Squash Autom & Squash Devops

Release 1.0.0-alpha1

squashtest

Nov 30, 2021

CONTENTS

> **Warning:** This site is obsolete, the Squash AUTOM and Squash DEVOPS documentation is now stored in https://autom-devops-en.doc.squashtest.com/.

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.

> **Warning:** This site is obsolete, the Squash AUTOM and Squash DEVOPS documentation is now stored in https://autom-devops-en.doc.squashtest.com/.

ONE

SQUASH AUTOM

1.1 Installation Guide

- *Squash Orchestrator*
- *Docker image of Squash AUTOM exclusive micro-services*
- *OpenTestFactory Agent*
- *Result Publisher Plugin for Squash TM*
- *Squash AUTOM Plugin for Squash TM*

1.1.1 Squash Orchestrator

Overview

Squash Orchestrator enables you to run and coordinate the various components of the test execution chain (execution environments, automata, reporting, etc.). It is based on the OpenTestFactory orchestrator and add a set of micro-services to extend its possibilities, such as running Squash TM execution plan or reporting to Squash TM.

Installation

Squash Orchestrator is a set of services running together. They may or may not run on the same machine, and they may or may not be started at the same time.

The only prerequisite is that the *EventBus*, the service they use to communicate together, is available when they launch.

To facilitate the installation of the orchestrator, an 'all-in-one' docker image is provided. It contains all core services and Squash specific services.

To get the latest image of **Squash Orchestrator**, use the following command:

```
docker pull squashtest/squash-orchestrator:latest
```

Usage

Configuring the 'all-in-one' image

The execution of the following command will start the **Squash Orchestrator** using an existing execution environment, with self-generated trusted keys (which is not recommended in a production setup):

```
docker run -d \
        --name orchestrator \
        -p 7774:7774 \
        -p 7775:7775 \
        -p 7776:7776 \
        -p 38368:38368 \
        -e SSH_CHANNEL_HOST=the_environment_ip_or_hostname \
        -e SSH_CHANNEL_USER=user \
        -e SSH_CHANNEL_PASSWORD=secret \
        -e SSH_CHANNEL_TAGS=ssh,linux,robotframework \
         squashtest/squash-orchestrator:latest
```

It exposes the following services on the corresponding ports:

- *receptionnist* (port 7774)

- *observer* (port 7775)

- *killswitch* (port 7776)

- *eventbus* (port 38368)

Run the image in Premium mode

If you have a **Premium** license for **Squash AUTOM** and have deployed the **Premium** version of the *Result Publisher* Squash TM plugin, you must run **Squash Orchestrator** in **Premium** mode in order to have access to **Premium** behavior for result publishing to **Squash TM**. In order to do that, you have to add the following parameter in the **Squash Orchestrator**'s run command: `-e SQUASH_LICENCE_TYPE=premium`

For more details about Squash Orchestrator configuration, please check the OpenTestFactory orchestrator documentation on which **Squash Orchestrator** is based on.

1.1.2 Docker image of Squash AUTOM exclusive micro-services

Overview

Squash AUTOM Premium license gives you access to a Docker image with **Squash Orchestrator** micro-services offering the following functionalities:

- Agilitest test execution management

- Ranorex test execution management

Installation

To install the Docker image of Squash AUTOM exclusives micro-services, you must get the compressed image from Squash support service. Then you have to execute the following command:

```
docker load -i squash-autom-premium-2.0.0.tar.gz
```

Usage

To run the Docker image of Squash AUTOM exclusives micro-services, the following command must be used:

```
docker run -e BUS_HOST=<IP or DNS name of event bus> \
         -e BUS_PORT=<port> -e BUS_TOKEN=<JWT token> \
         --volume /path/to/truststore:/etc/squashtf/ \
         docker.squashtest.org/squashtest/squash-autom-premium:2.0.0
```

with :

- *BUS_HOST* (mandatory) : IP address or DNS name of *EventBus* service of the **Squash Orchestrator** with which micro-services communicate.

- *BUS_PORT* (mandatory) : port of *EventBus* service of the **Squash Orchestrator** with which micro-services communicate.

- *BUS_TOKEN* (optionnal) : JWT token accepted by the *EventBus* service of the **Squash Orchestrator** with which micro-services communicate.

- *–volume* : create a volume to a truststore with trusted public keys that can be used to validate JWT tokens. The volume must be created if JWT token validation is required to exchange messages between micro-services.

1.1.3 OpenTestFactory Agent

Overview

OpenTestFactory agent is a process that runs on an execution environment. This process contacts the **Squash Orchestrator** at regular intervals, looking for orders to execute. Whenever there is a pending order, the agent will do as asked and send the result back to the orchestrator.

Installation

The OpenTestFactory agent is a Python application that is installed in the execution environment. It requires Python 3.8 or higher. It works on Linux, MacOS, and Windows.

The agent is a simple script. It only has one external dependency, the well known `requests` library (it will be installed if not already present on the execution environment).

