list' goal with Metadata

If there are metadata in the current test project, the goal **"list"** searches and checks if all metadata in this JUnit project respect the conventions for writing and using Squash TF metadata. (See *Metadata in JUnit runner* for more information about Metadata syntax conventions)

The goal will check through the project, collect all the metadata error(s) if any and lead to the *FAILURE*. Otherwise, a *SUCCESS* result will be obtained.

Metadata error(s), if found, will be grouped by test names.



```
[INFO] Squash TF: Metadata checking...
[ERROR] [maven.test.bundle:com.example.project.CalculatorTest2.This is my test] Metadata KEY syntax error: 'key3..*'.
[ERROR] [maven.test.bundle:com.example.project.CalculatorTest8.This is my test4] Metadata KEY syntax error: 'keY/4_'.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.046 s
[INFO] Finished at: 2019-09-23T16:51:32+02:00
[INFO] Final Memory: 18M/170M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-plugin:1.1.0-SNAPSHOT:list (default
-cli) on project junit-5-runner-test: Execution default-cli of goal org.squashtest.ta.galaxia:squash-tf-junit-runner-mave
n-plugin:1.1.0-SNAPSHOT:list failed: Squash Test Metadata checking failed.
[ERROR] Please refer to Squash TF wiki/doc for further details.
[ERROR] -> [Help 1]
```

1.1.2 Listing test JSON report with Metadata

If the build is successful, the generated report (JSON file) will contain the metadata associated with each of the test scripts.

```
{
        "timestamp": "2019-09-23T14:56:24.761+0000",
        "name": "tests",
        "contents": [{
                    "name": "maven.test.bundle:com.example.project.CalculatorTest2",
                    "contents": [{
                        "name": "testAdd()",
            "metadata": {},
            "contents": null
        }, {
            "name": "This is my test",
            "metadata": {
                "key1..": null
            },
            "contents": null
        }, {
            "name": "testAddWithTFMetadata2()",
            "metadata": {
                "key2": ["value2"]
            },
            "contents": null
        }, {
            "name": "testAddWithTFMetadata3()",
            "metadata": {
                "Key3-": null,
                "key_4": ["value4"],
```

(continues on next page)

(continued from previous page)

```
                "Key53": ["value4-","value4-"]
            },
            "contents": null
        }
    ]
}
```

Note: To ignore thoroughly metadata during the listing process as well as in the report, insert *tf.disableMetadata* property after the goal "list".

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-
↪plugin:1.0.0-RELEASE:list -Dtf.disableMetadata=true
```

or as a property in the **pom.xml** file

```
<properties>
        <tf.disableMetadata>true</tf.disableMetadata>
</properties>
```

1.2 Junit test Running

This Mojo enables one to run a selection of, or all possible, Junit tests and report their execution. In order to do so one only needs to run the following command :

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-
↪plugin:1.0.0-RELEASE:run
```

- `mvn` : launch Maven
- `clean` : (Optional) One of Maven default goals that enables one to clean everything that has been previously build by Maven.
- `compile` : One of Maven default goals that enables one to compile code that is stored in src/main.
- `test-compile` : One of Maven default goals that enables one to compile code that is stored in src/test.
- `org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-plugin:1.0.0-RELEASE:run` : the actual running Mojo provided by **Squash TF** Java Junit Runner. It runs the Junit tests that can be discovered (in the Junit sense) in the code compiled during the "compile" and "test-compile" phases.

By default the whole collection of tests available in the project will be executed. A summary of the execution is reported and available at target/squashTA/html-reports/squash-ta-report.html . A more detailed version of the report providing context in the case of technical error is also produced and available at target/squashTA/html-details/squash-ta-report.html. Finally a surfire report is also produced and available at target/squashTA/surefire-reports/ .

If one wants to only run a subset of possible test one can provide a list of tests via the Maven property "tf.test.suite". Two mechanism are possible:

- Mimic TM-TF link and provide a list of selected test via a Json file. In this scenario the tf.test.suite parameter should be given the value "{file:testsuite.json}" and the testsuite.json should be put right to the project pom.
- Provide a CSV like line, where qualified tests names are listed separated by semicolons

In both cases test convention should follow the one used by the listing Mojo and described in the *Junit tests reference scheme* section above.

Warning: If there are metadata syntax errors in the running test script(s), **warning** message(s) will be displayed in the console. (See *Metadata in JUnit runner* for more information about Metadata syntax conventions)

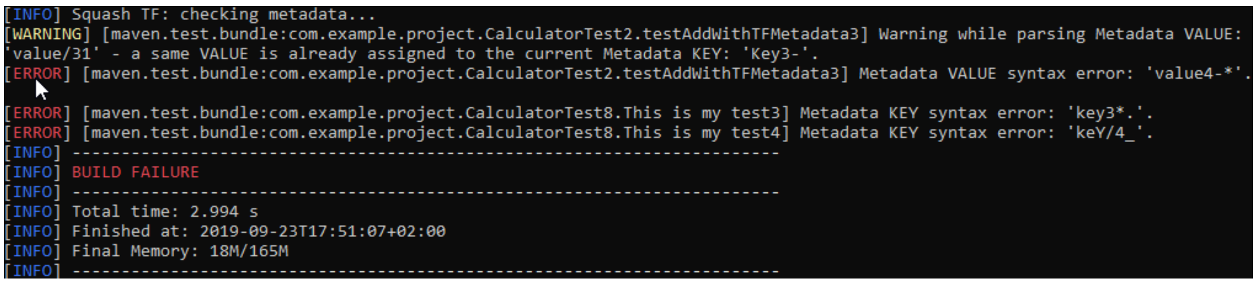
1.3 Junit test Metadata Checking

As the goal **"list"**, the goal **check-metadata** searches and checks if all metadata in this JUnit project respect the conventions for writing and using Squash TF metadata. (See *Metadata in JUnit runner* for more information about Metadata syntax conventions)

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-
↪plugin:1.1.0-RELEASE:check-metadata
```

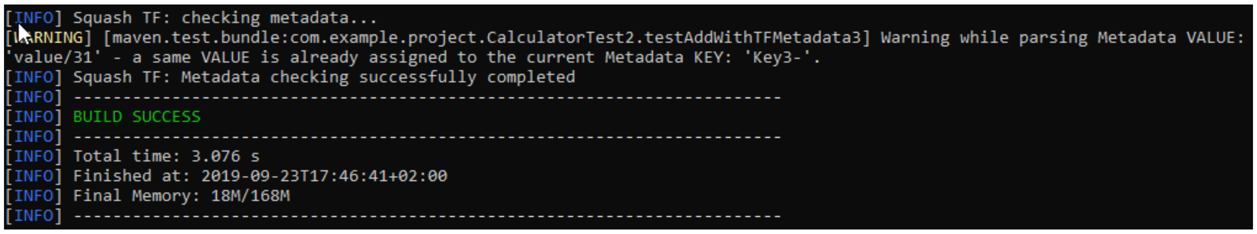
The goal will check through the project, collect all the metadata error(s) if any and lead to the *FAILURE*. Otherwise, a *SUCCESS* result will be obtained. (However, no JSON report will be created with a successful **check-metadata** goal.)

Metadata error(s), if found, will be grouped by test names.



