Squash Autom & Squash Devops

Release 1.0.0-alpha1

squashtest

Mar 25, 2021

CONTENTS

Squash AUTOM is a set of components for the management of your automated tests' executions.

Squash DEVOPS is a set of components for the integration to your continuous integration pipeline of your automated functional tests' executions.

CHAPTER

ONE

SQUASH AUTOM

1.1 Installation Guide

1.1.1 Squash Orchestrator

The **Squash Orchestrator** is available as *Docker* image on *DockerHub* (*squashtest/squash-orchestrator:1.0.0.alpha2*).

The deployment procedure can be found in the **Squash Orchestrator** documentation (*Squash Orchestrator Documentation – 1.0.0.alpha2*, `.pdf` version) downloadable at https://www.squashtest.com/community-download.

1.1.2 Result Publisher Plugin for Squash TM

The plugin exists in a **Community** version (*squash.tm.rest.result.publisher.community-1.0.0.alpha2.jar*) freely available, or a **Premium** version (*squash.tm.rest.result.publisher.premium-1.0.0.alpha2.jar*) available on request.

For details on the installation, please refer to the installation protocol of a **Squash TM** plugin (https://sites.google.com/a/henix.fr/wiki-squash-tm/installation-and-exploitation-guide/2—installation-of-squash-tm/7—jira-plug-in).

Warning: This plugin is compatible with version *1.22.2.RELEASE* of **Squash TM**.

1.2 Piloting automated tests executions with an EPAC (Execution Plan «as code»)

Squash AUTOM allows the redaction of an execution plan in a format specific to the **Squash Orchestrator**, the EPAC (Execution Plan «as code»), in order to orchestrate with precision the execution of automated tests outside of a test repository.

You can find more information regarding the redaction of an EPAC in the **Squash Orchestrator** documentation (*Squash Orchestrator Documentation – 1.0.0.alpha2*, `.pdf` version) downloadable from https://www.squashtest.com/community-download.

1.3 Piloting automated tests executions from Squash TM

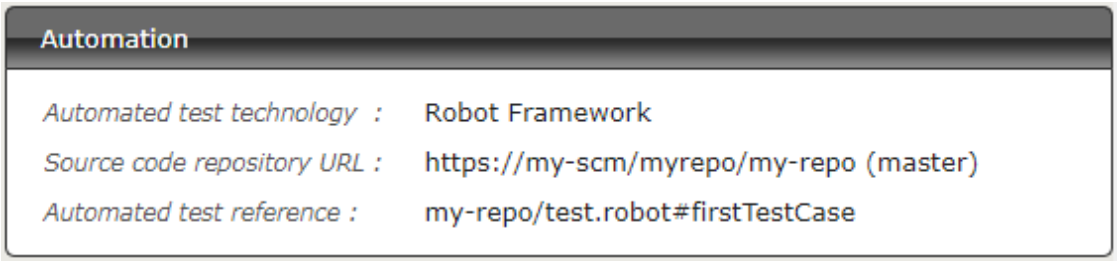
1.3.1 Squash TM test case automation

Note: This page describes the common operations to all supported test frameworks in this version. You can access the automation specifics for each technology directly with the following links :

- *Cucumber*
- *Cypress*
- *JUnit*
- *Robot Framework*
- *SoapUI*

Without using the Squash automation workflow

For a test case to be usable by the **Squash Orchestrator**, its *Automation* panel in the *Information* tab of the test case page must be correctly filled :

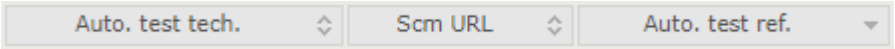


- `Automated test technology` : A dropdown list allowing you to choose the technology used for the execution of a test case. In this version, only *Robot Framework*, *Junit*, *Cucumber*, *Cypress* and *SoapUi* are functioning.
- `Source code repository URL` : The address of the source code repository where the project is located, as referenced in the *Source code management servers* area of the *Administration*.
- `Automated test reference` : This is the location of the automated test within the project. This reference must follow the format specific to the test technology being used (see *here*).

Using the Squash automation workflow

Regular test case

For a test case to be usable by the **Squash Orchestrator**, it must be automated in the *Automation Workspace* by filling three columns :



- `Auto. test tech.` : A dropdown list allowing you to choose the technology used for the execution of a test case. In this version, only *Robot Framework* and *Junit* are functioning.
- `Scm URL` : The address of the source code repository where the project is located.
- `Auto. test ref.` : This is the location of the automated test within the project. This reference must follow the format specific to the test technology used (see *here*).

BDD or Gherkin test case

The information of the *Automation* panel is automatically filled during the transmission of a BDD or Gherkin script to a remote source code repository hosting service. It can also be modified by the user at any moment.

Squash TM parameters exploitation

When a **Squash TM** execution plan is launched (through an EPAC or directly from the campaign workspace), **Squash TM** will transmit various information on ITPI that can be exploited by a *Cucumber*, *Cypress*, or *Robot Framework* test case. Details of this functionality can be found on the corresponding used technology section

---

Automation frameworks specifics

Automation with Cucumber

1. Test reference

Note: In this version of **Squash AUTOM**, it is not possible to select a specific scenario in a `.feature` file containing several ones : every scenario in the file are therefore executed together. The result of each executed **Squash TM** test case is calculated by taking into account the individual results of each scenario included in the bound file :

- If at least one scenario has an *Error* status (in case of a technical issue), the status of the execution will be *Blocked*.
- If at least one scenario fails functionally and none of the other has an *Error* status, the status of the execution will be *Failed*.
- If all scenarios succeed, the status of the execution will be *Success*.

In order to bind a **Squash TM** test case to a *Cucumber* automated test, the content of the *Automated test reference* field of the *Automation* panel of a test case must have the following format :

```
[1] / [2]
```

With :

- `[1]` : Name of the project on the source code repository.
- `[2]` : Path and name of the *Cucumber* test file, from the root of the project (with the `.feature` extension).

2. Nature of the exploitable Squash TM parameters

Squash AUTOM and **Squash DEVOPS** are able to use the name of a **Squash TM** dataset as a tag value to use for the execution of a specific subset of a *Cucumber* feature.

Both **Community** and **Premium** versions can use dataset names.

3. Squash TM parameters usage

When executing a **Squash TM** automated test case with *Cucumber*, it is possible to exploit the **Squash TM** dataset name in order to execute a specific dataset of a *Cucumber* scenario.

In order to achieve this, you'll have to follow these steps :

- Fill the datasets in the *Parameters* tab of the test case in **Squash TM**.
- Create in a *Cucumber* scenario as many example table as there are dataset in **Squash TM** test case. Annotate them with a tag corresponding to the name of a **Squash TM** dataset.

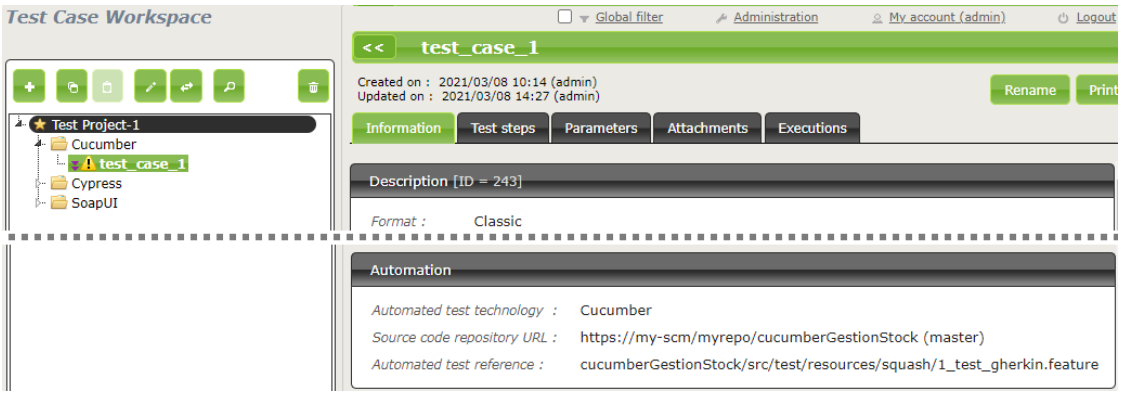
- Create one line of elements in each example table to set scenario's parameters values for the dataset.

