
Squash Keyword Framework Documentation

squahstest

Apr 08, 2020

Contents

1	Getting started	1
1.1	An example to begin with	1
1.2	An example to go further	14
2	Introduction to Squash Keyword Framework (SKF)	29
2.1	Automated Project Structure	29
2.2	Test Case	30
2.3	Macros	31
3	Create a new SKF automation Project	33
3.1	Create a Squash TF Project with IntelliJ	33
3.2	Create a Squash TF Project with Squash TA Toolbox	39
3.3	Create a Squash TF Project using a command line	47
3.4	Default SKF automation project pom.xml	52
4	Writing tests	57
4.1	Sections	57
4.2	Resource Components	60
4.3	Macros	64
4.4	Ecosystem	67
4.5	Writing tests - Advanced Users	71
5	Execution and Reporting	87
5.1	Logging	87
5.2	Tests Execution and Reporting	88
5.3	List tests in an SKF project	109
5.4	Check TF metadata in project	112
6	SKF Plugins	117
6.1	Commons component plugin	117
6.2	Database Plugin	160
6.3	Filechecker Plugin	235
6.4	FTP Plugin	255
6.5	JUnit Plugin	271
6.6	Local process Plugin	278
6.7	MEN XML Checker Plugin	296
6.8	SAHI Plugin	303

6.9	Selenium Plugin	310
6.10	Selenium Plugin Legacy	317
6.11	SoapUI Plugin	329
6.12	SSH/SFTP Plugin	338
6.13	XML Functions Plugin	347
7	Tutorials	367
7.1	Automated Tests Rudiments	367
8	Overview	375
9	A small example to illustrate	377
10	SKF benefits	379

1.1 An example to begin with

Contents :

- *Create a project*
- *Configure database*
- *Create SKF script*
- *Execute an SQL script to create a table in database*
- *Populate the database table with a DbUnit dataset*
- *Test that our table contains expected data with a DbUnit dataset*
- *Test that our table contains all the expected data*
- *Clean the database*

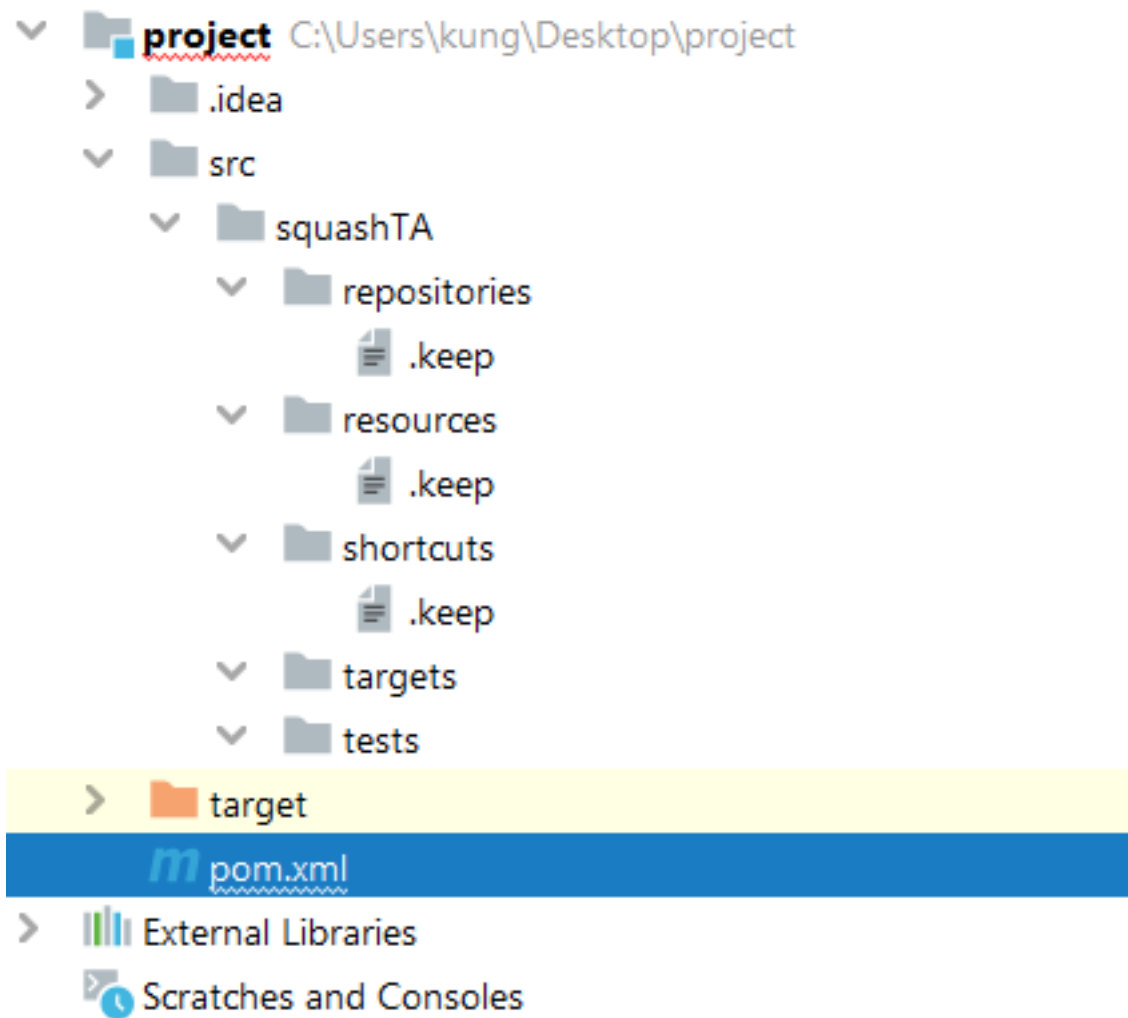
In this example, we will show you a simple SKF script that uses macros to do tests against an H2 database in embedded mode.

