
Squash Autom & Squash Devops

Release 1.0.0-alpha1

squashtest

Mar 12, 2021

CONTENTS

1	Squash AUTOM	3
1.1	Installation Guide	3
1.2	Piloting automated tests executions with an EPAC (Execution Plan «as code»)	4
1.3	Piloting automated tests executions from Squash TM	4
2	Squash DEVOPS	13
2.1	Installation Guide	13
2.2	Calling the Squash Orchestrator from a Jenkins pipeline	14
2.3	Squash TM test execution plan retrieval with a PEAC	16

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

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

SQUASH AUTOM

1.1 Installation Guide

- *Squash Orchestrator*
- *Result Publisher Plugin for Squash TM*

1.1.1 Squash Orchestrator

The **Squash Orchestrator** is available as *Docker* image on *DockerHub* (*squashtest/squash-orchestrator:1.0.0.alpha1*).

The deployment procedure can be found in the **Squash Orchestrator** documentation (*Squash Orchestrator Documentation – 1.0.0.alpha1*, .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.alpha1.jar*) freely available, or a **Premium** version (*squash.tm.rest.result.publisher.premium-1.0.0.alpha1.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-plugin-in>).

Warning: This plugin is compatible with version *1.22.1.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.alpha1*, .pdf version) downloadable from <https://www.squashtest.com/community-download>.

1.3 Piloting automated tests executions from Squash TM

- *Squash TM test case automation*
- *Test plan execution from Squash TM*
- *Results publication after a Squash TM test plan execution*

1.3.1 Squash TM test case automation

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 :

Automatisation	
Technologie du test automatisé :	Robot Framework
URL du dépôt de code source :	https://my-scm/myrepo
Référence du test automatisé :	my-repo/test.robot#firstTestCase

- 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* and *Junit* are functioning.
- Source code repository URL : The address of the source code repository where the project is located.

- `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 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 :

Tech. test auto. ▾	URL du scm ▾	Ref. test auto. ▾
--------------------	--------------	-------------------

- `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](#)).

Warning: A known issue of this version is the lack of repository name at the beginning of the reference when automatically filled by Squash TM. It is therefore necessary to add it manually. This name is the last part of the source code repository URL.

BDD or Gherkin test case

The informations of the *Automation* panel are automatically filled during the transmission of a BDD or Gherkin script to a remote source code repository host service. They can also be modified by the user at any moment.

Automation frameworks specifics

Automation with Robot Framework

In order to bind a **Squash TM** test case with 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.

Below is an exemple of a `.robot` file and the corresponding **Squash TM** test case automation :

The screenshot displays the Squash TM interface. At the top, a code editor shows the content of `robot-parm-demo / parmTest.robot`. The code includes settings for documentation and libraries, followed by test cases for a parameter test. Below the code editor, the 'Test Case Workspace' shows a tree view with 'Test Project-1' and 'Test Folder a', containing the test case '42 - test_case_robot_framework'. The main panel shows the details for this test case, including its creation and update timestamps, and tabs for 'Information', 'Test steps', 'Parameters', 'Attachments', and 'Executions'. The 'Automation' panel at the bottom provides details about the automated test technology, source code repository URL, and the specific test reference.

```

1  *** Settings ***
2  Documentation      Example of Squash TF parameter use.
3  Library            squash_tf.TFParamService
4  Library            conditionalHang.py      42
5
6  *** Test Cases ***
7  Parameter Test
8      [Documentation]  This test hangs, fails or passes depending on parameter value
9      ${parmValue}=    Get Param      TC_REFERENCE
10     Hang If Not      ${parmValue}
11     Should Be Equal  ${parmValue}    42
12

```

Test Case Workspace

42 - test_case_robot_framework

Created on : 2021/02/04 14:12 (admin)
Updated on : 2021/02/04 14:12 (admin)

Information | Test steps | Parameters | Attachments | Executions

Description [ID = 239]