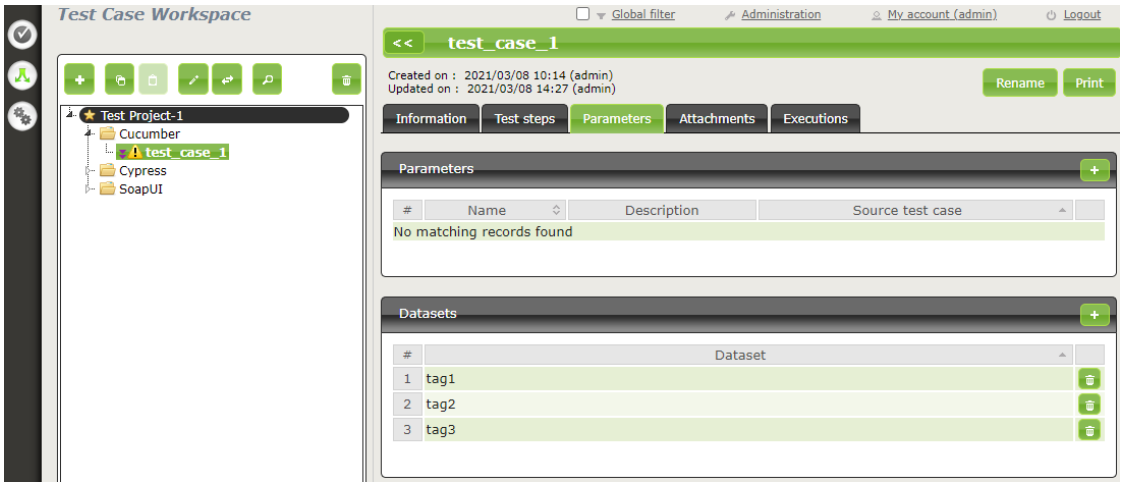
Below is an example of a *Cucumber* test file and the corresponding **Squash TM** test case automation :

cucumberGestionStock / src / test / resources / squash / 1_test_gherkin.feature in master

Edit file | Preview changes

```
Feature: Stock management

  Scenario Outline: Stock Addition
    Given I must add <element>
    And I assert its quantity
    When I add it to my stock
    Then I must at least have a minimum quantity in my stock

    @tag1
    Examples:
    | element |
    | "Ladders" |

    @tag2
    Examples:
      | element |
      | "Trunks" |

    @tag3
    Examples:
      | element |
      | "Planks" |

```





Automation with Cypress

1. Test reference

Note: In this version of **Squash AUTOM**, it is not possible to select a specific scenario in a `.spec.js` file containing several ones : every scenario in the file are therefore executed together. The result of each executed **Squash TM** test case is calculated by taking into account the individual results of each scenario included in the bound file :

- If at least one scenario has an *Error* status (in case of a technical issue), the status of the execution will be *Blocked*.
- If at least one scenario fails functionally and none of the other has an *Error* status, the status of the execution will be *Failed*.
- If all scenarios succeed, the status of the execution will be *Success*.

In order to bind a **Squash TM** test case to a *Cypress* automated test, the content of the *Automated test reference* field of the *Automation* panel of a test case must have the following format :

```
[1] / [2]
```

With :

- `[1]` : Name of the project on the source code repository.
- `[2]` : Path and name of the *Cypress* test file, from the root of the project (with the `.spec.js` extension).

2. Nature of the exploitable Squash TM parameters

The exploitable **Squash TM** parameters in a *Cypress* script will differ depending on whether you're using the **Community** or **Premium** version of **Squash DEVOPS**.

Here is a table showing the exploitable parameters :

Nature	Key	Community	Premium
Name of the dataset	DSNAME	✓	✓
Dataset parameter	DS_[name]	✓	✓
Test case reference	TC_REF	✓	✓
Test case CUF	TC_CUF_[code]	✓	✓
Iteration CUF	IT_CUF_[code]	✗	✓
Campaign CUF	CPG_CUF_[code]	✗	✓
Test suite CUF	TS_CUF_[code]	✗	✓

Legend :

- `CUF` : *Custom Field*
- `[code]` : *Value of a CUF's "Code" field*
- `[name]` : *Parameter name as filled in Squash TM*

3. Squash TM parameters usage

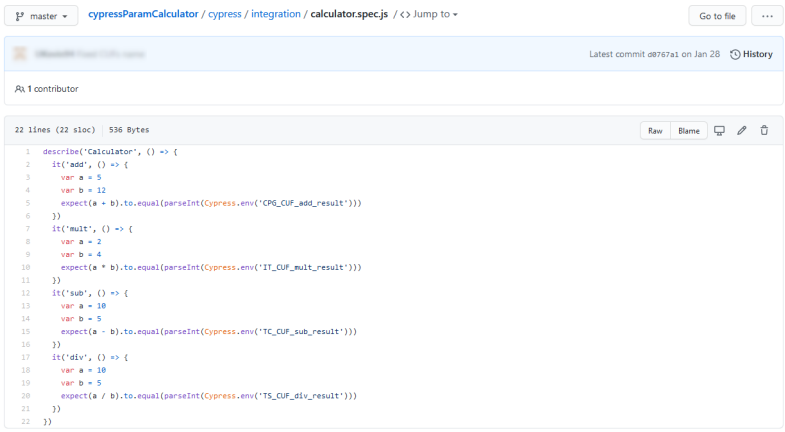
When executing a **Squash TM** automated test case with *Cypress*, it is possible to exploit the **Squash TM** parameters inside the test.

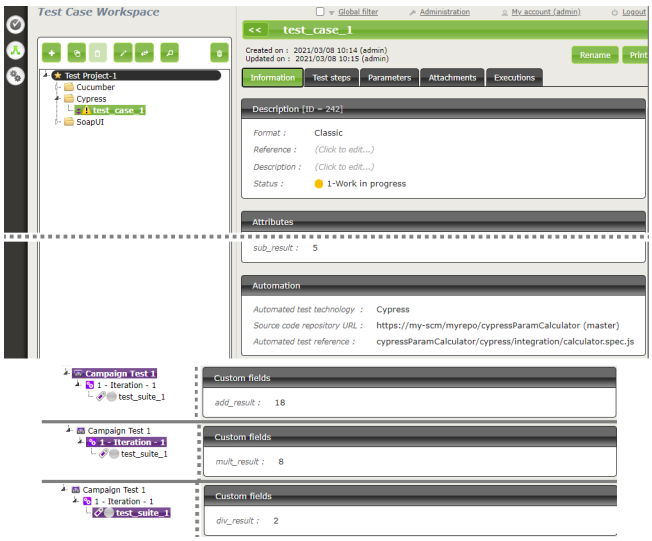
In order to achieve this, you'll have to follow these steps :

- Create custom fields in **Squash TM** and bind them to the project bearing the test plan to execute.
- Make sure that the *code* fields of the parameters correspond to the names of the existing environment variables present in the *Cypress* script.

Note: **Squash TM** adds a prefix to the *code* of the transmitted custom field. Make sure to take it into account. Please refer to the **Squash TM** documentation for more information.

Below is an example of a *Cypress* test file and the corresponding **Squash TM** test case automation :





Automation with JUnit

Test reference

In order to bind a **Squash TM** test case to a *JUnit* automated test, the content of the *Automated test reference* field of the *Automation* panel of a test case must have the following format :

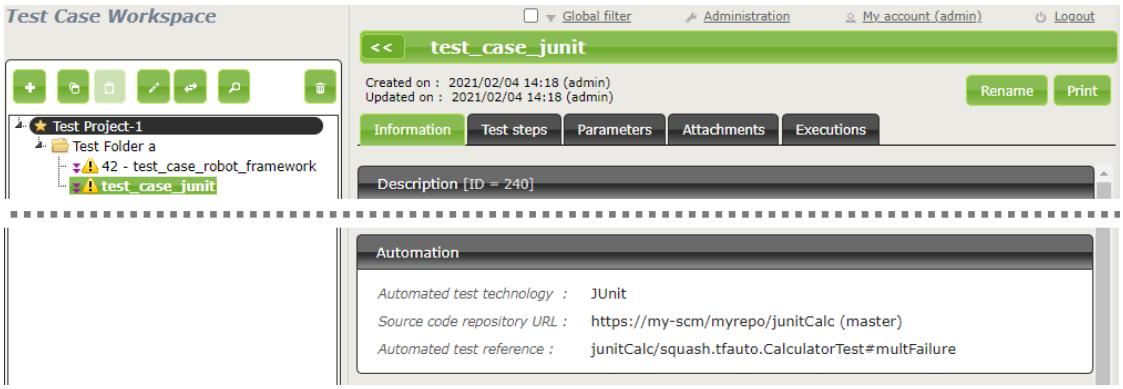
`[1] / [2] # [3]`

With :

- `[1]` : Name of the project on the source code repository.
- `[2]` : Qualified name of the test class.
- `[3]` : Name of the method to test in the test class.