1.1.1 Create a project

First of all you need to open your favorite IDE and create a new maven project with squash-ta-archetype.

If you don't know how to generate a maven archetype, you can follow our [guide](#).

Delete all the samples in the folders of the generated project. Just keep the structure :



1.1.2 Configure database

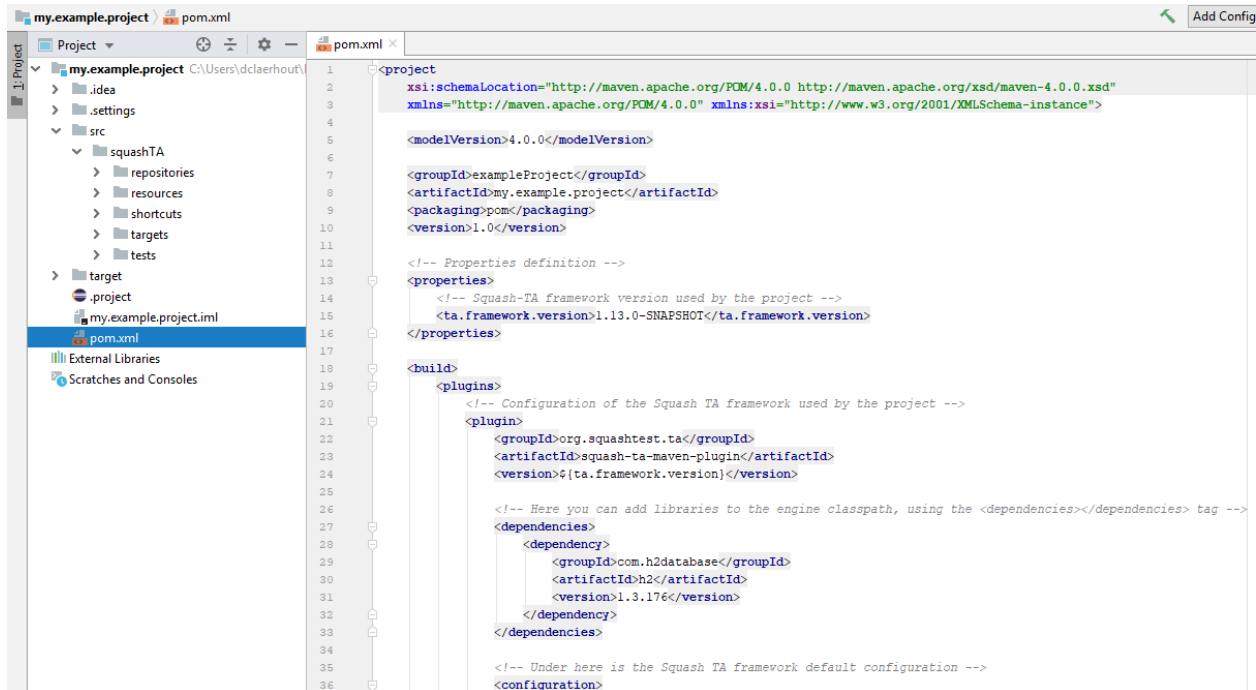
For the database, you need to add the following dependency to your **POM** file :

```

<dependencies>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.3.176</version>
  </dependency>
</dependencies>