```
[INFO] Squash TF: checking metadata...
[WARNING] [maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3] Warning while parsing Metadata VALUE:
'value/31' - a same VALUE is already assigned to the current Metadata KEY: 'Key3-'.
[ERROR] [maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3] Metadata VALUE syntax error: 'value4-*'.

[ERROR] [maven.test.bundle:com.example.project.CalculatorTest8.This is my test3] Metadata KEY syntax error: 'key3*.'.
[ERROR] [maven.test.bundle:com.example.project.CalculatorTest8.This is my test4] Metadata KEY syntax error: 'keY/4_'.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.994 s
[INFO] Finished at: 2019-09-23T17:51:07+02:00
[INFO] Final Memory: 18M/165M
[INFO] ------------------------------------------------------------------------
```

When a JUnit project has *duplicate values* in a *multi-value key* on a given test, the **check-metadata** goal will create a **WARNING** message in the console.



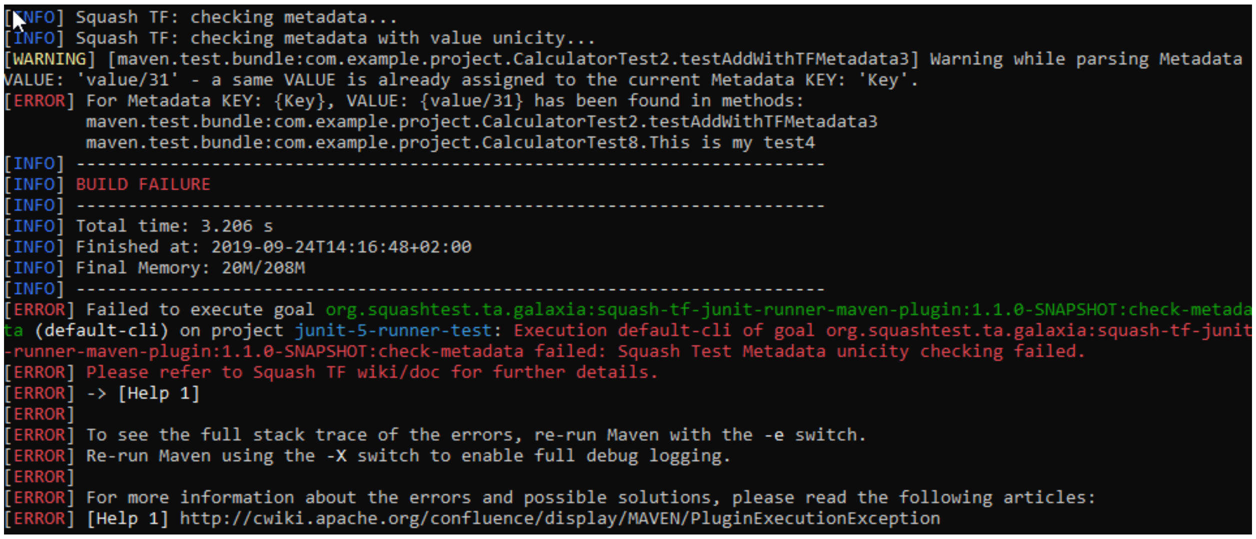
```
[INFO] Squash TF: checking metadata...
[WARNING] [maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3] Warning while parsing Metadata VALUE:
'value/31' - a same VALUE is already assigned to the current Metadata KEY: 'Key3-'.
[INFO] Squash TF: Metadata checking successfully completed
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.076 s
[INFO] Finished at: 2019-09-23T17:46:41+02:00
[INFO] Final Memory: 18M/168M
[INFO] ------------------------------------------------------------------------
```

1.3.1 'check-metadata' goal with Unicity checking

In addition to the normal syntax checking, you can insert the *tf.metadata.check* property after the goal "check-metadata" to check the unicity of each Metadata Key - Value pair.

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-
↪plugin:1.1.0-RELEASE:check-metadata -Dtf.metadata.check=[valueUnicity]
```

If there are metadata Key - Value duplicate(s) existed in the SKF project (even if the syntax is OK), a *FAILURE* result will be obtained.



```
[INFO] Squash TF: checking metadata...
[INFO] Squash TF: checking metadata with value unicity...
[WARNING] [maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3] Warning while parsing Metadata
VALUE: 'value/31' - a same VALUE is already assigned to the current Metadata KEY: 'Key'.
[ERROR] For Metadata KEY: {Key}, VALUE: {value/31} has been found in methods:
        maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3
        maven.test.bundle:com.example.project.CalculatorTest8.This is my test4
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.206 s
[INFO] Finished at: 2019-09-24T14:16:48+02:00
[INFO] Final Memory: 20M/208M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-plugin:1.1.0-SNAPSHOT:check-metada
ta (default-cli) on project junit-5-runner-test: Execution default-cli of goal org.squashtest.ta.galaxia:squash-tf-junit
-runner-maven-plugin:1.1.0-SNAPSHOT:check-metadata failed: Squash Test Metadata unicity checking failed.
[ERROR] Please refer to Squash TF wiki/doc for further details.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/PluginExecutionException
```

‘check-metadata’ goal with Unicity checking for specific Keys

You can even check the unicity of each metadata Key - Value pair with just some specific Keys by inserting the second property *tf.metadata.check.key* after the first one mentioned above.