Automation

Automated test technology : Robot Framework
Source code repository URL : https://bitbucket.org/.../robot-parm-demo
Automated test reference : robot-parm-demo/parmTest.robot#Parameter Test

Automation with JUnit

In order to bind a **Squash TM** test case with 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
10
11
12
13
14
15
16
17
18
19
20
21
22
23
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  }

```

The screenshot displays the Squash TM Test Case Workspace. On the left, a tree view shows the project structure: 'Test Project-1' containing 'Test Folder a', which in turn contains '42 - test_case_robot_framework' and 'test_case_junit'. The main panel shows details for 'test_case_junit' (ID = 240). It includes a description field, tabs for 'Information', 'Test steps', 'Parameters', 'Attachments', and 'Executions', and an 'Automation' section. The automation details are as follows:

Automation	
Automated test technology :	JUnit
Source code repository URL :	https://github.com/ /junitCalc
Automated test reference :	junitCalc/squash.tfauto.CalculatorTest#multFailure

1.3.2 Test plan execution from Squah TM

This fonctionnnality is not available in the *1.0.0.alpha1* version.

It is however possible to trigger the execution of a **Squash TM** test plan from a *Jenkins* pipeline. Please refer to the *Squash DEVOPS* user guide.

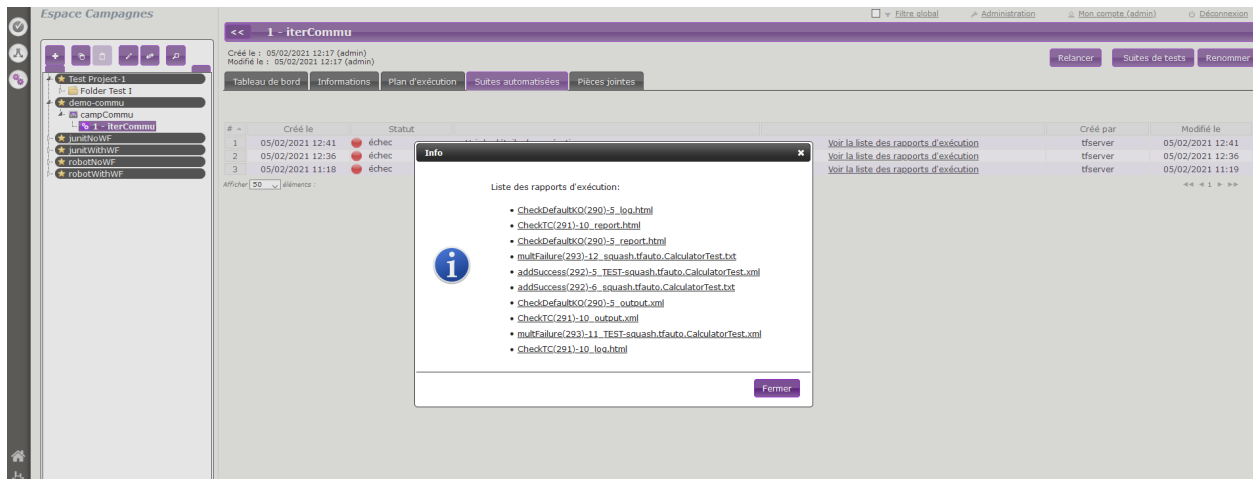
1.3.3 Results publication after a Squash TM test plan execution

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 the usage of 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 informations are updated :

- ITPIs status update.
- Automated suite status update.
- The various ITPIs execution reports are accessible from the *Automated Suites* tab of the iteration or test suite :



This, however, doesn't happen :

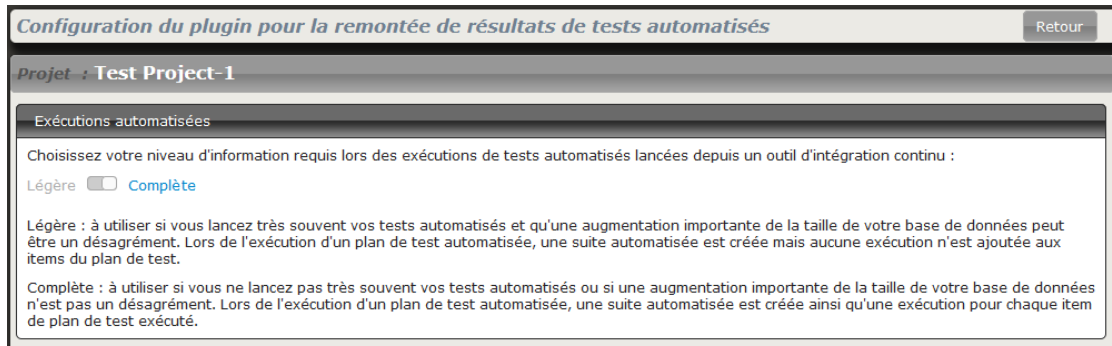
- Creation of a new execution for each executed ITPI.

Squash AUTOM Premium

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

- Light (default value).
- Complete.

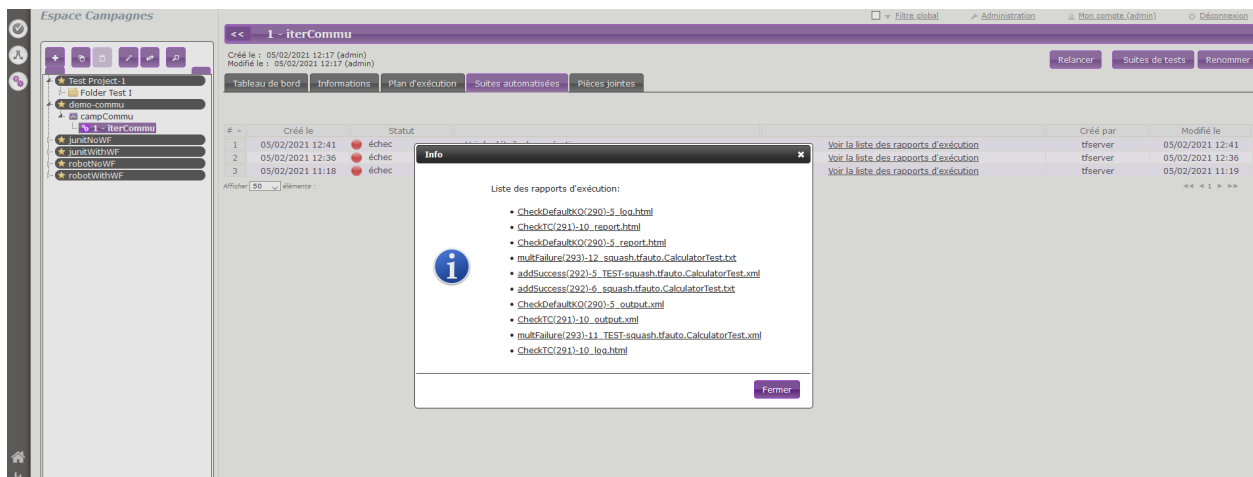
The choice of publication type is operated on a project basis by accessing the configuration of the **Squash TM Result Publisher** plugin from the *Plugins* tab of your project page, inside the *Administration* :



Light results publication

By choosing the “Light” results publication, the following informations are updated after the execution of a **Squash TM** test plan (iteration or test suite) :

- ITPIs status update.
- Automated suite status update.
- The various ITPIs execution reports are accessible from the *Automated Suites* tab of the iteration or test suite :