To install it, use the following command:

```
pip3 install --upgrade opentf-agent
```

You can test your installation by running the following command:

```
opentf-agent --help
```

Usage

Summary

```
$ opentf-agent --help
usage: opentf-agent [-h] --tags TAGS --host HOST [--port PORT] [--path_prefix PATH_
→PREFIX] [--token TOKEN] [--encoding ENCODING] [--script_path SCRIPT_PATH]
                    [--workspace_dir WORKSPACE_DIR] [--name NAME] [--polling_delay␣
→POLLING_DELAY] [--liveness_probe LIVENESS_PROBE] [--retry RETRY] [--debug]

OpenTestFactory Agent

optional arguments:
  -h, --help            show this help message and exit
  --tags TAGS           a comma-separated list of tags (e.g. windows,robotframework)
  --host HOST           target host with protocol (e.g. https://example.local)
  --port PORT           target port (default to 24368)
  --path_prefix PATH_PREFIX
                        target context path (default to no context path)
  --token TOKEN         token
  --encoding ENCODING   encoding on the console side (defaults to utf-8)
  --script_path SCRIPT_PATH
                        where to put temporary files (defaults to current directory)
  --workspace_dir WORKSPACE_DIR
                        where to put workspaces (defaults to current directory)
  --name NAME           agent name (defaults to "test agent")
  --polling_delay POLLING_DELAY
                        polling delay in seconds (default to 5)
  --liveness_probe LIVENESS_PROBE
                        liveness probe in seconds (default to 300 seconds)
  --retry RETRY         how many time to try joining host (default to 5,
                        0 = try forever)
  --debug               whether to log debug informations.
```

Example

Assuming there is a **Squash Orchestrator** running on `orchestrator.example.com`, with a known token stored in the `TOKEN` environment variable, the following command will register the Windows-based execution environment and will possibly receive commands from **Squash Orchestrator** targeting windows and/or robotframework tagged environments:

```
chcp 65001
opentf-agent --tags windows,robotframework --host https://orchestrator.example.com/ --
→token %TOKEN%
```

The agent will poll the orchestrator every 5 seconds, and will execute the received commands. The `chcp` command sets the console to Unicode. It is Windows-specific. It is not mandatory but may be needed depending on the test framework available in the execution environment.

1.1.4 Result Publisher Plugin for Squash TM

Overview

The plugin is used simultaneously with **Squash Orchestrator** to allow reporting to **Squash TM** at the end of a **Squash TM** automated test plan execution by **Squash Orchestrator**.

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

Installation

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* or higher of **Squash TM**.

1.1.5 Squash AUTOM Plugin for Squash TM

Overview

The plugin is used simultaneously with **Squash Orchestrator** to allow automated test execution launch from **Squash TM**.

The plugin exists in a **Community** version (*plugin.testautomation.squashautom.community-1.0.0.RELEASE.jar*) freely available, or a **Premium** version (*plugin.testautomation.squashautom.premium-1.0.0.RELEASE.jar*) available on request.

Installation

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* or higher of **Squash TM**.

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

OpenTestFactory allows the redaction of an execution plan in a specific format, 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 OpenTestFactory orchestrator documentation

1.3 Piloting automated tests executions from Squash TM

- *Squash TM test case automation*
- *Test plan execution from Squash TM*
- *Published results after a Squash TM test plan execution*

Note: To pilote automated tests executions from Squash TM, following components are required:

- Squash TM
- Squash Orchestrator
- Result Publisher Plugin for Squash TM
- Squash AUTOM plugin for Squash TM
- In case of *Agilitest* test execution : an *OpenTestFactory* agent for *Agilitest* execution environment
- In case of *Ranorex* test execution : an *OpenTestFactory* agent for *Ranorex* execution environment, and an environment variable called *SQUASH_MSBUILD_PATH* containing the path to the parent folder of *MSBuild.exe*

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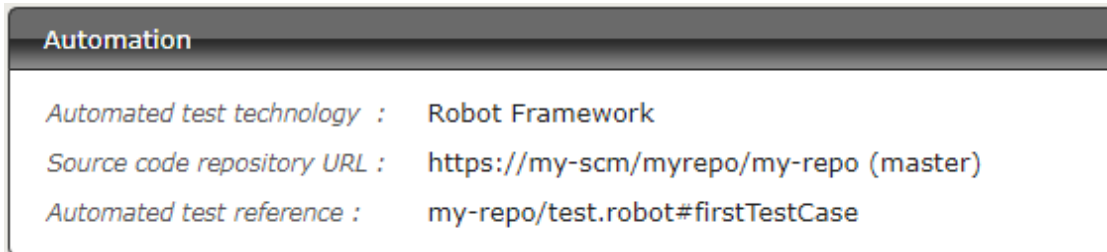
Note: This page describes the common operations to all supported test frameworks in this version. You can access the automation specificities for each technology directly with the following links :

- *Agilitest*
- *Cucumber*
- *Cypress*

- *JUnit*
- *Ranorex*
- *Robot Framework*
- *SoapUI*

Without using the Squash automation workflow

To execute a test case using the **Squash Orchestrator**, its *Automation* panel in the *Information* tab of the test case page must be correctly filled :
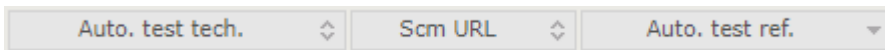
- `Automated test technology` : A dropdown list allowing you to choose the technology used for the execution of a test case. In this version, only *Robot Framework*, *Junit*, *Cucumber*, *Cypress*, *SoapUi*, *Agilitest* and *Ranorex* are functioning.
- `Source code repository URL` : The address of the 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 specific format of the used test technology (see *here*).