Below is an example of a test class and the corresponding **Squash TM** test case automation :

```
1   package squash.tfauto;
2
3   import org.junit.jupiter.api.Assertions;
4   import org.junit.jupiter.api.DisplayName;
5   import org.junit.jupiter.api.Test;
6
7   @DisplayName("Calculator")
8   public class CalculatorTest {
9
24
25      @Test
26      public void multFailure(){
27          int first = 2;
28          int second = 4;
29          Assertions.assertTrue((first*second)==6, "Le résultat du calcul est incorrect. " + first + " *
30      }
```



Automation with Robot Framework

1. Test reference

In order to bind a **Squash TM** test case to a *Robot Framework* automated test, the content of the *Automated test reference* field of the *Automation* panel of a test case must have the following format :

`[1] / [2] # [3]`

With :

- `[1]` : Name of the project on the source code repository.
- `[2]` : Path and name of the *Robot Framework* test, from the root of the project (with the `.robot` extension).
- `[3]` : Name of the test case to execute in the `.robot` file.

2. Nature of the exploitable Squash TM parameters

The exploitable **Squash TM** parameters in a *Robot Framework* script will differ depending on whether you're using the **Community** or **Premium** version of **Squash DEVOPS**.

Here is a table showing the exploitable parameters :

Nature	Key	Community	Premium
Name of the dataset	DSNAME	✓	✓
Dataset parameter	DS_[name]	✓	✓
Test case reference	TC_REF	✓	✓
Test case CUF	TC_CUF_[code]	✓	✓
Iteration CUF	IT_CUF_[code]	✗	✓
Campaign CUF	CPG_CUF_[code]	✗	✓
Test suite CUF	TS_CUF_[code]	✗	✓

Legend :

- `CUF` : *Custom Field*
- `[code]` : *Value of a CUF's "Code" field*
- `[name]` : *Parameter name as filled in Squash TM*

3. Squash TM parameters usage

When executing a **Squash TM** automated test case with *Robot Framework*, it is possible to exploit the **Squash TM** parameters inside the test.

In order to achieve this, you'll have to follow these steps :

- Create custom fields in **Squash TM** and bind them to the project bearing the test plan to execute.
- Install the *squash-tf-services* python library on the environment where the *Robot Framework* execution takes place. It is accessible through the `pip` package management and can be installed by executing the following command line :

```
python -m pip install squash-tf-services
```

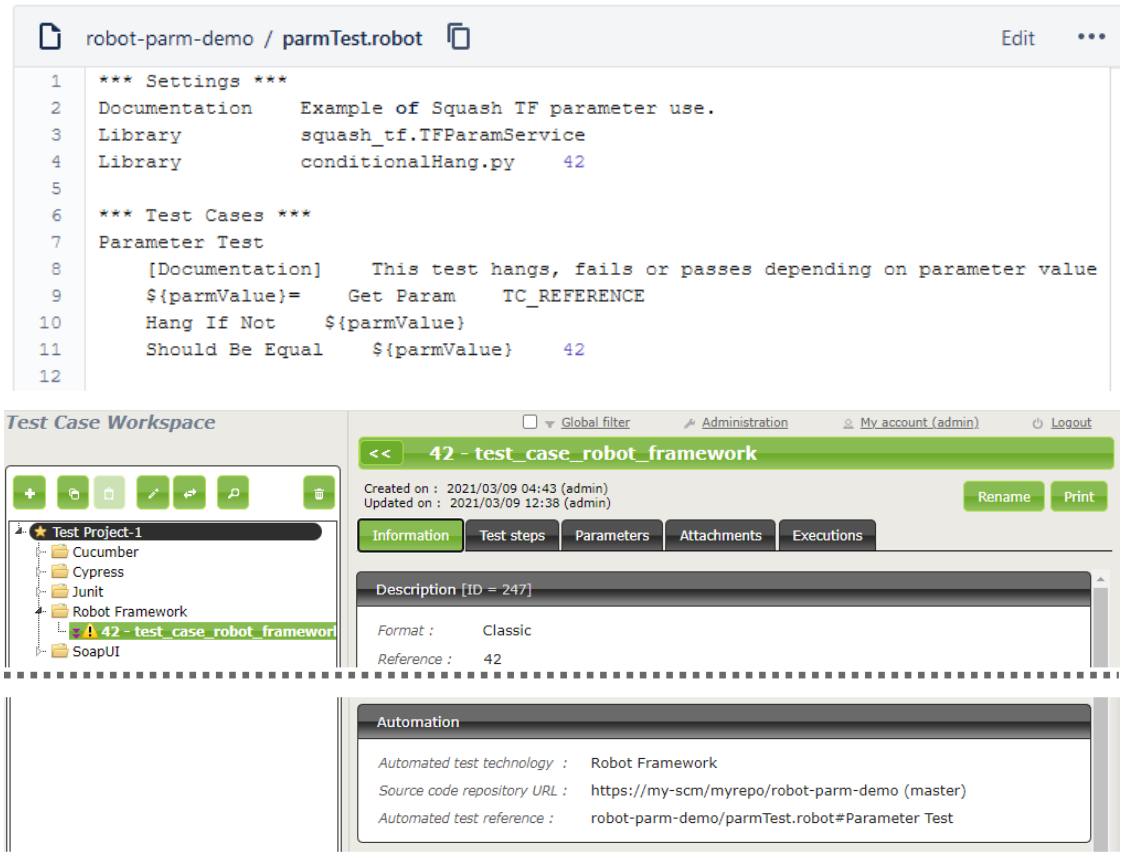
- Import the library inside the `.robot` file in the *Settings* section :

```
Library squash_tf.TFParamService
```

- You can then retrieve the value of a **Squash TM** parameter by calling the following keyword :

```
Get Param <parameter key>
```

Below is an example of a *Robot Framework* test file and the corresponding **Squash TM** test case automation :



Automation with SoapUI

Test reference

In order to bind a **Squash TM** test case to a *SoapUI* automated test, the content of the *Automated test reference* field of the *Automation* panel of a test case must have the following format :

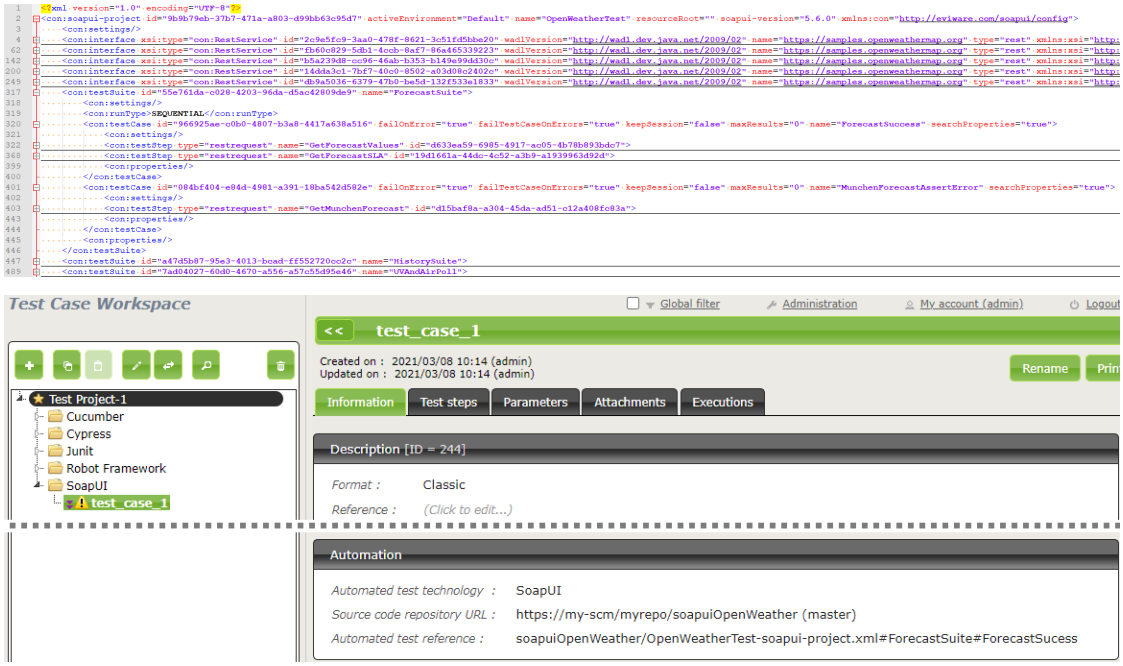
`[1] / [2] # [3] # [4]`

With :

- `[1]` : Name of the project on the source code repository.