This, however, doesn't happen :

- Creation of a new execution for each executed ITPI.

Complete results publication

By choosing the “Complete” results publication, the following informations are 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 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) :

Espace Campagnes

Créé le : 03/03/2021 18:24 (admin)
Modifié le : 03/03/2021 18:33 (admin)

Relancer Lancer les tests automatisés Suites de tests Renommer

Tableau de bord Informations Plan d'exécution Suites automatisées Pièces jointes

Filtrer Réordonner Suites de tests Statut Assigner Ajouter Retirer du plan d'exécution

#	Emplacement	Mode	Ref.	Test	Imp.	Jeux de données	Suite de tests	Statut	% succès	Utilisateur	Dernière exécution
1	robotWF	z	z	checkCPG	F	-	-	succès	100 %	Squash Administrator (admin)	03/03/2021 18:33
2	robotWF	z	z	checkDSNAME	F	dataset1	-	succès	100 %	Squash Administrator (admin)	03/03/2021 18:33
3	robotWF	z	z	checkDSPARAM	F	ds	-	succès	100 %	Squash Administrator (admin)	03/03/2021 18:33
4	robotWF	z	z	checkDefaultFailure	F	-	ts	échec	0 %	Squash Administrator (admin)	03/03/2021 18:34
<p>Exec. 1 : checkDefaultFailure - échec admin 03/03/2021 17:33</p> <p>Exec. 2 : checkDefaultFailure - échec admin 03/03/2021 17:34</p> <p>Nouvelle exécution Exécuter automatiquement</p>											
5	robotWF	z	z	checkIT	F	-	ts	succès	100 %	Squash Administrator (admin)	03/03/2021 18:34
6	robotWF	z	z	checkTC	F	-	-	succès	100 %	Squash Administrator (admin)	03/03/2021 18:33
7	robotWF	z	z	checkTS	F	-	ts	succès	100 %	Squash Administrator (admin)	03/03/2021 18:34

Afficher 50 éléments :

Espace Campagnes □ Filtre global Administration Mon compte (admin.) Déconnexion

Exécution : #1 - checkDefaultFailure Retour

Référence :
Description :
Statut : ● 1-En cours de rédaction
Script auto : (supprimé)

Attributs

Importance : ▼ 4-Faible
Nature : Non définie
Type : Non défini
testcase (issu du cas de test) : testcaseValue

Prérequis

Exigences vérifiées

#	Projet	ID version	Référence	Exigence	Criticité
Aucun élément à afficher					

Affichage 0 à 0 sur 0 élément(s)

Champs personnalisés

Résumé du résultat

Scénario d'exécution

#	Action	Résultat att.	Statut	Dernière exec.	Utilisateur	Commentaires	PJ.	Exec.
Aucun élément à afficher								

Afficher 50 éléments : ◀◀ 1 ▶▶

Commentaires

(Cliquez pour éditer...)

Pièces jointes Ajouter une pièce jointe Organiser

[checkDefaultFailure\[306\]-output.xml](#)
[checkDefaultFailure\[306\]-log.html](#)
[checkDefaultFailure\[306\]-report.html](#)