Note: *Agilitest* and *Ranorex* are only supported by the *premium* version of **Squash AUTOM**.

Using the Squash automation workflow

Regular test case

To execute a test case using 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*, *Junit*, *Cucumber*, *Cypress*, *SoapUi*, *Agilitest* and *Ranorex* 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 specific format of the used test technology (see *here*).

Note: *Agilitest* and *Ranorex* are only supported by the *premium* version of **Squash AUTOM**.

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*, *Robot Framework*, *Agilitest* or *Ranorex* test case. Details of this functionality can be found on the corresponding used technology section

Automation frameworks specifics

Automation with Agilitest

1. Test reference

In order to bind a **Squash TM** test case to an *Agilitest* 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 *ActionTestScript* file, from the root of the project (with the `.ats` extension).

> **Warning:** The ATS script **must** be located in `src/main/ats/*` , as in any regular ATS project architecture.

2. Nature of the exploitable Squash TM parameters

The exploitable **Squash TM** parameters in an *ActionTestScript* 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*
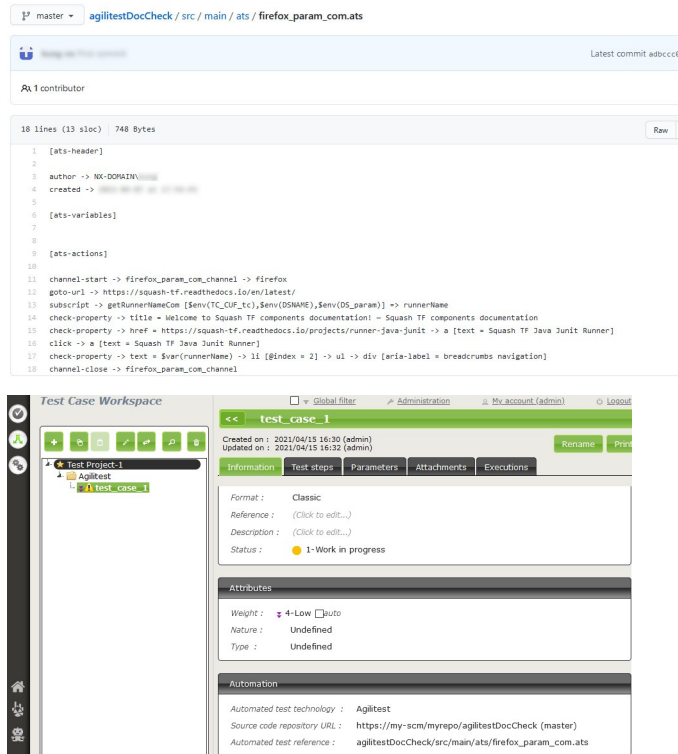- [name] : *Parameter name as filled in Squash TM*

3. Squash TM parameters usage

When executing a **Squash TM** automated test case with *Agilitest*, 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 an *Agilitest* test file and the corresponding **Squash TM** test case automation :

```
 master ▼   agilitestDocCheck / src / main / ats / firefox_param_com.ats

                                                                    Latest commit adbccc0

 1 contributor

18 lines (13 sloc)   748 Bytes                                            Raw

 1   [ats-header]
 2
 3   author -> NX-DOMAIN\...
 4   created -> ...
 5
 6   [ats-variables]
 7
 8
 9   [ats-actions]
10
11   channel-start -> firefox_param_com_channel -> firefox
12   goto-url -> https://squash-tf.readthedocs.io/en/latest/
13   subscript -> getRunnerNameCom [$env(TC_CUF_tc),$env(DSNAME),$env(DS_param)] => runnerName
14   check-property -> title = Welcome to Squash TF components documentation! — Squash TF components documentation
15   check-property -> href = https://squash-tf.readthedocs.io/projects/runner-java-junit -> a [text = Squash TF Java Junit Runner]
16   click -> a [text = Squash TF Java Junit Runner]
17   check-property -> text = $var(runnerName) -> li [@index = 2] -> ul -> div [aria-label = breadcrumbs navigation]
18   channel-close -> firefox_param_com_channel
```

Test Case Workspace   Global filter   Administration   My account (admin)   Logout

<< test_case_1

Created on : 2021/04/15 16:30 (admin)   Rename   Print
Updated on : 2021/04/15 16:32 (admin)

Information   Test steps   Parameters   Attachments   Executions

★ Test Project-1
  Agilitest
    test_case_1

Format :   Classic
Reference :   (Click to edit...)
Description :   (Click to edit...)
Status :   1-Work in progress