```

Your **POM** file should look like this :



In the targets folder (be careful about the name, targets != target), you need to create a **.properties** file.

The **.properties** file should have the following properties :

- **#!db** : The shebang to indicate that this file contains informations about a database.
- **squashtest.ta.database.driver** : The driver to be used.
- **squashtest.ta.database.url** : The path to the database.
- **squashtest.ta.database.username** (optional, not used in our example) : The username to use to connect to the database.
- **squashtest.ta.database.password** (optional, not used in our example) : The password to use to connect to the database.

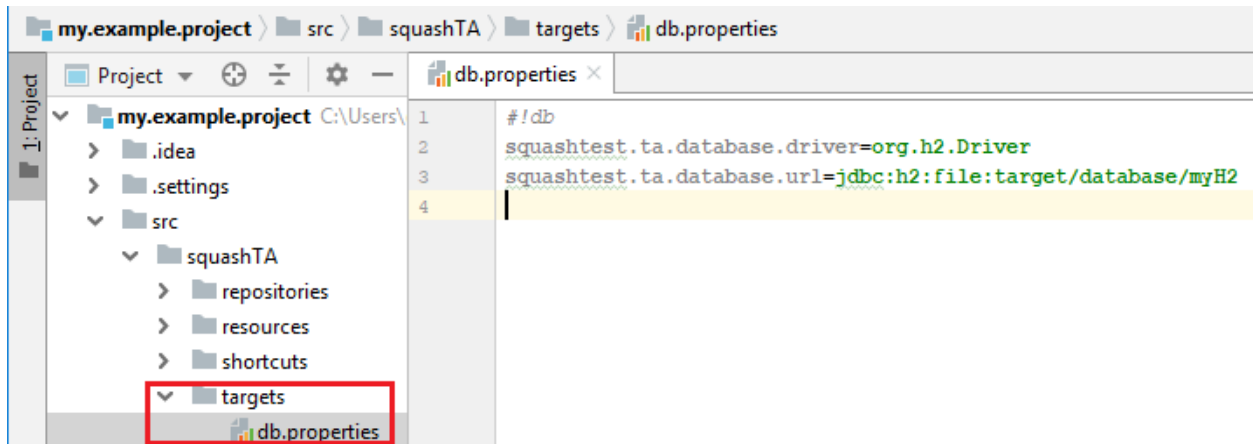
In our example, it will be as follow :

```

#!db
squashtest.ta.database.driver=org.h2.Driver
squashtest.ta.database.url=jdbc:h2:file:target/database/myH2

```

.properties file to connect to a database :



1.1.3 Create SKF script

In **tests** folder create a **.ta** file.

In this file, write down :

```
SETUP :

TEST :

TEARDOWN :
```

We will use those 3 phases in our example.

1.1.4 Execute an SQL script to create a table in database

First of all, during the **SETUP** phase, we want to create a new table in our H2 database.