This guide will show you the various possibilities offered by the version *1.0.0.alpha1* 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.alpha1* 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 informations towards **Squash TM** at the end of the execution of a **Squash TM** execution plan by the **Squash Orchestrator**.

SQUASH DEVOPS

2.1 Installation Guide

- *Squash Orchestrator*
- *Test Plan Retriever plugin for Squash TM*
- *Squash DEVOPS plugin for Jenkins*

2.1.1 Squash Orchestrator

This micro-service exists as a **Squash DEVOPS Community** and a **Squash DEVOPS Premium** versions. 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.alpha1*, .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.alpha1.jar*) freely available, or a **Premium** version (*squash.tm.rest.test.plan.retriever.premium-1.0.0.alpha1.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-plugin-in>).

Warning: This plugin is compatible with version *1.22.1.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.alpha1.hpi*).

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

Jenkins · Gestion des plugins

Nom d'utilisateur

Mot de passe

No Proxy Host

Envoyer

Soumettre un plugin

Vous pouvez téléverser un fichier .hpi pour installer un plugin extérieur au dépôt centralisé de plugin.

Fichier: Aucun fichier sélectionné.

Soumettre

Site de mise à jour

URL

Envoyer

Update information obtained: 1 j 1 h ago [Vérifier maintenant](#)

REST API Jenkins 2.249.2

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 :

System Configuration



Configurer le système

Configurer les paramètres généraux et les chemins de fichiers.



Configuration globale des outils

Configurer les outils, leur localisation et les installateurs automatiques.



Gestion des plugins

Ajouter, supprimer, activer ou désactiver des plugins qui peuvent étendre les fonctionnalités de Jenkins.
▲ mises à jour disponibles



Gérer les nœuds

Ajouter, supprimer, contrôler et monitorer les divers nœuds que Jenkins utilise pour exécuter les jobs.

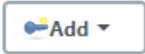


Configuration files

administration des fichiers de configurations tels que settings.xml pour Apache Maven.

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

Squash Orchestrator servers

Server id	46705135
Server name	defaultServer
Receptionist endpoint URL	http://127.0.0.1:7774
Workflow Status endpoint URL	http://127.0.0.1:7775
Credential	- none - 
Workflow Status poll interval	2S
Workflow creation timeout	5S

- `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 = runSquashTFWorkflow(
            workflowPathName:'testWorkflow.json',
            workflowTimeout: '20S',
            serverName:'defaultServer'
        )
        echo "We just ran The Squash Orchestrator workflow $workflow_id"
    }
}
```