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-
↪plugin:1.1.0-RELEASE:check-metadata -Dtf.metadata.check=[valueUnicity] -Dtf.
↪metadata.check.keys=[xxx,yyy,zzz]
```

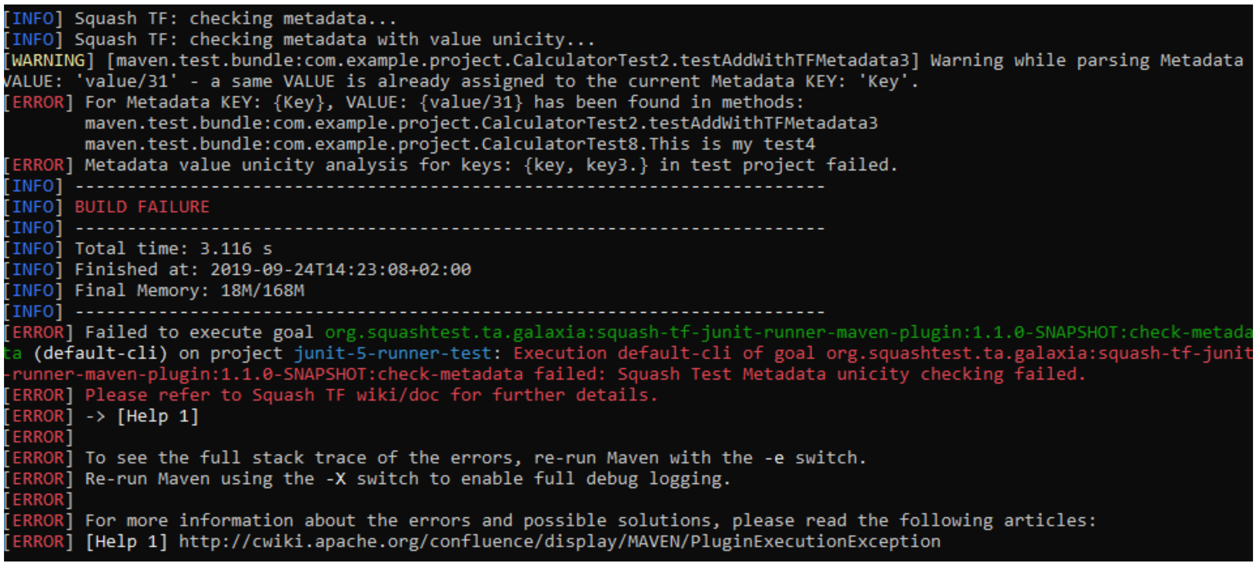
Important: In the bracket, the key list MUST be a string of characters composed by concatenation from 1 to n keys separated by commas: -Dtf.metadata.check.keys=[xxx,yyy,zzz]

If the list is surrounded by double quotes, spaces are allowed: -Dtf.metadata.check.keys=”[xxx, yyy, zzz]”

It is NOT allowed to have two commas without any key OR only spaces/tabulations between them. (ex: -Dtf.metadata.check.keys=”[xxx, ,yyy,,zzz]”)

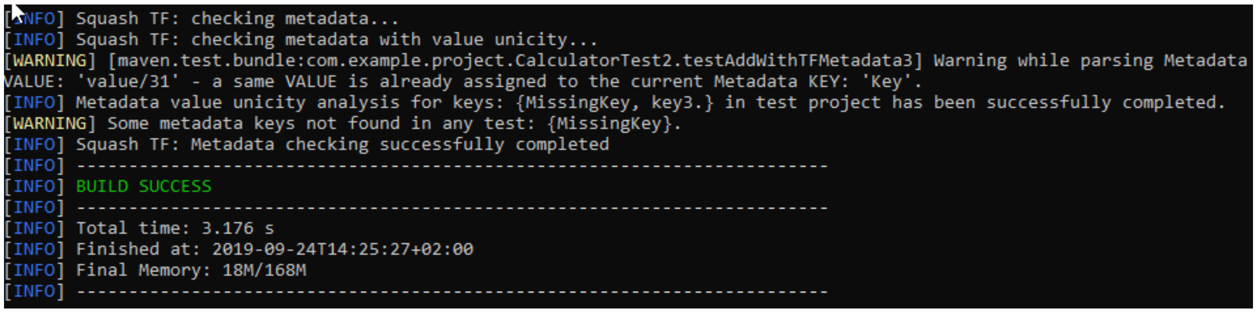
Key list is NOT allowed to be either uninitiated or empty. (ex: -Dtf.metadata.check.keys= OR -Dtf.metadata.check.keys=[])

For each searched metadata key, if there are Key - Value duplicate(s) existed in the SKF project, a *FAILURE* result will be obtained.



```
[INFO] Squash TF: checking metadata...
[INFO] Squash TF: checking metadata with value unicity...
[WARNING] [maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3] Warning while parsing Metadata
VALUE: 'value/31' - a same VALUE is already assigned to the current Metadata KEY: 'Key'.
[ERROR] For Metadata KEY: {Key}, VALUE: {value/31} has been found in methods:
        maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3
        maven.test.bundle:com.example.project.CalculatorTest8.This is my test4
[ERROR] Metadata value unicity analysis for keys: {key, key3.} in test project failed.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.116 s
[INFO] Finished at: 2019-09-24T14:23:08+02:00
[INFO] Final Memory: 18M/168M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-plugin:1.1.0-SNAPSHOT:check-metada
ta (default-cli) on project junit-5-runner-test: Execution default-cli of goal org.squashtest.ta.galaxia:squash-tf-junit
-runner-maven-plugin:1.1.0-SNAPSHOT:check-metadata failed: Squash Test Metadata unicity checking failed.
[ERROR] Please refer to Squash TF wiki/doc for further details.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/PluginExecutionException
```

Note: If searched metadata key(s) are not found in any Test files, a **WARNING** message will be raised in the console.