Attributes

Weight :   4-Low auto
Nature :   Undefined
Type :   Undefined

Automation

Automated test technology :   Agilitest
Source code repository URL : https://my-scm/myrepo/agilitestDocCheck (master)
Automated test reference :   agilitestDocCheck/src/main/ats/firefox_param_com.ats

Automation with Cucumber

1. Test reference

Note: If a scenario name is not specified, 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] # [3] # [4]
```

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).

- `[3]` : feature name as specified in the *Cucumber* test file.
- `[4]` : scenario name as specified in the *Cucumber* test file.

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 |

```
1    Feature: Stock management
2
3      Scenario Outline: Stock Addition
4        Given I must add <element>
5        And I assert its quantity
6        When I add it to my stock
7        Then I must at least have a minimum quantity in my stock
8
9        @tag1
10       Examples:
11       | element |
12       | "Ladders" |
13
14       @tag2
15       Examples:
16         | element |
17         | "Trunks" |
18
19       @tag3
20       Examples:
21         | element |
22         | "Planks" |
23
```

Test Case Workspace

☐ ▽ Global filter   ⚲ Administration   ⚇ My account (admin)   ⏻ Logout

<<   test_case_1

Created on : 2021/03/08 10:14 (admin)
Updated on : 2021/03/08 14:27 (admin)   Rename   Print

| Information | Test steps | Parameters | Attachments | Executions |

- ★ Test Project-1
  - 📁 Cucumber
    - ⚠ test_case_1
  - 📁 Cypress
  - 📁 SoapUI

Description [ID = 243]

Format :   Classic

Automation

Automated test technology :   Cucumber
Source code repository URL :   https://my-scm/myrepo/cucumberGestionStock (master)
Automated test reference :   cucumberGestionStock/src/test/resources/squash/1_test_gherkin.feature
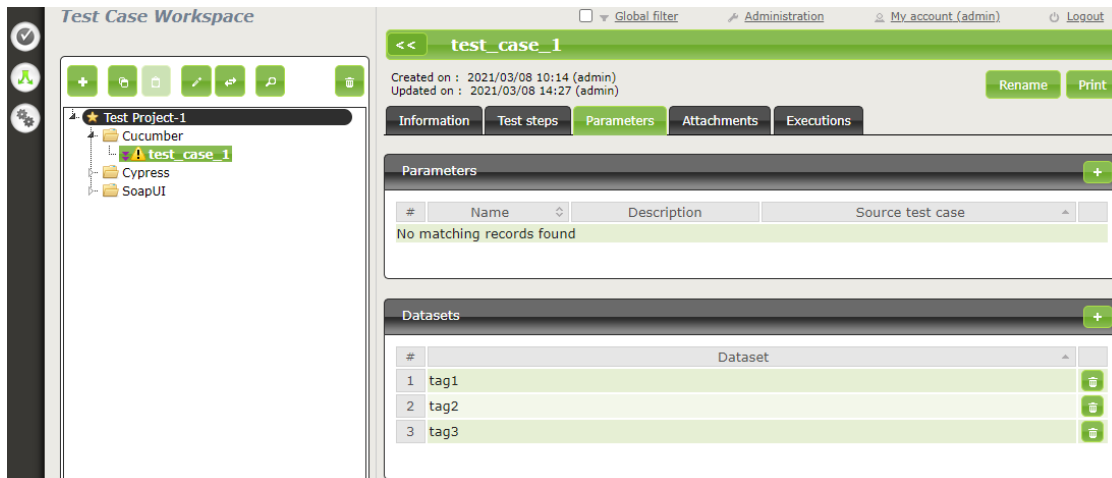
Automation with Cypress

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 : all scenarios 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 :
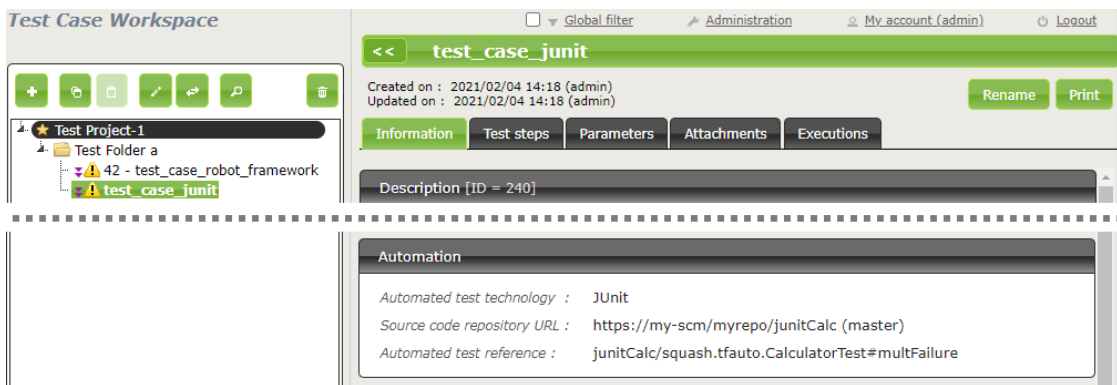
`[1] / [2]`

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 Ranorex

This feature is available only in the *Premium* version of **Squash AUTOM**.

In order to use this feature, an *OpenTestFactory* agent for *Ranorex* execution environment is needed. Furthermore, an environment variable called **SQUASH_MSBUILD_PATH** containing the path to the parent folder of **MSBuild.exe** must be created. You can execute the following command to get the path to your *MSBuild* executable :