The *runSquashTFWorkflow* 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

- *Prerequisites*
- *Integration of the Squash TM execution plan retrieval step into an EPAC*
- *Squash TM parameters to exploit in a Robot Framework 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).

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 <i>tagLabel</i> or <i>tagValue</i> fields is present, the other must also be specified.
--















2.3.3 Squash TM parameters to exploit in a Robot Framework automated test

By executing an EPAC retrieving a **Squash TM** execution plan, **Squash TM** passes various informations on ITPIs that can be exploited in a *Robot Framework* test case.

Nature of the Squash TM parameters to exploit

The exploitable **Squash TM** parameters in a *Robot Framework* script will differ depending on the usage of a **Squash DEVOPS Community** or **Squash DEVOPS Premium** licence.

Here is a table of the exploitable parameters :

Nature	Key	Community	Premium
Name of the dataset	DSNAME		
Parameter of a dataset	DS_[name]		
Reference of a test case	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 the “Code” field of a CUF*
- [name] : *Name of the parameter as filled in Squash TM*

Usage of Squash TM parameters in a Robot Framework test case

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

In order to achieve this, you need to follow these steps :

- Install on the environment(s) where the *Robot Framework* execution takes place the *squash-tf-services* python library. 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>
```

Here is an example of a *Robot Framework* test case exploiting **Squash TM** parameters :

```
robot-parm-demo / parmTest.robot  Edit  ...  
1  *** Settings ***  
2  Documentation      Example of Squash TF parameter use.  
3  Library             squash_tf.TFParamService  
4  Library             conditionalHang.py      42  
5  
6  *** Test Cases ***  
7  Parameter Test  
8      [Documentation]  This test hangs, fails or passes depending on parameter value  
9      ${parmValue}=    Get Param      TC_REFERENCE  
10     Hang If Not      ${parmValue}  
11     Should Be Equal  ${parmValue}    42  
12
```

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.alpha1** user guide for more informations (see [here](#)).

This guide will show you the various possibilities offered by the version *1.0.0.alpha1* 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.alpha1* 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 informations on the **Squash Orchestrator** and the EPAC.
- **Test Plan Retriever for Squash TM** : this plugin for **Squash TM** allows the sending to the **Squash Orchestrator** of details about a **Squash TM** execution plan.
- **Squash DEVOPS plugin for Jenkins** : this plugin for *Jenkins* facilitates the sending of an EPAC to the **Squash Orchestrator** from a *Jenkins* pipeline.