```
[INFO] Squash TF: checking metadata...
[INFO] Squash TF: checking metadata with value unicity...
[WARNING] [maven.test.bundle:com.example.project.CalculatorTest2.testAddWithTFMetadata3] Warning while parsing Metadata
VALUE: 'value/31' - a same VALUE is already assigned to the current Metadata KEY: 'Key'.
[INFO] Metadata value unicity analysis for keys: {MissingKey, key3.} in test project has been successfully completed.
[WARNING] Some metadata keys not found in any test: {MissingKey}.
[INFO] Squash TF: Metadata checking successfully completed
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.176 s
[INFO] Finished at: 2019-09-24T14:25:27+02:00
[INFO] Final Memory: 18M/168M
[INFO] ------------------------------------------------------------------------
```

CHAPTER 2

Junit tests reference scheme

2.1 Tests naming scheme

Junit tests detected in the project have to receive unique - and as far as possible readable - names in the runner system. Such names are used in the following contexts :

- Listing available tests
- Requiring the runner to run a subset of tests
- Linking **Squash TM** test cases to the junit test covering it
- Displaying tests results in execution reports

In this context, we define the qualified name of the Junit test as follows :

`bundle-name`**:**`qualified_class_name`**/**`Display Name`, where:

- `bundle-name` : name of the code bundle in which the tests are defined. The runner defines two such bundles :
  - `maven.main.bundle` : junit code from maven main sources (by default located in `the src/main/java` and `src/main/resources` subdirectories)
  - `maven.test.bundle` : junit code from maven test sources (by default located in `the src/test/java` and `src/test/resources` subdirectories)
- `qualified_class_name` : the dot-separated qualified name of the class where the test is defined, as specified by java conventions.
- `Display Name` : the display name of the test, as defined by junit. By default this is the name of the method that defines the test, but junit allows you to override this using the `@DisplayName` annotation.

This test name appears in test lists for the **TM-TF** link, they may also be used to select tests to run from the command line (see below).

There is a small difference in test execution reports : tests are grouped in test groups named `ecosystems`. These groups are defined like the prefix we add to the Display Name to make qualified test names : according to the code bundle and test class. Ecosystems are named accordingly, but the separator between the bundle and class parts is a dot instead of a colon :

`bundle-name.qualified_class_name`

In these reports, tests appear under a qualified test result name where the colon and slash separators are also replaced by a dot :

`bundle-name.qualified_class_name.Display Name`

2.2 Tests with the same `@displayName`

The use of the `@displayName` junit annotation allows you to give two tests from the same class the same unqualified display name. And as these tests come from the same class, in the same bundle, this will given them the same **qualified** display name through the runner.

We consider this **bad practice** and **strongly advise against it**, even if you don't use our runner. Giving two tests the same display name will tend to make reports harder to read: if two or more tests cover exactly the same case, then why use time and work to write more than one test, and read more than one result ? On the other hand, if they test different things you should make yourself a favor and name them differently, to avoid confusion.

If, however, you have inherited such tests, or you have your own reason to go against this advice, the runner will work with this tests in the following way :

- All tests with the same qualified display name will be seen as a single test, and listed and executed as such.
  - All junit tests will be executed and checked as a global unit which result will be the worst result of any test.
- You will be able to link this global functional test defined by its common qualified display name to **Squash TM** test cases as any other automated test, execute the test case and see the result from **Squash TM**

Known limitation : if several tests share the same qualified display name, the runner will not handle metadata declaration (defined through the use of **Squash TF**'s `@Metadata` annotations on the methods) properly. We intend to fix this in a later version.

CHAPTER 3

Metadata in JUnit runner

In your JUnit project, you can insert Squash Metadata into test methods via an Annotation named *TFMetadata*. For example:

```
@Test
@DisplayName("This is my test with Squash Metadata")
@TFMetadata(key ="key", value ={"value"})
public void testAddWithTFMetadata1() {
    Calculator calculator = new Calculator();
    assertEquals(2, calculator.add(1, 1), "1 + 1 should equal 2");
}
```

3.1 Configuration : new dependency

In order to be able to use TF metadata in your Junit project, you have to add a new dependency to your project.

To do so :

1. Add to the `dependencies` section of your project pom.xml the following dependency :

```
<dependency>
  <groupId>org.squashtest.ta.galaxia</groupId>
  <artifactId>squash-tf-galaxia-annotations</artifactId>
  <version>1.0.0-RELEASE</version>
</dependency>
```

2. Add to the `repositories` section (or create it if doesn't exist) of your project pom.xml the following repository. (This is needed because our dependency is not available on maven central but only in our repository)

```
<repositories>
  <repository>
    <id>org.squashtest.ta.release</id>
    <name>squashtest test automation - releases</name>
```

(continues on next page)

(continued from previous page)

```xml
    <url>http://repo.squashtest.org/maven2/releases</url>
  </repository>
</repositories>
```

3.2 Metadata syntax conventions

In a Metadata annotation, the **key** is mandatory. A metadata key MUST be ONE WORD which contains only *alphanumeric characters, dashes, underscores and dots*. Spaces/tabulation are allowed before and after the word.

Moreover, metadata key is **case insensitive** and must be **unique** in a test file, even a test method.

```java
@Test
@DisplayName("For a typical Metadata Key")
@TFMetadata(key ="   Key.01-is_Insensitive-and_MUST-be_UNIQUE      ", value ={"value"}
↪)
public void testAddWithTFMetadata2() {
    Calculator calculator = new Calculator();
    assertEquals(3, calculator.add(1, 2), "1 + 2 should equal 3");
}
```

On the other hand, the **value** is optional. However, a Metadata value, if it exists, MUST be ONE WORD containing only *alphanumeric characters, dashes, slashes, underscores and dots*. Spaces/tabulation are also allowed before and after the word.

Metadata value is **case sensitive** and must be assigned to a metadata key. It is also possible to have many metadata values associated to a same key.

```java
@Test
@TFMetadata(key ="key1")
public void testAddWithTFMetadata3() {
    Calculator calculator = new Calculator();
    assertEquals(4, calculator.add(1, 3), "1 + 3 should equal 4");
}

@Test
@TFMetadata(key ="key2", value ={})
public void testAddWithTFMetadata4() {
    Calculator calculator = new Calculator();
    assertEquals(5, calculator.add(1, 4), "1 + 4 should equal 5");
}

@Test
@TFMetadata(key ="key3", value ={"   pathTo/Value.03-is_Sensitive      "})
public void testAddWithTFMetadata5() {
    Calculator calculator = new Calculator();
    assertEquals(6, calculator.add(1, 5), "1 + 5 should equal 6");
}

@Test
@TFMetadata(key ="key4", value ={"value1", "Value2", "value3"})
public void testAddWithTFMetadata6() {
    Calculator calculator = new Calculator();
    assertEquals(7, calculator.add(1, 6), "1 + 6 should equal 7");
}
```

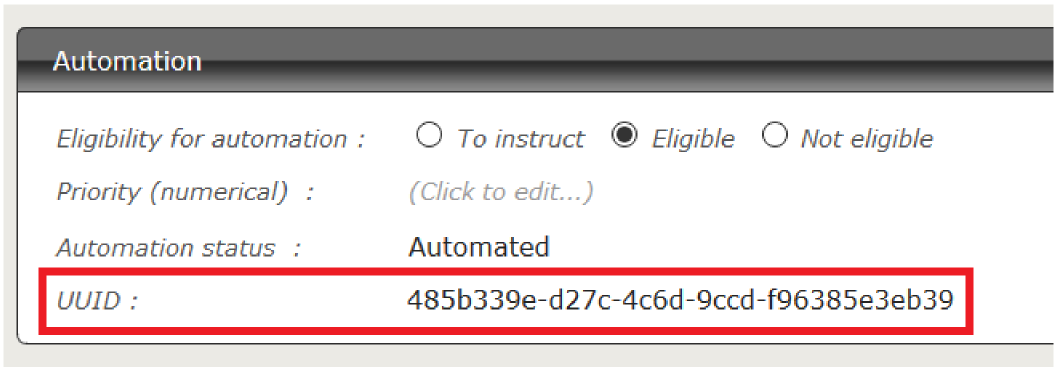
A test method can be associated with from zero to many Metadata.