```
reg.exe query "HKLM\SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0" /v MSBuildToolsPath
```

1. Test reference

In order to bind a **Squash TM** test case to a *Ranorex* 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]` : The path to the solution's `.sln` file from the root of the project source repository.
- `[2]` : *Ranorex* project name to execute.
- `[3]` : *Ranorex* test suite name to execute.
- `[4]` : *Ranorex* test case name to execute. This parameter is optional.

2. Nature of the exploitable Squash TM parameters

Here is a table showing the exploitable parameters :

Nature	Key
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 *Ranorex*, 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 parameters present in the *Ranorex* solution.

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 *Ranorex* solution and the corresponding **Squash TM** test case automation :
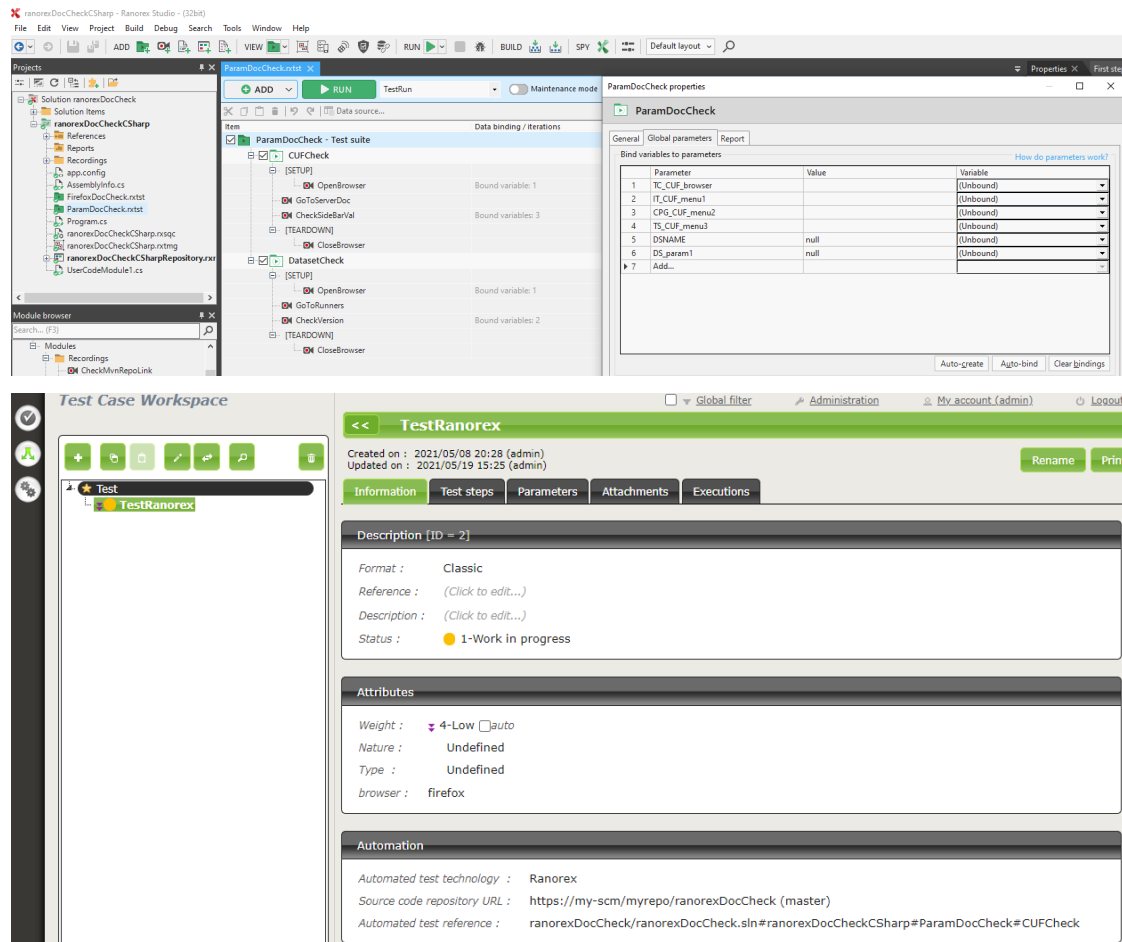
Automation with Robot Framework

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**.

### Squash TM project configuration

Once an automation server has been declared, it needs to be link to a Squash TM Project in order to be used for project's automated tests execution.

For that, follow these steps:

1. Click on [`Administration`] then click on [`Projects`]

2. Select an existing project, scroll down to `Test automation management`

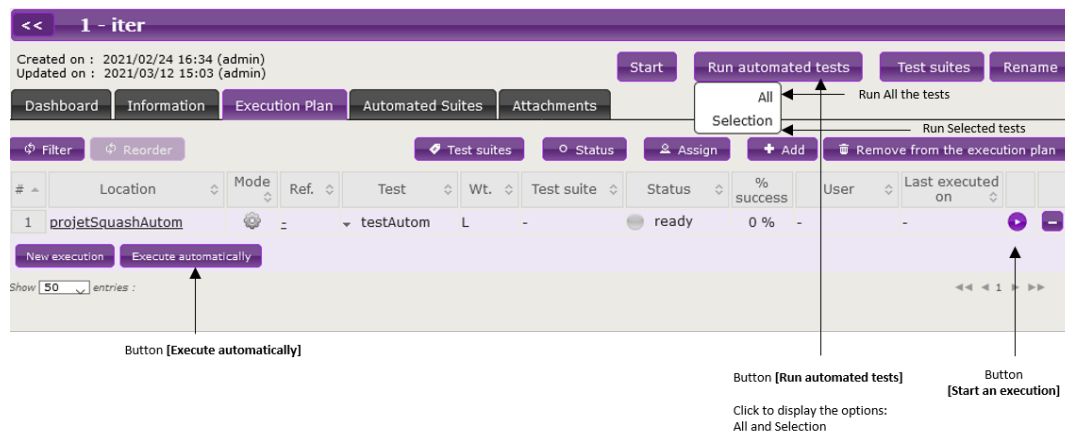3. Click on `No server`. You'll see the list of available servers.
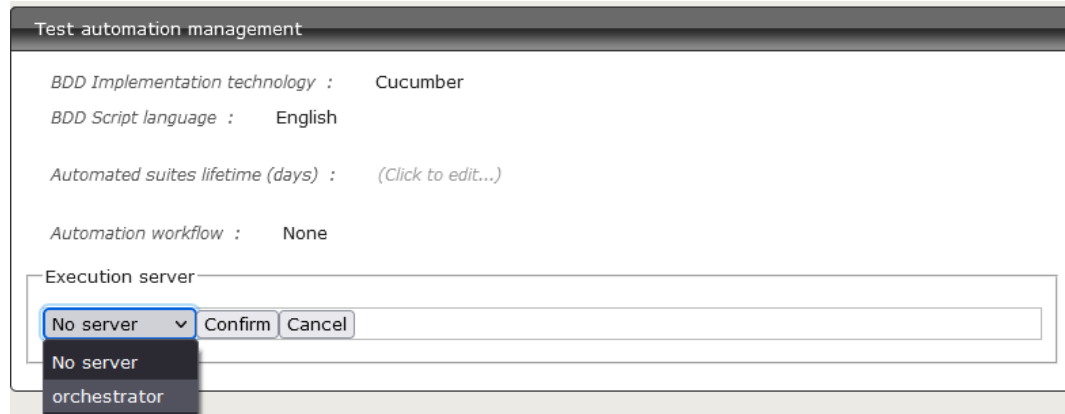
4. Choose the server and [Confirm]

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 Published results 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.