- `[2]` : Path and name of the *SoapUI* test file, from the root of the project (with the `.xml` extension).
- `[3]` : Name of the TestSuite containing the test case.
- `[4]` : Name of the test case to execute.

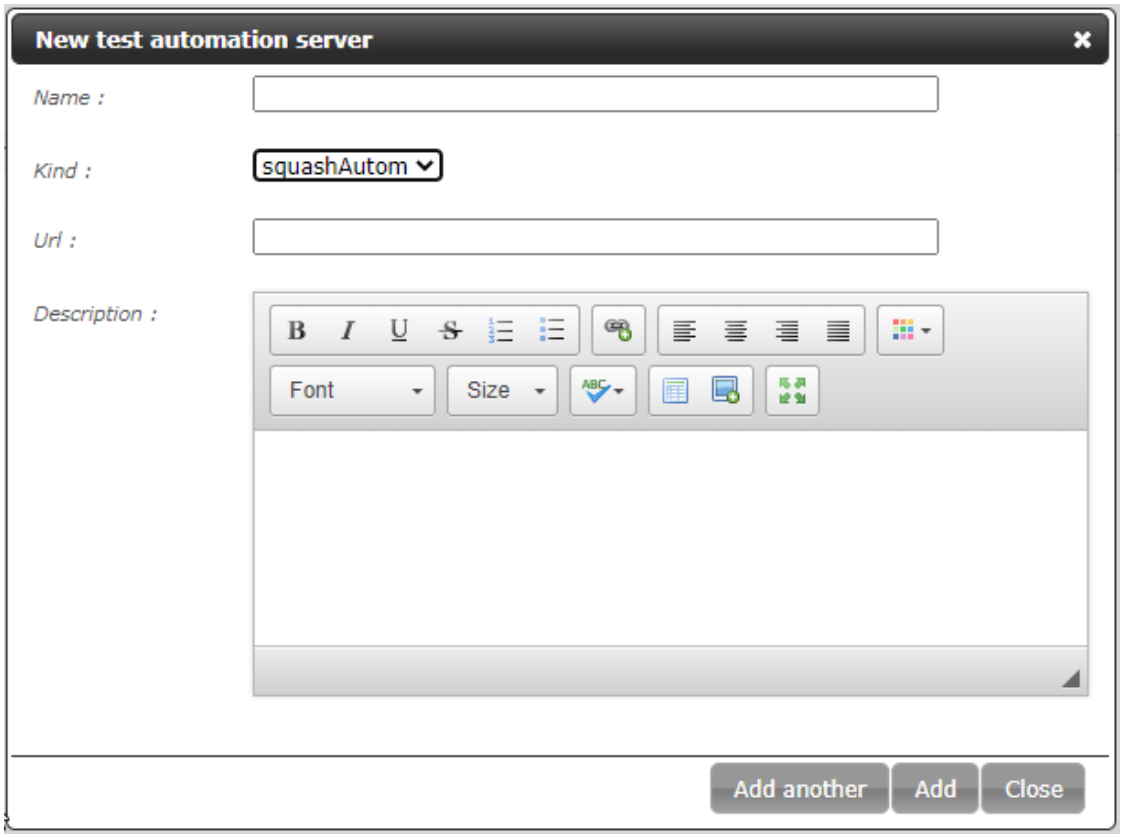
Below is an example of a *SoapUI* test file and the corresponding **Squash TM** test case automation :



1.3.2 Test plan execution from Squash TM

Squash Orchestrator server declaration

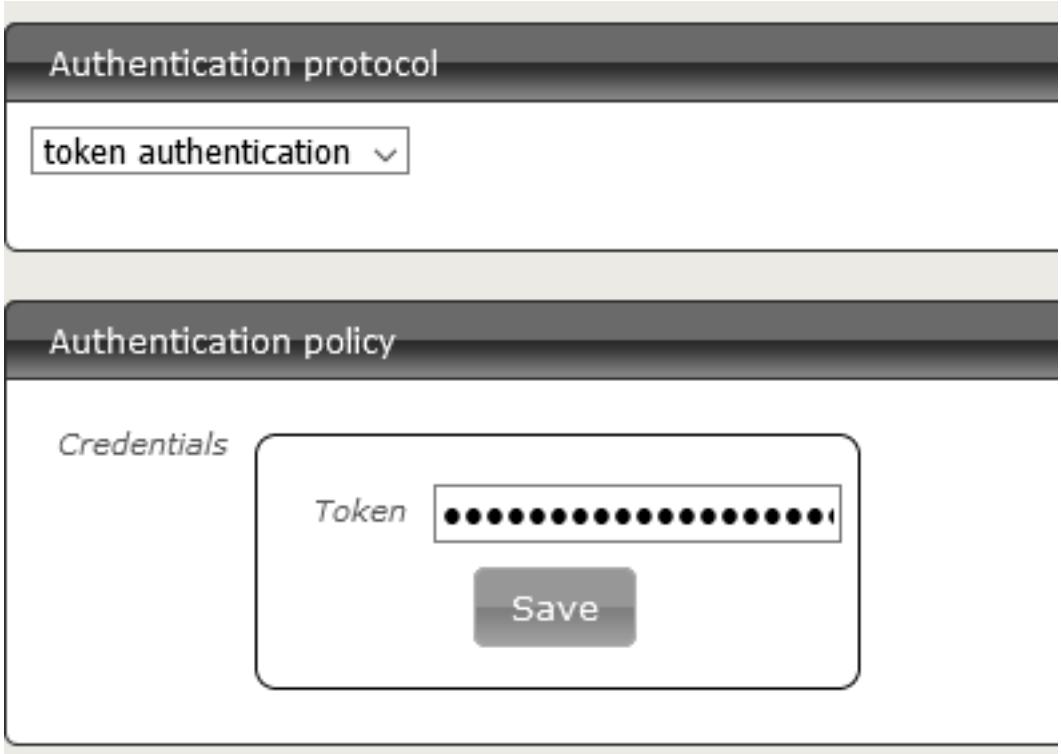
In order to manually launch an execution plan from **Squash TM**, the **Squash Orchestrator** server that will execute the automated tests in the suitable environments has to be declared. It is done in the *Automation servers* space of the *Administration* :



- `Name` : The name of server, as it will appear in the *Test Case* workspace.
- `Type` : Select *squashAutom* in the dropdown list.
- `Url` : The address of the **Squash Orchestrator** *Receptionist*.

> **Warning:** The **Squash Orchestrator** *event bus* service **must** be accessible by the same url as the *Receptionnist*, on port 38368.

Once the server is created, you can set an authentication token.

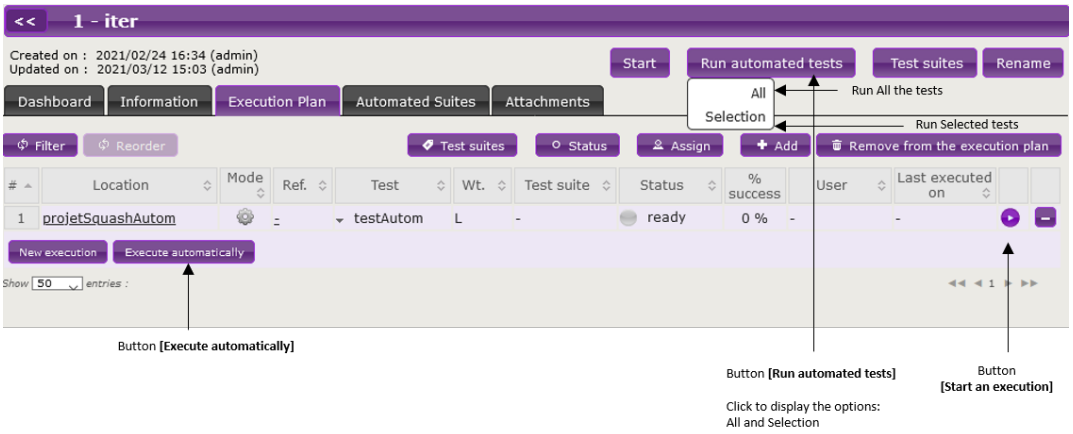


Note: A token is mandatory for the execution of automated tests from **Squash TM**. If the automation server does not require authentication token, you still have to set some value in **Squash TM**.