```
@Test
@DisplayName("No metadata")
public void testAddWithTFMetadata7() {
    Calculator calculator = new Calculator();
    assertEquals(8, calculator.add(1, 7), "1 + 7 should equal 8");
}

@Test
@DisplayName("With one metadata")
@TFMetadata(key ="key", value ={"value"})
public void testAddWithTFMetadata8() {
    Calculator calculator = new Calculator();
    assertEquals(9, calculator.add(1, 8), "1 + 8 should equal 9");
}

@Test
@DisplayName("With many metadata")
@TFMetadata(key ="key1", value ={})
@TFMetadata(key ="key2", value ={"value"})
@TFMetadata(key ="key3", value ={"value1", "value2"})
public void testAddWithTFMetadata9() {
    Calculator calculator = new Calculator();
    assertEquals(10, calculator.add(1, 9), "1 + 9 should equal 10");
}
```

Important: Please ensure that all Metadata keys in every JUnit method of a Test script are unique.

3.3 Use metadata for TM - TF autolink

TF metadata handles the TM - TF autolink. (TM - TF autolink is available since **TM 1.20.0** and **Java Junit Runner 1.1.0**) Autolink is a feature to ease the link between a TM test case and a test automation script. On TM side, a UUID is now provided (when the workflow is activated) :



Eligibility for automation : ○ To instruct [✓ Eligible] ○ Not eligible

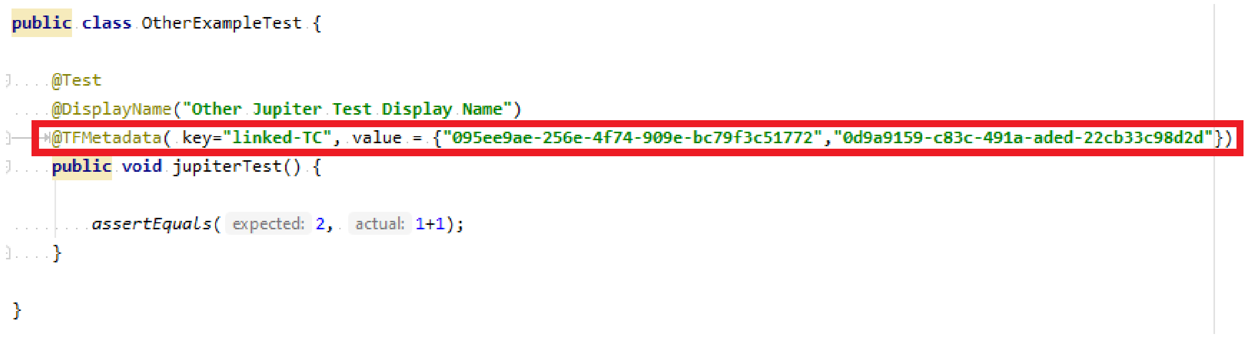
Priority (numerical) : (Click to edit...)

Automation status : Automated

UUID : 485b339e-d27c-4c6d-9ccd-f96385e3eb39

This UUID is used as an identifier.

In your automation test add a TF Metadata which key is `linked-TC` and value is the UUID from the corresponding TM test case. As you can see in the example below, it's possible to link many TM test case to the same automation test (two UUID are set in the "value"):



```
public class OtherExampleTest {

    @Test
    @DisplayName("Other Jupiter Test Display Name")
    @TFMetadata( key="linked-TC", value = {"095ee9ae-256e-4f74-909e-bc79f3c51772","0d9a9159-c83c-491a-aded-22cb33c98d2d"})
    public void jupiterTest() {

        assertEquals( expected: 2, actual: 1+1);
    }

}
```

CHAPTER 4

TF Param Service

This service allows you to retrieve the values of the parameters that are present in a **.json** file in order to use them in your JUnit test.

Warning: TF Param Service will only work properly with 1.2.0-RELEASE (or newer) version of TF JUnit Runner.

4.1 Configuration

In order to be able to use TF param Service in your JUnit project, you need to add a new dependency to your project.

Add the following dependency to the pom.xml of your project :

```
<dependency>
    <groupId>org.squashtest.ta.galaxia</groupId>
    <artifactId>tf-param-service</artifactId>
    <version>1.0.0-RELEASE</version>
</dependency>
```

4.2 Call the service

If you want to use methods of the service in your JUnit test, you first need to call it. To do so, write the following:

```
TFParamService.getInstance().[methodName()];
```

You will need to write this everytime you want to use the service.

4.3 Available methods

- **getTestParam(String paramName) :** Returns a String of the value associated with the "paramName" parameter in the params section of the test in the supplied **.json** file. Returns << null >> if it can not be found.
- **getTestParam(String paramName, String defaultValue) :** Returns a String of the value associated with the "paramName" parameter in the params section of the test in the supplied **.json** file. Returns defaultValue if it can not be found.
- **getParam(String paramName) :** Search the "paramName" parameter in the test params section of the test in the supplied **.json** file. If the service does not find it, it looks in the global-params section. Returns a String of the result of its search or << null >> if nothing was found.
- **getParam(String paramName, String defaultValue) :** Search the "paramName" parameter in the test params section of the test in the supplied **.json** file. If the service does not find it, it looks in the global-params section. Returns a String of the result of its search or defaultValue if nothing was found.
- **getGlobalParam(String paramName) :** Returns a String of the value associated with the "paramName" parameter in the global-params section of the supplied **.json** file. Returns << null >> if it can not be found.
- **getGlobalParam(String paramName, String defaultValue) :** Returns a String of the value associated with the "paramName" parameter in the global-params section of the supplied **.json** file. Returns the defaultValue if it can not be found.

4.4 Manually provide .json file

If you want to manually provide the **.json file**, you need to add the following parameter **-Dtf.test.suite={file:path/to/json/FileName.json}** to your maven goal.

"path/to/json/FileName.json" must be the relative path of your **.json file** from the root of your project.

If the **.json file** is located directly at the root of your project, just type **-Dtf.test.suite={file:FileName.json}**

For example :