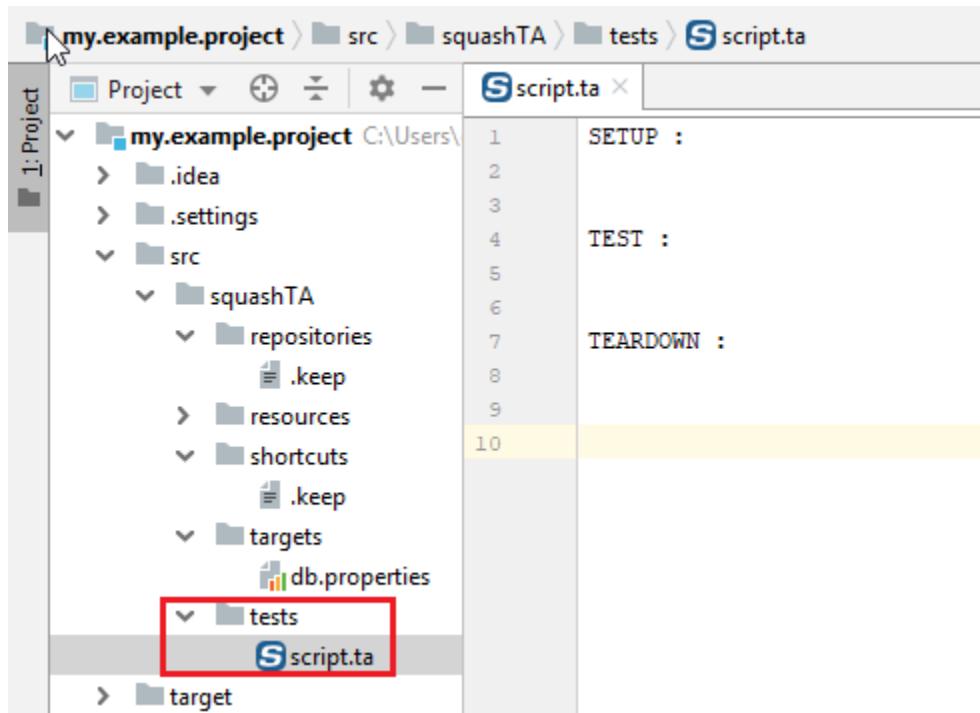
To do so, we need to create a **.sql** script file in **resources** folder. It's good practice to create different subfolders for each type of resources (sql, selenium, soapui, etc).

Here is the script :

```
DROP TABLE IF EXISTS PETS_STORE;

CREATE TABLE PETS_STORE (
    ID INT NOT NULL,
    ANIMAL VARCHAR(45) NULL,
```

(continues on next page)

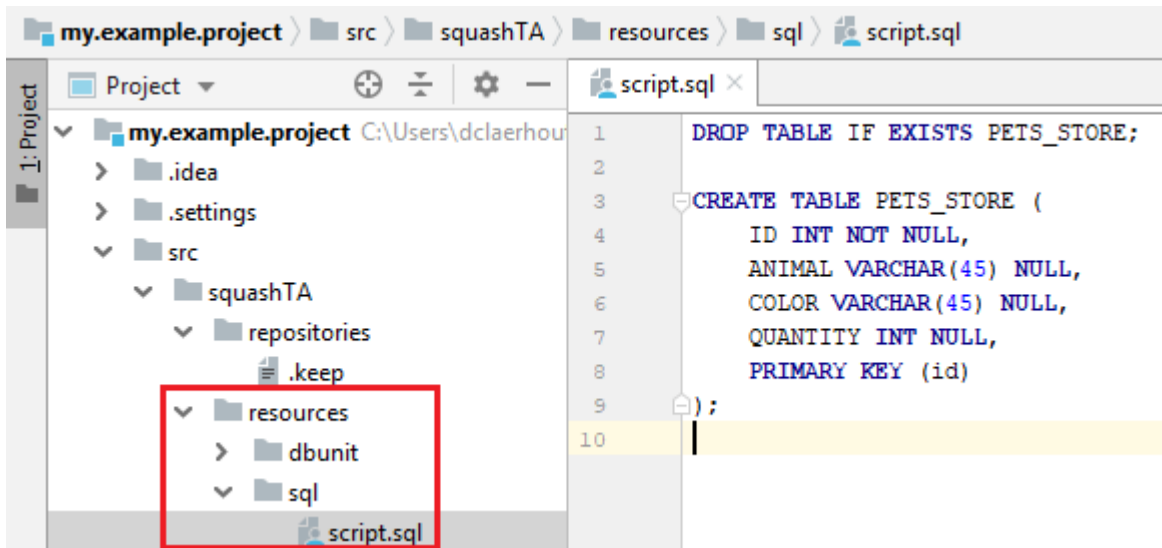


(continued from previous page)

```

COLOR VARCHAR(45) NULL,
QUANTITY INT NULL,
PRIMARY KEY (id)
);