Automated suite execution

Steps to run an automated test plan in **Squash TM** are the usual ones:

- Get to the execution plan of the selected Iteration or Test Suite.
- Run the test using one of the button on the screen below :



- An Overview of automated test executions popup shows up.

Note: The execution overview popup contains a new section displaying the ongoing executions performed by the

Squash Orchestrator. However, the state of the executions are not updated once launched in the current version.

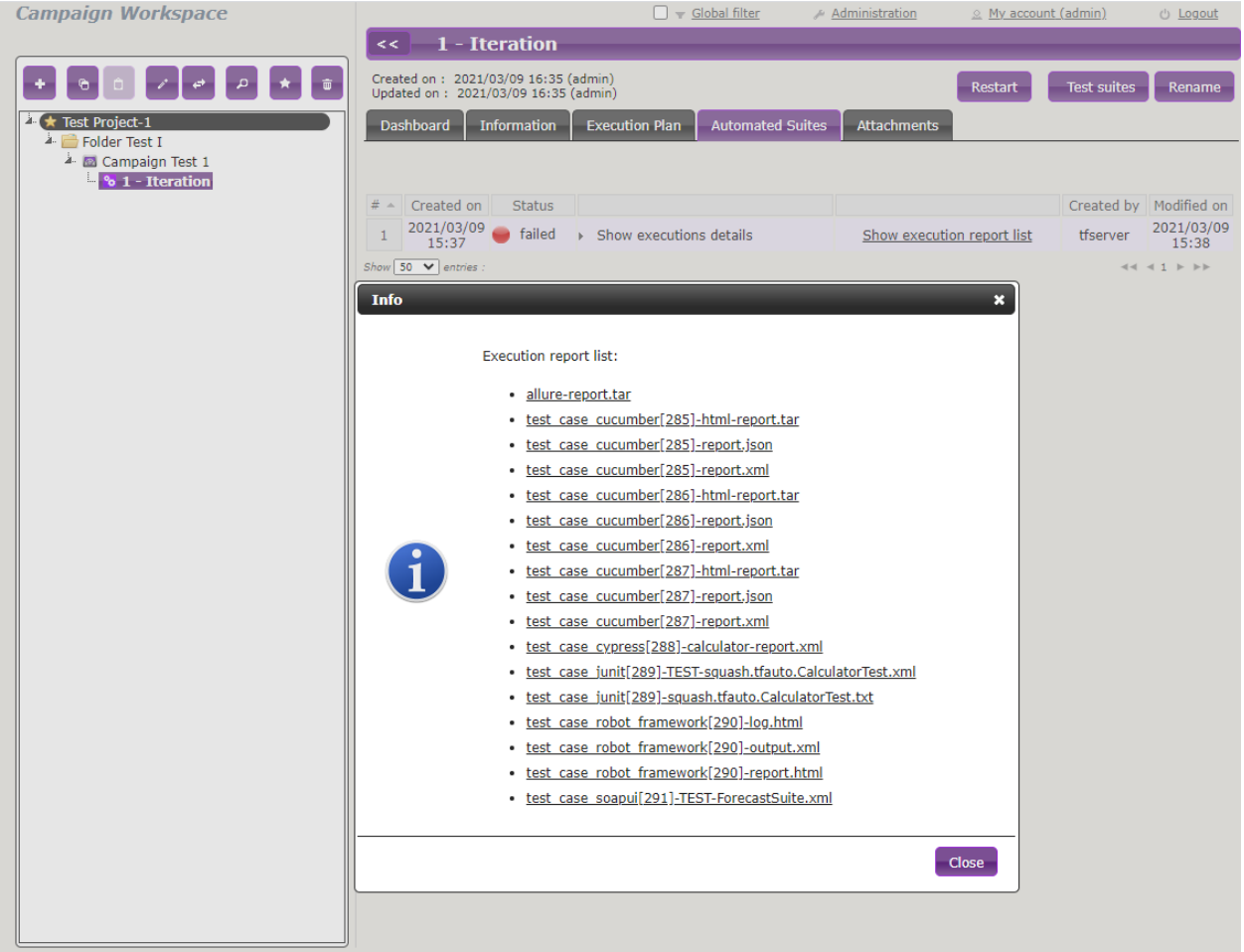
1.3.3 Results publication after a Squash TM test plan execution

Independently from the means used to trigger a test plan execution (from **Squash TM** or a *Jenkins* pipeline), the kind of results published in **Squash TM** at the end of the execution of a test plan will differ depending on your using a **Squash AUTOM Community** or **Squash AUTOM Premium** licence.

Squash AUTOM Community

After the execution of a **Squash TM** test plan (iteration or test suite), the following information is updated :

- ITPIs status update.
- Automated suite status update.
- The *Allure* type report containing all the results from the executed tests.
- The various ITPIs execution reports are accessible from the *Automated Suites* tab of the iteration or test suite :



Note: All the results from the automated suite are compiled in an *Allure* type report, available in the list of reports as a *.tar* archive.

However, in version *1.0.0.alpha2*, the *Robot Framework* test results can't be included in this report. If the automated suite contains only *Robot Framework* tests, the archive will be generated with an empty report.

For more information on the means to exploit and customize the *Allure* report, please refer to the Allure documentation.

This, however, doesn't happen :

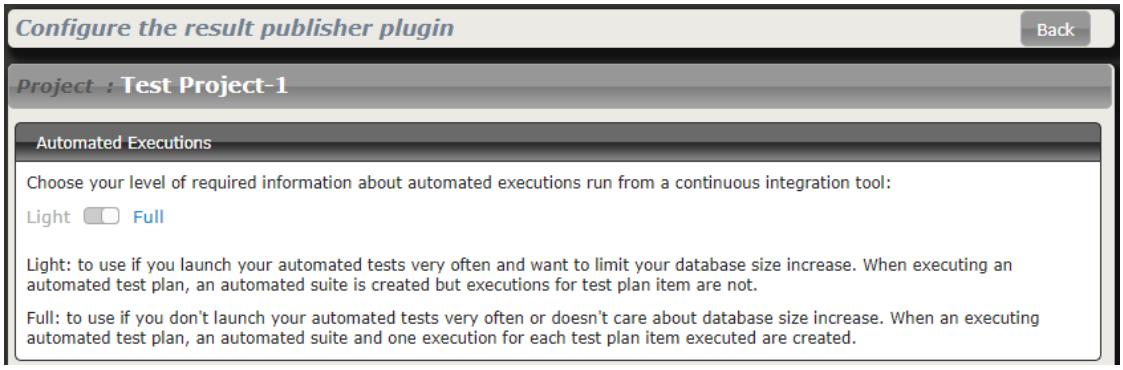
- Creation of a new execution for each executed ITPI.

Squash AUTOM Premium

If you are using the **Squash AUTOM Premium** components, you have access to two types of results publication :

- Light (default value).
- Full.