```
mvn clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-
↪runner-maven-plugin:1.2.0-RELEASE:run -Dtf.test.suite={file:testSuite.json}
```

Example of .json file :

```
{
    "test": [{
            "id": "39",
            "script": "maven.test.bundle:org.squashtest.tf.other.OtherExampleTest/
↪Other Jupiter Test Display Name",
            "param": {
                "TC_REFERENCE": "",
                "TC_CUF_CUF_CUSTOM": "true",
                "TC_UUID": "3a7099ff-ab59-4e99-b21d-07e7d71d1ed5"
            }
        }, {
            "id": "40",
            "script": "maven.test.bundle:org.squashtest.tf.other.OtherExampleTest/
↪Other Jupiter Test Display Name",
            "param": {
                "TC_REFERENCE": "",
                "DS_name": "Bertrand",
```

(continues on next page)

(continued from previous page)

```
                "DSNAME": "dataset1",
                "TC_CUF_CUF_CUSTOM": "true",
                "DS_age": "41",
                "TC_UUID": "adec6164-5dec-4c8c-a0ae-c036340d519b"
            }
        }, {
            "id": "41",
            "script": "maven.test.bundle:org.squashtest.tf.other.OtherExampleTest/
↪Other Jupiter Test Display Name",
            "param": {
                "TC_REFERENCE": "",
                "DS_name": "Damien",
                "DSNAME": "dataset2",
                "TC_CUF_CUF_CUSTOM": "true",
                "DS_age": "undefined",
                "TC_UUID": "adec6164-5dec-4c8c-a0ae-c036340d519b"
            }
        }],
    "param": {
        "globalParamSection": "This is global param section",
        "user": "foo",
        "ow_ner.Na-me": "bar",
    }
}
```

CHAPTER 5

Creating projects

5.1 Starting a new project

In case you are starting a new Maven Java Junit project from scratch, a Maven archetype is provided to help you generate a correctly structured project with recommended dependencies. The archetetype group Id is `org.squashtest.ta.galaxia`, the archetype artifact Id is `squash-tf-junit-runner-project-achetype` and the current version is `1.0.0-RELEASE`. Hence, to use it simply run the following command :

```
mvn archetype:generate -DarchetypeGroupId=org.squashtest.ta.galaxia -
↪DarchetypeArtifactId=squash-tf-junit-runner-project-archetype -DarchetypeVersion=1.
↪0.0-RELEASE
```

5.2 Tests implementation: sample with SoapUI Smartbear Api

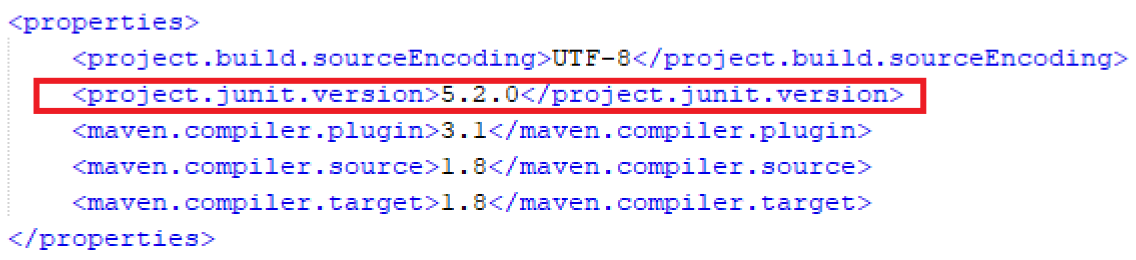
Once your project is created, you just have to implement your tests as Junit tests according to the version of Junit chosen.

Here is an example to run a REST test with the open source API proposed by SoapUI Smartbear.

We used:

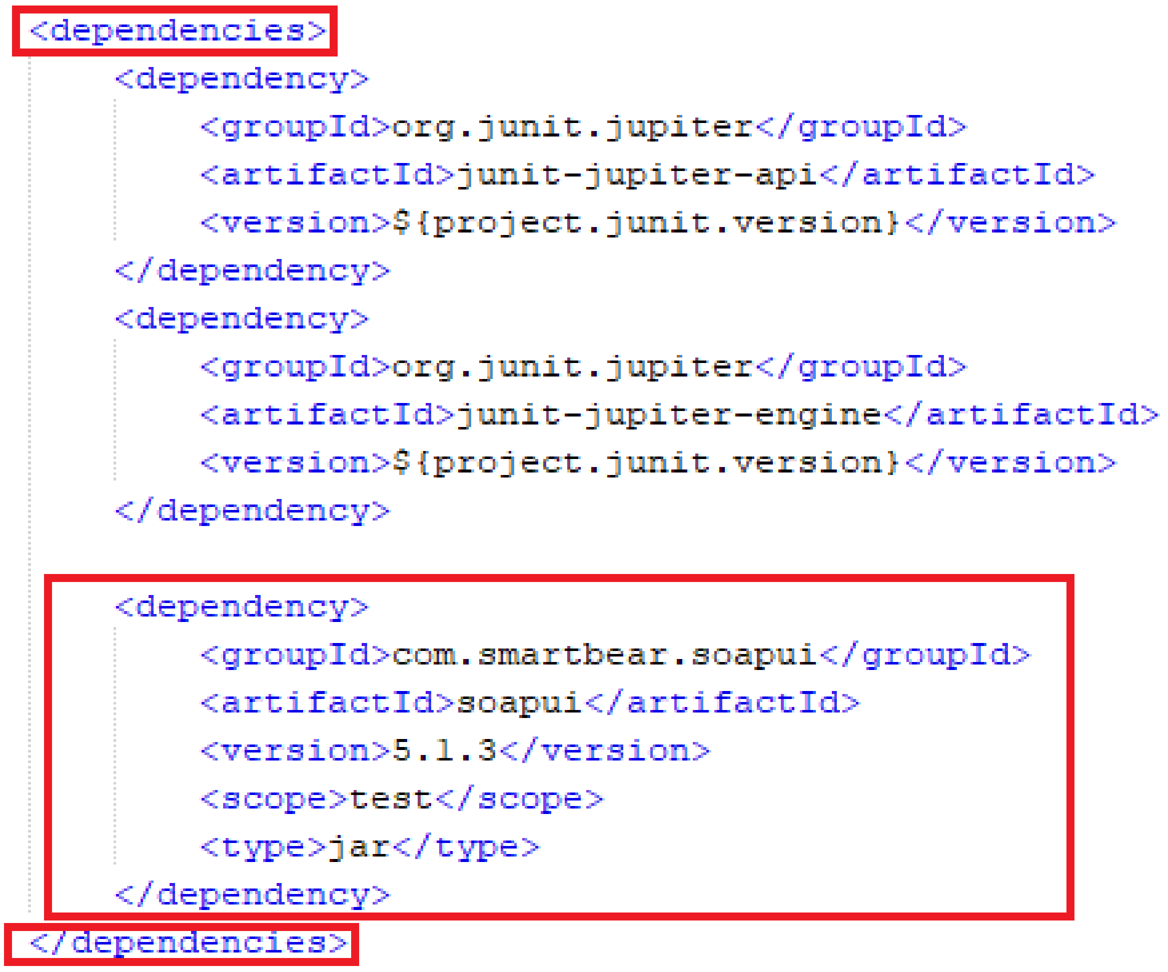
- a REST test given in the tutorial installed with version 5.5.0 of SoapUI Smartbear software.
- the open source SoapUI Smartbear API version 5.1.3
- the TF JUnit runner version 1.0.0-RELEASE
- junit (jupiter) version 5.2.0

We have set in the POM's project the version of junit chosen:



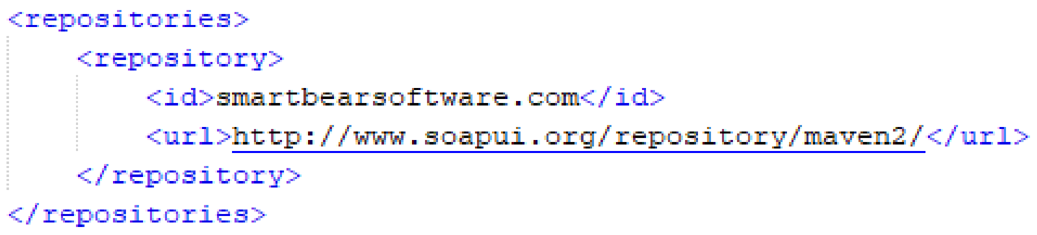
```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.junit.version>5.2.0</project.junit.version>
    <maven.compiler.plugin>3.1</maven.compiler.plugin>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