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 :
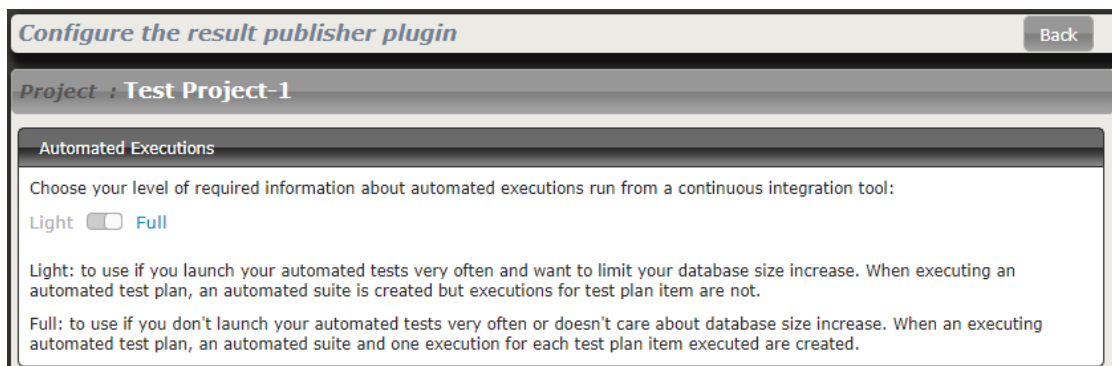
Light results publication

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.

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.

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 *1.1.0.RELEASE* 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 *1.1.0.RELEASE* version provides the following 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.
- **OpenTestFactory Agent** : agent to allow HTTP communication between a **Squash Orchestrator** and a test execution environment. It is mandatory for Agilitest or Ranorex test execution.

- **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**.

> **Warning:** This site is obsolete, the Squash AUTOM and Squash DEVOPS documentation is now stored in https://autom-devops-en.doc.squashtest.com/.

TWO

SQUASH DEVOPS

2.1 Installation Guide

- *Squash Generator Service*
- *Test Plan Retriever plugin for Squash TM*
- *Squash DEVOPS plugin for Jenkins*

2.1.1 Squash Generator Service

Overview

The Squash Generator micro-service allows **Squash TM** automated test plan retrieval when executing an EPaC with **Squash Orchestrator**.

Installation

It is included in the*Docker* image of the **Squash Orchestrator**. For further details on **Squash Orchestrator** installation, please refer to the *dedicated section*.

To run **Squash Orchestrator** Docker image with **Squash DEVOPS Premium** version of the Squash Generator Service, parameter `-e SQUASH_LICENCE_TYPE=premium` must be set in the `docker run` command of **Squash Orchestrator**.

2.1.2 Test Plan Retriever plugin for Squash TM