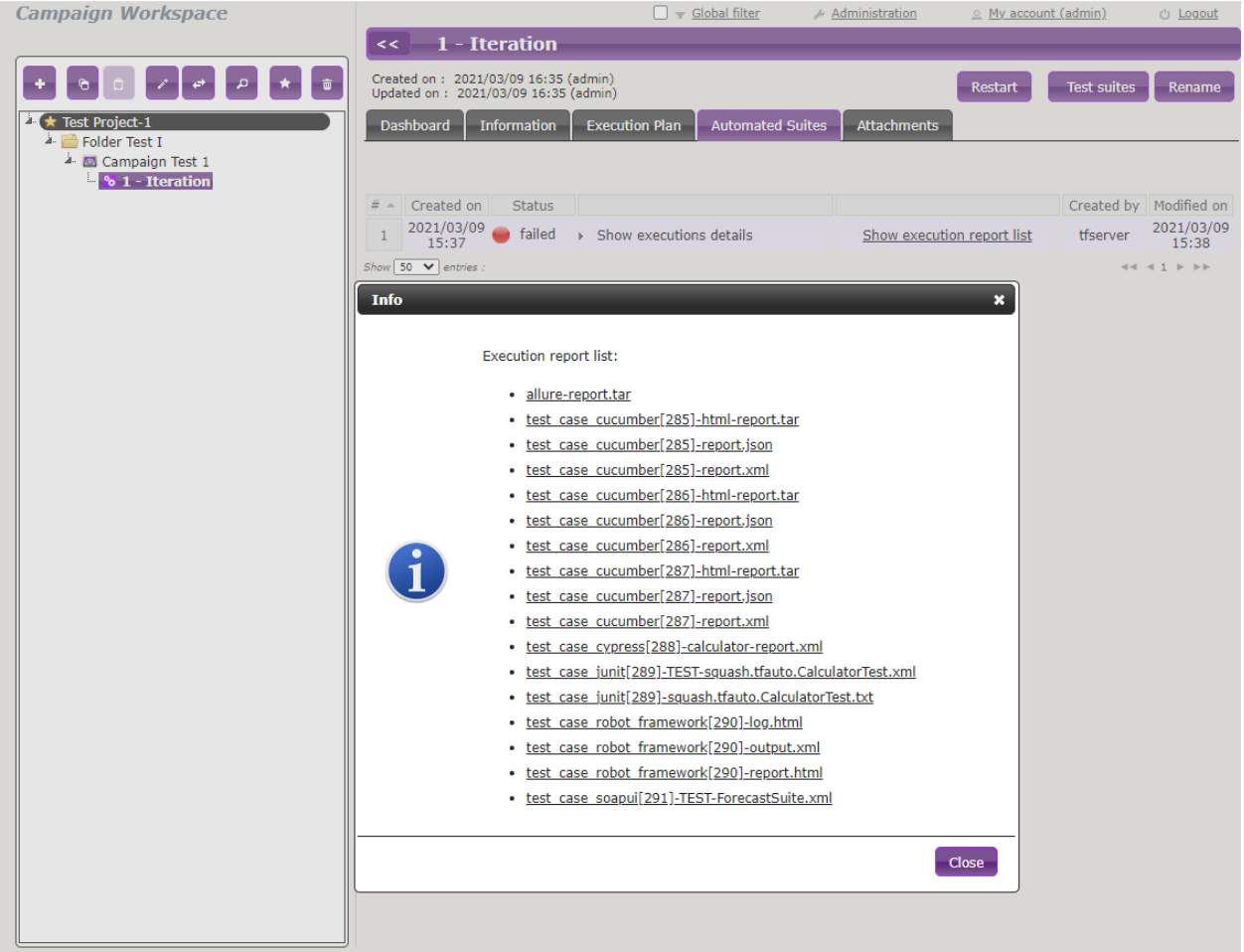
The choice of publication type is operated on a project basis by accessing the configuration of the **Squash TM Result Publisher** plugin from the *Plugins* tab of your project page, inside the *Administration* Tab :



Light results publication

By choosing the "Light" results publication, the following information is updated after the execution of a **Squash TM** test plan (iteration or test suite) :

- ITPIs status update.
- Automated suite status update.
- The *Allure* type report containing all the results from the executed tests.
- The various ITPIs execution reports are accessible from the *Automated Suites* tab of the iteration or test suite :



Note: All the results from the automated suite are compiled in an *Allure* type report, available in the list of reports as a *.tar* archive.

However, in version *1.0.0.alpha2*, the *Robot Framework* test results can't be included in this report. If the automated suite contains only *Robot Framework* tests, the archive will be generated with an empty report.

For more information on the means to exploit and customize the *Allure* report, please refer to the Allure documentation.

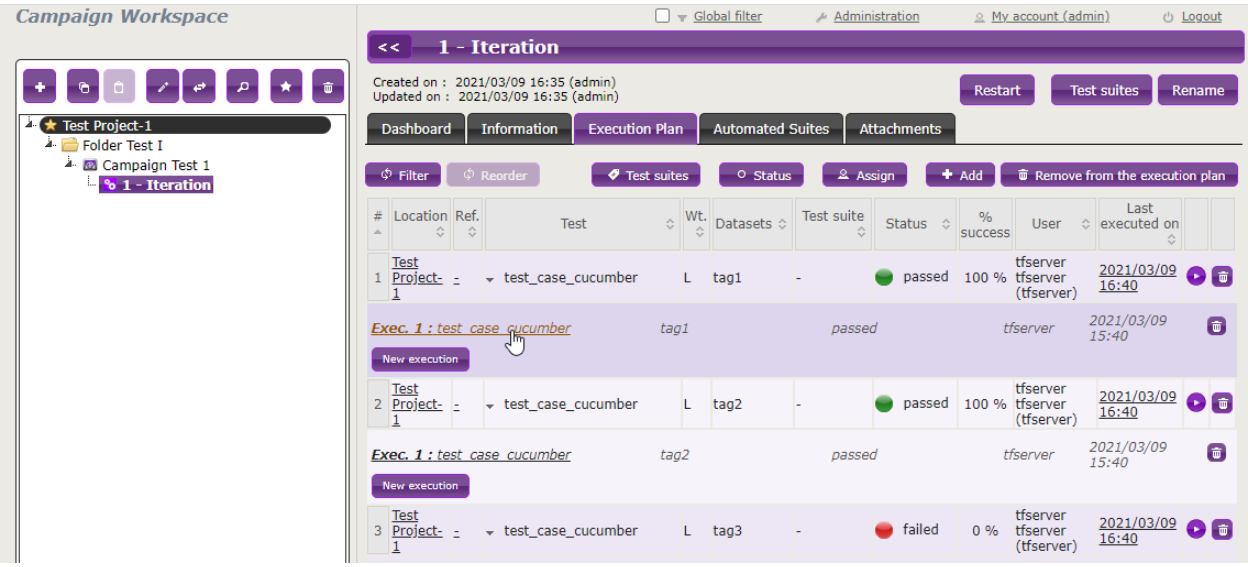
This, however, doesn't happen :

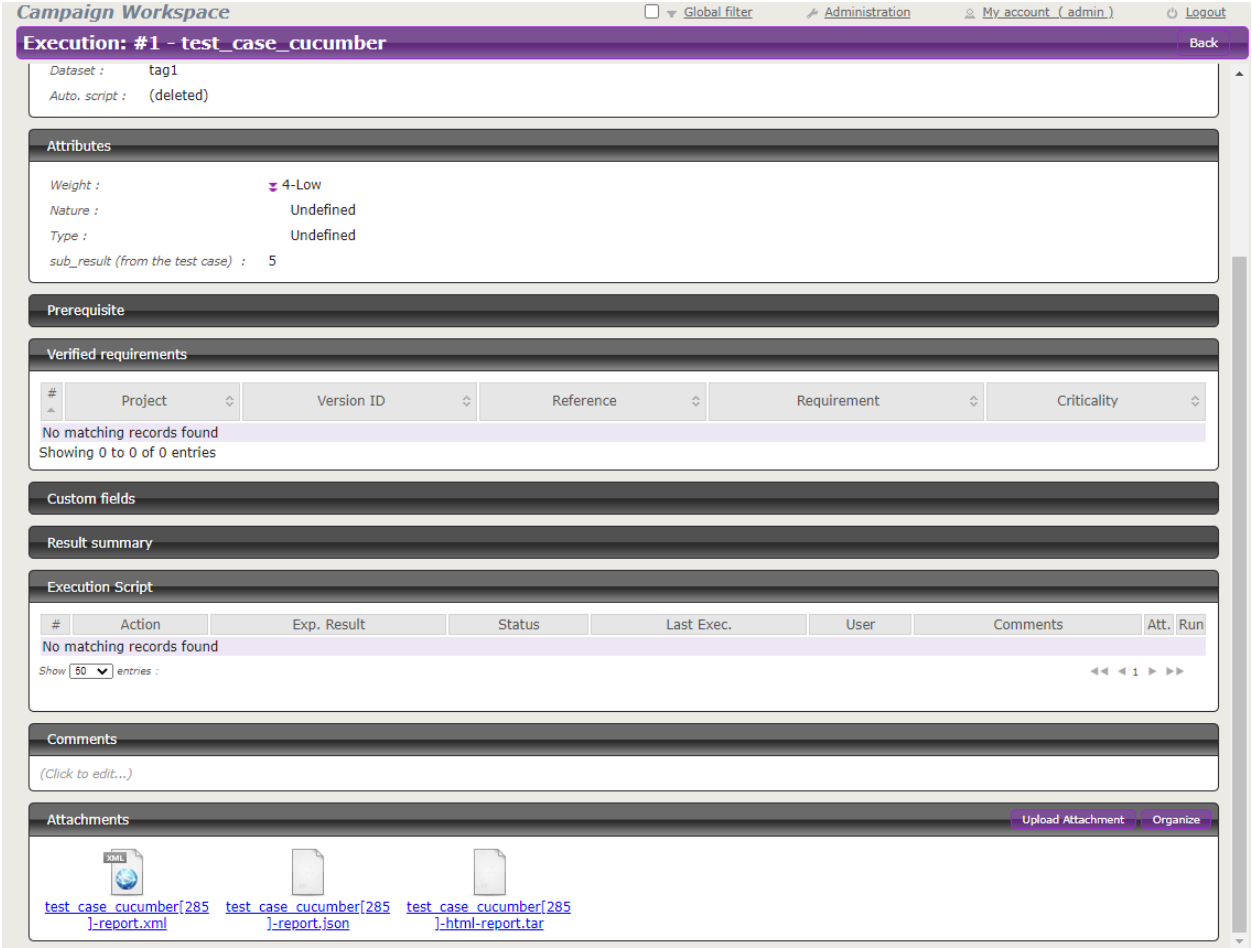
- Creation of a new execution for each executed ITPI.