You need to add the external dependency the SoapUI API in your POM file as follows:



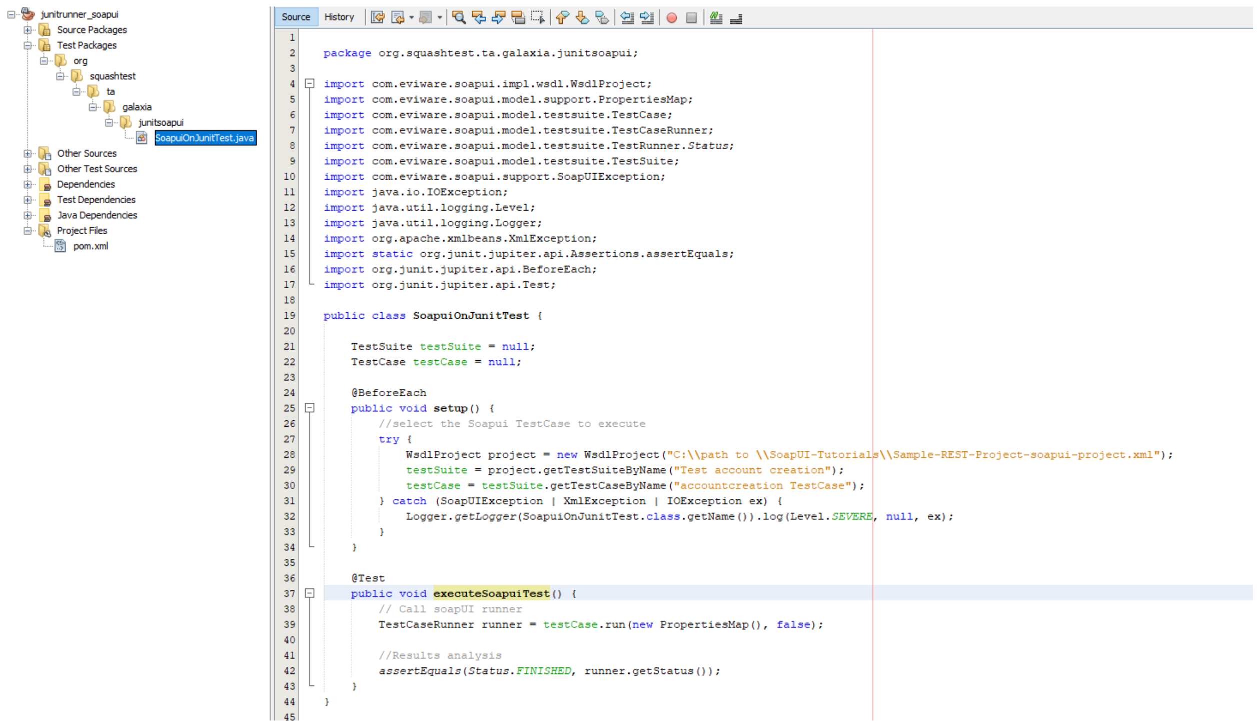
```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${project.junit.version}</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${project.junit.version}</version>
    </dependency>

    <dependency>
        <groupId>com.smartbear.soapui</groupId>
        <artifactId>soapui</artifactId>
        <version>5.1.3</version>
        <scope>test</scope>
        <type>jar</type>
    </dependency>
</dependencies>
```

In this example, you must also add in your POM a block for the Smartbear repository:



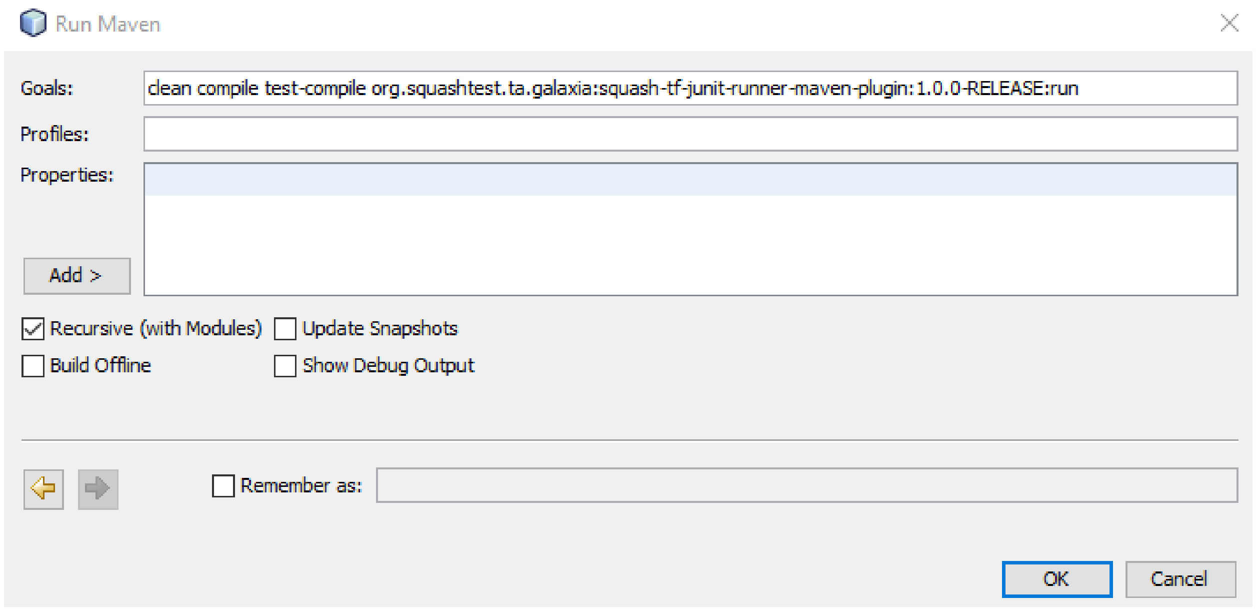
```
<repositories>
    <repository>
        <id>smartbearsoftware.com</id>
        <url>http://www.soapui.org/repository/maven2/</url>
    </repository>
</repositories>
```

Create your test class and implement it. Your project now looks like this:



```java

package org.squashtest.ta.galaxia.junitsoapui;

import com.eviware.soapui.impl.wsdl.WsdlProject;
import com.eviware.soapui.model.support.PropertiesMap;
import com.eviware.soapui.model.testsuite.TestCase;
import com.eviware.soapui.model.testsuite.TestCaseRunner;
import com.eviware.soapui.model.testsuite.TestRunner.Status;
import com.eviware.soapui.model.testsuite.TestSuite;
import com.eviware.soapui.support.SoapUIException;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.xmlbeans.XmlException;
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class SoapuiOnJunitTest {

    TestSuite testSuite = null;
    TestCase testCase = null;

    @BeforeEach
    public void setup() {
        //select the Soapui TestCase to execute
        try {
            WsdlProject project = new WsdlProject("C:\\path to \\SoapUI-Tutorials\\Sample-REST-Project-soapui-project.xml");
            testSuite = project.getTestSuiteByName("Test account creation");
            testCase = testSuite.getTestCaseByName("accountcreation TestCase");
        } catch (SoapUIException | XmlException | IOException ex) {
            Logger.getLogger(SoapuiOnJunitTest.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Test
    public void executeSoapuiTest() {
        // Call soapUI runner
        TestCaseRunner runner = testCase.run(new PropertiesMap(), false);

        //Results analysis
        assertEquals(Status.FINISHED, runner.getStatus());
    }
}

```

Launch the TF junit runner:



Goals: clean compile test-compile org.squashtest.ta.galaxia:squash-tf-junit-runner-maven-plugin:1.0.0-RELEASE:run

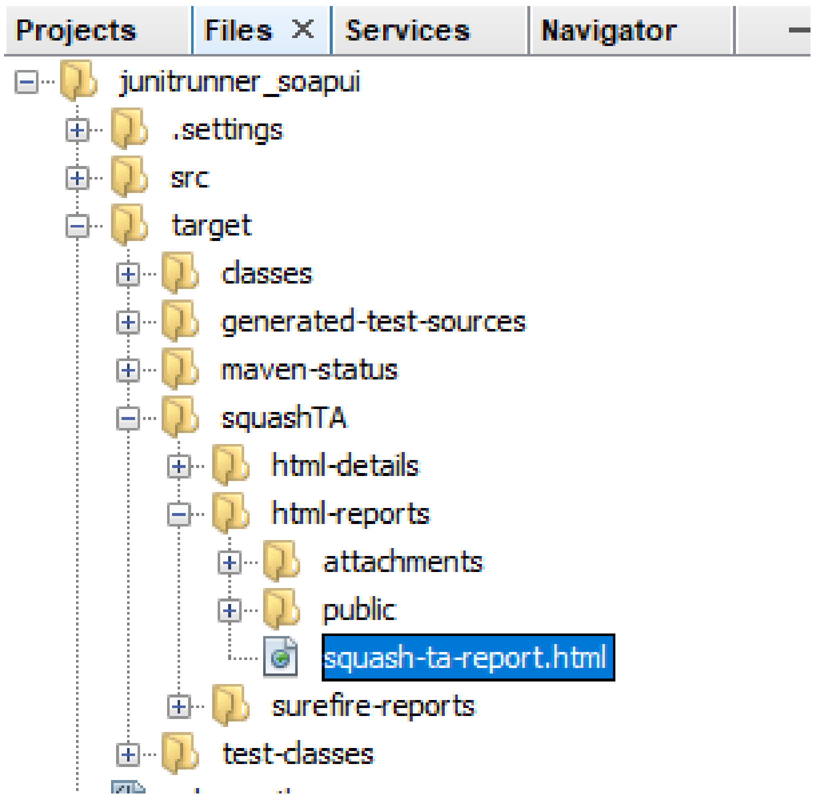
Profiles:

Properties:

[✓] Recursive (with Modules)
[ ] Update Snapshots
[ ] Build Offline
[ ] Show Debug Output

[ ] Remember as:

The HTML results file of the execution `squash-ta-report.html` is available in the subdirectory squashTA/html-report of the target folder.



Note: If you have not started the server mock, as indicated in the SoapUI tutorial, your test will fail on a connection error.

CHAPTER 6

Overview

The **Squash T**est **F**actory (**Squash TF**) Java Junit Runner aims to provide a seamless integration to our ecosystem when automated test are already (will be) implemented using Java as a language and Junit as the underlying test framework.

As a **Squash TF** runner its main goal is twofold :

- List in JavaScript Object Notation (Json) format the available implemented tests
- Run a selection (that can include all available tests) and report the execution. In the case where the execution order originates from **Squash T**est **M**anagement (**Squash TM**), test status and report are sent back to **Squash TM**.

The actual implementation of the Runner is based on the Maven technology. More precisely each major feature ("List" and "Run") is provided via an implementation of a Maven Mojo.