Overview

This plugin allows **Squash TM** automated test plan retrieval by a **Squash Orchestrator** server.

Installation

The plugin exists in a **Community** version (*squash.tm.rest.test.plan.retriever.community-1.0.0.RELEASE.jar*) freely available, or a **Premium** version (*squash.tm.rest.test.plan.retriever.premium-1.0.0.RELEASE.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* or higher of **Squash TM**.

2.1.3 Squash DEVOPS plugin for Jenkins
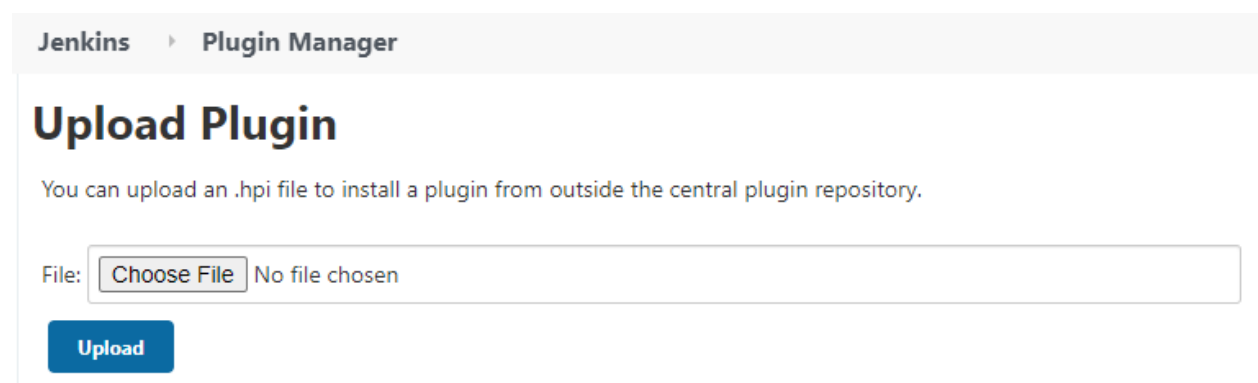
Overview

The plugin make calls to **Squash Orchestrator** inside a pipeline easier.

Installation

The plugin is freely available from https://www.squashtest.com/community-download, as a `.hpi` file (*squash-devops-1.1.0.hpi*).

To install it, you have to 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'
    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 be activated 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

- *Prerequisites*
- *Integration of the Squash TM execution plan retrieval step into an EPAC*
- *Squash TM parameters to exploit in an automated test*
- *Results publication in Squash TM at the end of the execution*

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).

Note: To a proper functioning of this functionality, Test Plan Retriever plugin and Result Publisher plugin are required on targeted Squash TM.

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",
                "technologyLabels":{
                    "ranorex": ["windows","ranorex"],
                    "robotframework": ["linux","robotframework"],
                    "junit": ["linux","junit"]
                }
            }
        }
    }
}
```