Full results publication

By choosing the "Full" results publication, the following information is updated after the execution of a **Squash TM** test plan (iteration or test suite) :

- ITPIs status update.
- Creation of a new execution for each executed ITPI.
- Automated suite status update.
- The *Allure* type report containing all the results from the executed tests.
- The execution reports of the various executions can be accessed from the *Automated Suites* tab of the iteration or test suite, or from the execution page (the reports are present in the attached files) :





Note: All the results from the automated suite are compiled in an *Allure* type report, available in the list of reports as a *.tar* archive.

However, in version *1.0.0.alpha2*, the *Robot Framework* test results can't be included in this report. If the automated suite contains only *Robot Framework* tests, the archive will be generated with an empty report.

For more information on the means to exploit and customize the *Allure* report, please refer to the Allure documentation.

This guide will show you the various possibilities offered by the version *1.0.0.alpha2* of **Squash AUTOM**.

Warning: This version is intended to be used as a POC and therefore not in a production context (notably with a **Squash TM** whose database is new or a copy of an existing one).

This *1.0.0.alpha2* version provides two components :

- **Squash Orchestrator** : it is a tool composed of a set of micro-services to be used by sending an execution plan written in a specific format, the EPAC (Execution plan «as code»), in order to orchestrate automated tests.

- **Result Publisher Plugin for Squash TM** : this plugin for **Squash TM** allows the return of information towards **Squash TM** at the end of the execution of a **Squash TM** execution plan by the **Squash Orchestrator**.
- **Squash AUTOM Plugin for Squash TM** : this plugin for **Squash TM** allows to execute automated test from **Squash TM** with **Squash Orchestrator**.

CHAPTER

TWO

SQUASH DEVOPS

2.1 Installation Guide

2.1.1 Squash Orchestrator

This micro-service exists as a **Squash DEVOPS Community** and a **Squash DEVOPS Premium** version. It is included in the *Docker* image of the **Squash Orchestrator**. For further details on the deployment of the **Squash Orchestrator** and the activation of the **Squash TM Generator** micro-service in **Community** or **Premium** version, please refer to the **Squash Orchestrator** documentation (*Squash Orchestrator Documentation – 1.0.0.alpha2*, `.pdf` version) downloadable from https://www.squashtest.com/community-download.

2.1.2 Test Plan Retriever plugin for Squash TM

The plugin exists in a **Community** version (*squash.tm.rest.test.plan.retriever.community-1.0.0.alpha2.jar*) freely available, or a **Premium** version (*squash.tm.rest.test.plan.retriever.premium-1.0.0.alpha2.jar*) available on request.

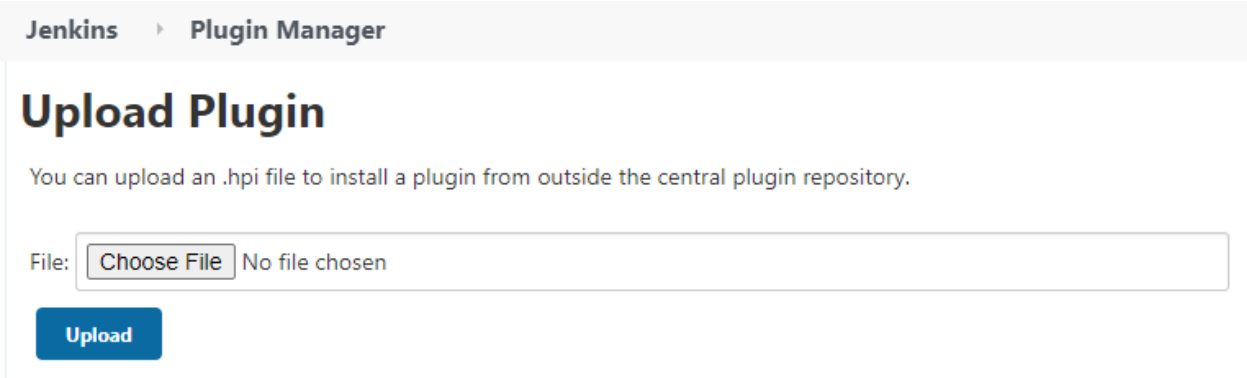
For details on the installation, please refer to installation protocol of a **Squash TM** plugin (https://sites.google.com/a/henix.fr/wiki-squash-tm/installation-and-exploitation-guide/2—installation-of-squash-tm/7—jira-plug-in).

Warning: This plugin is compatible with version *1.22.2.RELEASE* of **Squash TM**.

2.1.3 Squash DEVOPS plugin for Jenkins

The plugin is freely available from https://www.squashtest.com/community-download, as a `.hpi` file (*squash-devops-1.0.0.alpha2.hpi*).

To install it, submit the plugin in the *Upload Plugin* area accessible by the *Advanced* tab of the *Plugin Manager* in *Jenkins* configuration :



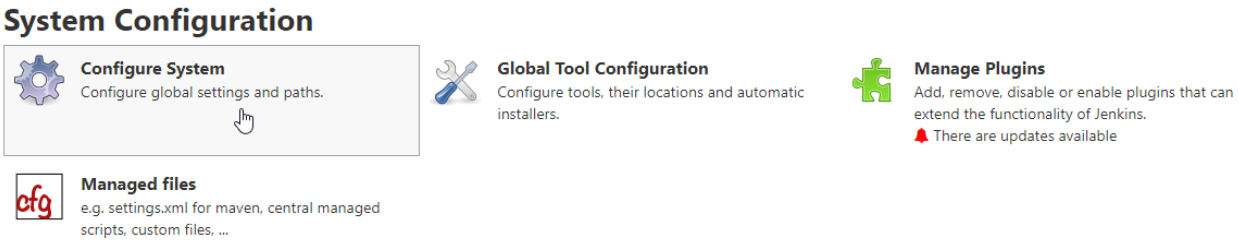
Warning: This plugin is compatible with version *2.164.1* or higher of *Jenkins*

2.2 Calling the Squash Orchestrator from a Jenkins pipeline

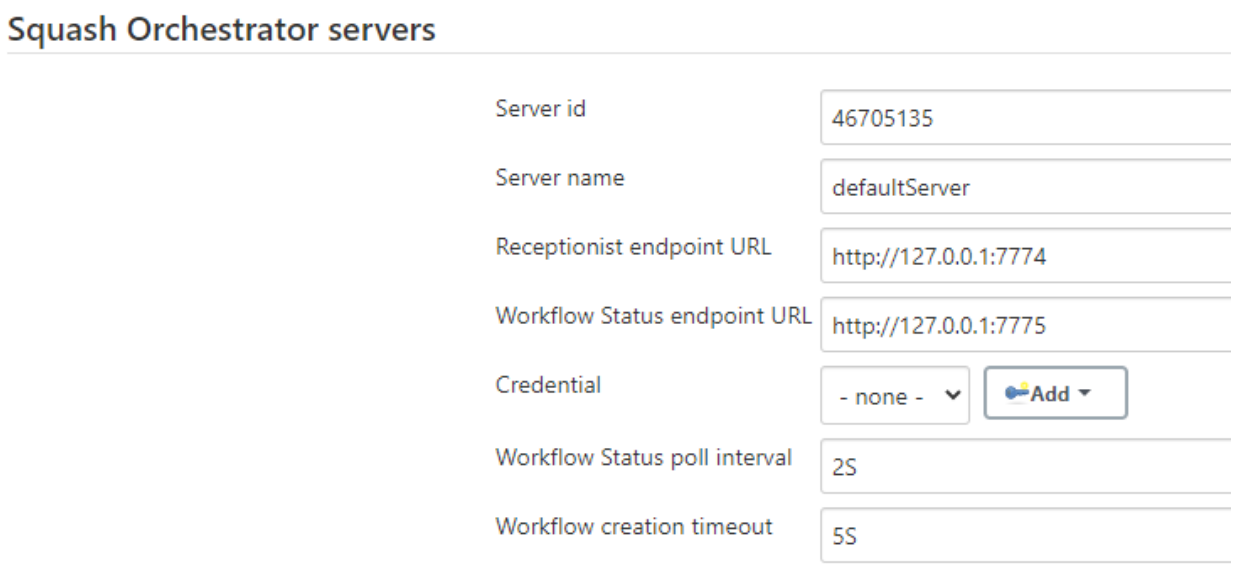
- *Configuring a Squash Orchestrator in Jenkins*
- *Call to the Squash Orchestrator from a Jenkins pipeline*