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 one **must** also be specified.

- `technologyLabels` : To use for example if you retrieve an execution plan containing tests which must be execute on different environments.

  It enables you to specify, for each automation framework, targeted environment execution's tags. They will be add to the ones specify in the job's runs-on attribute.

  For a technology, tags are defined by adding an entry to the field list with format `"technology prefix": ["tag1", "tag2"]`

Note: By default, jobs created by the Generator after retrieving an execution plan are tagged with the corresponding technology prefix (technology name in lower case with no space, for example "robotframework" for Robot Framework) in addition of the ones specified in Generator job's runs-on attribute.

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 *1.1.0.RELEASE* of **Squash DEVOPS**.

This *1.1.0.RELEASE* 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.

> **Warning:** This site is obsolete, the Squash AUTOM and Squash DEVOPS documentation is now stored in https://autom-devops-en.doc.squashtest.com/.

FAQ

3.1 FAQ : Generalities about Squash AUTOM and Squash DEVOPS

This FAQ answers question about why develop these new products, what does this mean for **Squash TF** and how will the transition from **Squash TF** to S|squashautom| and **Squash DEVOPS** take place?

- *Why develop two new products: Squash AUTOM et Squash DEVOPS?*
- *What is the model of Squash AUTOM and Squash DEVOPS?*
- *Can Squash AUTOM and Squash DEVOPS be used without Squash TM?*
- *Will new features for Squash TF be developed in the future?*
- *Does support for Squash TF end with the release of Squash AUTOM and Squash DEVOPS?*
- *Does my legacy of automated tests, previously run with Squash TF, need to be modified to be used with Squash AUTOM?*
- *Can I run SKF tests with Squash AUTOM and Squash DEVOPS?*
- *What do I need to do in Squash TM to run my automated execution plans with Squash AUTOM instead of Squash TF?*
- *Can I mix in the same execution plan automated test cases executed by Squash TF and test cases executed by Squash AUTOM?*
- *Do I need to have a Jenkins server to run my automated tests from Squash TM via Squash AUTOM?*
- *Can I launch my Squash TM automated execution plans from a Jenkins pipeline with Squash AUTOM and SQUASH DEVOPS?*

3.1.1 Why develop two new products: Squash AUTOM et Squash DEVOPS?

The creation of Squash AUTOM and Squash DEVOPS is the result of a reflection on the evolution of automation practices (growth of CI/CD and DevOps practices, more and more democratized use of containerization, multiplication of integration tools) and on how the Squash software suite could be in line with them.

It emerged that Squash TF had architectural and other limitations for its adoption within the DevOps principles.

This is why we decided to develop a new tool for managing the execution of automated tests, and this tool must respect the following principles:

- Micro-service architecture, particularly for deployment and usability reasons in DevOps environments.

- Separation between the functionalities allowing automation (for testers and automation engineers) and those allowing the integration of automated tests (for the pipeline manager) within the DevOps plant. This gave rise to 2 products named Squash AUTOM and Squash DEVOPS.

- Removal of the adhesion with Squash TM in order to make these two products independent of it.

3.1.2 What is the model of Squash AUTOM and Squash DEVOPS?

The model chosen is an "open core" model.

This model, which is the same as Squash TM, allows to offer two versions:

- A free Community version composed of an open source core and freemium modules. This version is fully functional (unrestrained).

- A commercial version, with annual subscription, composed of the Community version and commercial plugins. It brings additional value-added features, but not essential, as well as support.

3.1.3 Can Squash AUTOM and Squash DEVOPS be used without Squash TM?

Yes.

For both products, our goal is to bring value to companies or projects that do not use Squash TM:

- The use of Squash AUTOM "alone" thus makes it possible to unify/homogenize the use of various automatons (Selenium, Cypress, SoapUI, Appium. . .) and various studios (Robot Framework, Cucumber, UFT, Agilitest. . .) while generating a common reporting format (such as Allure).

- The use of Squash DEVOPS "alone" enables the orchestration of all automated tests, their integration into the DevOps pipeline (CI/CD) and the posting of results to the recipients (the pipeline itself, the test asset tool or the framework for reporting and aggregating test results).

3.1.4 Will new features for Squash TF be developed in the future?

No, there will be no new features developed for Squash TF.

We encourage you to make the transition from Squash TF to Squash AUTOM for the execution of your automated test assets in order to take advantage of all the new features offered by Squash.

Nevertheless, the elements of Squash TF will remain available for download. Likewise, the open source repositories will remain accessible.

3.1.5 Does support for Squash TF end with the release of Squash AUTOM and Squash DEVOPS?

No.

We will continue to provide support on Squash TF via the Squashtest forum and, for customers of the Squash AUTOM commercial offer, via our support service.

3.1.6 Does my legacy of automated tests, previously run with Squash TF, need to be modified to be used with Squash AUTOM?

No.

The scripts/automated tests that you run via Squash TF can be used by Squash AUTOM without any modification.

3.1.7 Can I run SKF tests with Squash AUTOM and Squash DEVOPS?

You will not be able to do it with the version 1.0.0.RELEASE.

SKF test support will be available in a later version before the end of Q2 2021.

3.1.8 What do I need to do in Squash TM to run my automated execution plans with Squash AUTOM instead of Squash TF?

It is necessary to create a link between a Squash TM test case and your automated test according to the Squash AUTOM documentation.

This action is almost instantaneous and can be done, at the same time, for several Squash TM Gherkin or BDD test cases using the Git plugin. For the other test cases, an action on each test case will be necessary according to the Squash AUTOM documentation.

The link action between a Squash TM test case and an automated test for an execution with Squash AUTOM is different from the one for an execution with Squash TF.

3.1.9 Can I mix in the same execution plan automated test cases executed by Squash TF and test cases executed by Squash AUTOM?

Yes.

In order to ease the transition, it is perfectly possible to have test cases from a project running Squash TF and test cases from a project running Squash AUTOM within the same Squash TM execution plan.

3.1.10 Do I need to have a Jenkins server to run my automated tests from Squash TM via Squash AUTOM?

No.

The specific Jenkins jobs required to run automated tests from Squash TM via Squash TF are no longer a prerequisite for running from Squash TM via Squash AUTOM.

With Squash AUTOM, execution is handled by the Squash Orchestrator, a specific component of Squash AUTOM.

3.1.11 Can I launch my Squash TM automated execution plans from a Jenkins pipeline with Squash AUTOM and SQUASH DEVOPS?

Yes.

Running a Squash TM execution plan from a Jenkins pipeline is a new feature of Squash DEVOPS compared to Squash TF and requires the implementation of jobs as described in the Squash DEVOPS documentation.

This section contains a set of FAQ about **Squash AUTOM** and **Squash DEVOPS**.

Available FAQ are:

- *Generalities* : FAQ about generalities on **Squash AUTOM**, **Squash DEVOPS** and consequences for **Squash TF**