2.2.1 Configuring a Squash Orchestrator in Jenkins

To access the configuration of the **Squash Orchestrator**, you first need to go the *Configure System* page accessible in the *System Configuration* space of *Jenkins*, through the *Manage Jenkins* tab :



A panel named *Squash Orchestrator servers* will then be available :



- `Server id` : This ID is automatically generated and can't be modified. It is not used by the user.
- `Server name` : This name is defined by the user. It is the one that will be mentioned in the pipeline script of the workflow to be executed.
- `Receptionist endpoint URL` : The address of the *receptionist* micro-service of the orchestrator, with its port as defined at the launch of the orchestrator. Please refer to the **Squash Orchestrator** documentation for further details.
- `Workflow Status endpoint URL` : The address of the *observer* micro-service of the orchestrator, with its port as defined at the launch of the orchestrator. Please refer to the **Squash Orchestrator** documentation for further details.
- `Credential` : *Secret text* type *Jenkins* credential containing a *JWT Token* allowing authentication to the orchestrator. Please refer to the **Squash Orchestrator** documentation for further details on secure access to the orchestrator.
- `Workflow Status poll interval` : This parameter sets the interval between each update of the workflow status.
- `Workflow creation timeout` : Timeout on the reception of the EPAC by the *receptionist* on the orchestrator side.

2.2.2 Call to the Squash Orchestrator from a Jenkins pipeline

Once there is at least one **Squash Orchestrator** configured in *Jenkins*, it is possible to call the **Squash Orchestrator** from a *pipeline* type job in *Jenkins* thanks to a dedicated pipeline method.

Below is an example of a simple pipeline using the calling method to the orchestrator :

```
node {
   stage 'Stage 1 : sanity check'
   echo 'OK pipelines work in the test instance'
```

(continues on next page)

(continued from previous page)

```
   stage 'Stage 2 : steps check'
   configFileProvider([configFile(
fileId: '600492a8-8312-44dc-ac18-b5d6d30857b4',
targetLocation: 'testWorkflow.json'
)]) {
        def workflow_id = runSquashWorkflow(
        workflowPathName:'testWorkflow.json',
        workflowTimeout: '20S',
        serverName:'defaultServer'
        )
        echo "We just ran The Squash Orchestrator workflow $workflow_id"
   }
}
```

The *runSquashWorkflow* method allows the transmission of an EPAC to the orchestrator for an execution.

It uses 3 parameters :

- `workflowPathName` : The path to the file containing the EPAC. In the present case, the file is injected through the *Config File Provider* plugin, but it is also possible to get it through other means (retrieval from a SCM, on the fly generation in a file, . . .).
- `workflowTimeout` : Timeout on the actions execution. This timeout will activate for example if an environment is unreachable (or doesn't exist), or if an action is not found by an actionProvider. It is to be adapted depending on the expected duration of the execution of the various tests in the EPAC.
- `serverName` : Name of the **Squash Orchestrator** server to use. This name is defined in the *Squash Orchestrator servers* space of the *Jenkins* configuration.

2.3 Squash TM test execution plan retrieval with a PEAC

Squash DEVOPS gives you the possibility to retrieve an execution plan for automated tests defined in **Squash TM** with an EPAC. The EPAC can be triggered by a *Jenkins* pipeline (see the *corresponding page* of this guide).

2.3.1 Prerequisites

In order to retrieve an execution plan from **Squash TM** with an EPAC, you need to perform the following tasks in **Squash TM** :

- Create a user belonging to the *Test automation server* group.
- Create an execution plan (iteration or test suite) containing at least one ITPI linked to an automated test case, as described in the **Squash AUTOM** user guide (see *here*).

2.3.2 Integration of the Squash TM execution plan retrieval step into an EPAC

In order to retrieve an execution plan from **Squash TM** with an EPAC, you need to call the corresponding *generator* action.

Here is a simple example of an EPAC in *Json* format allowing the retrieval of a **Squash TM** execution plan :

```
{
    "apiVersion": "opentestfactory.org/v1alpha1",
    "kind": "Workflow",
    "metadata": {
        "name": "Simple Workflow"
    },
    "defaults": {
        "runs-on":"ssh"
    },
    "jobs": {
        "explicitJob": {
            "runs-on":"ssh",
            "generator":"tm.squashtest.org/tm.generator@v1",
            "with": {
                "testPlanUuid":"1e2ae123-6b67-44b2-b229-274ea17ad489",
                "testPlanType":"Iteration",
                "squashTMUrl":"https://mySquashTMInstance.org/squash",
                "squashTMAutomatedServerLogin":"tfserver",
                "squashTMAutomatedServerPassword":"tfserver"
            }
        }
    }
}
```

A **Squash TM** *generator* step must contain the following parameters :

- `testPlanType` : Defines the type of test plan to retrieve in **Squash TM**. Only the values *Iteration* and *TestSuite* are accepted.
- `testPlanUuid` : This is the UUID of the requested test plan. It can be found in the *Description* panel by clicking on the *Information* tab of the iteration or test suite in **Squash TM**.
- `squashTMUrl` : URL of the targeted **Squash TM**.
- `squashTMAutomatedServerLogin` : Name of the *Test automation server* group user to log into **Squash TM**.
- `squashTMAutomatedServerPassword` : Password of the *Test automation server* group user to log into **Squash TM**.

[*Optional fields*] :

- `tagLabel` : Specific to the **Premium** version - It refers to the name of the *tag* type custom field on which the test cases to retrieve are to be filtered. It is not possible to specify more than one.
- `tagValue` : Specific to the **Premium** version - It refers to the value of the *tag* type custom field on which the test cases to retrieve are to be filtered. It is possible to specify multiple ones separated by "|" (*Example:* value1|value2). There has to be at least one value specified for the test case to be taken into account.

Warning: If one of the two *tagLabel* or *tagValue* fields is present, the other **must** also be specified.

2.3.3 Squash TM parameters to exploit in an automated test

By executing an EPAC retrieving a **Squash TM** execution plan, **Squash TM** passes various pieces of information on ITPIs that can be exploited in a *Cucumber*, *Cypress* or *Robot Framework* test case.

For more information, please refer to the *Squash TM parameters exploitation* section of the **Squash AUTOM** documentation, as well as the dedicated section on the desired automation framework.

2.3.4 Results publication in Squash TM at the end of the execution

The nature of the results published in Squash TM at the end of the execution will depend on the usage of a **Squash AUTOM Community** or **Squash AUTOM Premium** licence.

Please refer to the **Squash AUTOM** *1.0.0.alpha2* user guide for more information (see *here*).

This guide will show you the various possibilities offered by the version *1.0.0.alpha2* of **Squash DEVOPS**.

Warning: This version is intended to be used as a POC and therefore not in a production context (notably with a **Squash TM** whose database is new or a copy of an existing one).

This *1.0.0.alpha2* version provides the following components :

- **Squash TM Generator Micro-service for the Squash Orchestrator** : it is a micro-service for the **Squash Orchestrator** allowing the retrieval of a **Squash TM** test execution within an EPAC (Execution Plan «as code»). Please refer to the *Squash AUTOM* user guide for more information on the **Squash Orchestrator** and the EPAC.
- **Test Plan Retriever for Squash TM** : this plugin for **Squash TM** allows the sending of details about a **Squash TM** execution plan to the **Squash Orchestrator**.
- **Squash DEVOPS plugin for Jenkins** : this plugin for *Jenkins* facilitates the sending of an EPAC to the **Squash Orchestrator** from a *Jenkins* pipeline.