```



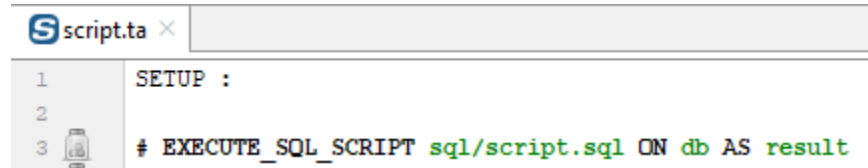
In the SKF script, add the following macro to your **SETUP** phase :

```
# EXECUTE_SQL_SCRIPT {file} ON {database} AS {result}
```

{file} : The SQL script, we have just created. Give the path of the file in the **resources** folder.

{database} : The database we want to operate the script on. Give the name of the .properties file you have created in the **targets** folder (without the .properties extension).

{result} : A free identifier for the result. As the 'execute' command with an sql script return an empty resource, this result resource will also be empty.



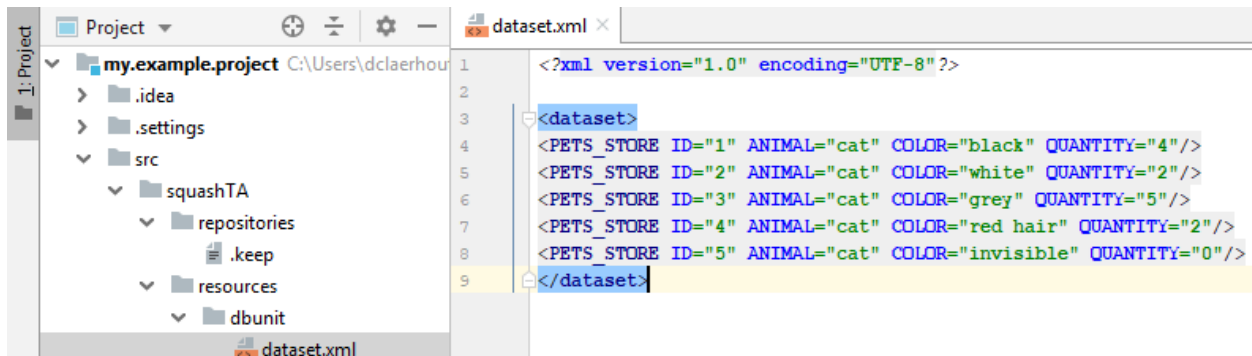
1.1.5 Populate the database table with a DbUnit dataset

To populate the table, we will use a DbUnit dataset.

Create an **.xml** file in resources folder. You should also create a **dbunit** subfolder.

In this file, write down the following :

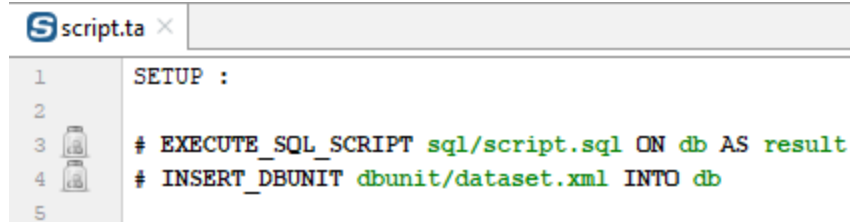
```
<?xml version="1.0" encoding="UTF-8"?>  
  
<dataset>  
  <PETS_STORE ID="1" ANIMAL="cat" COLOR="black" QUANTITY="4"/>  
  <PETS_STORE ID="2" ANIMAL="cat" COLOR="white" QUANTITY="2"/>  
  <PETS_STORE ID="3" ANIMAL="cat" COLOR="grey" QUANTITY="5"/>  
  <PETS_STORE ID="4" ANIMAL="cat" COLOR="red hair" QUANTITY="2"/>  
  <PETS_STORE ID="5" ANIMAL="cat" COLOR="invisible" QUANTITY="0"/>  
</dataset>
```



In the SKF script, add the following macro to your **SETUP** phase :

```
# INSERT_DBUNIT {dataset} INTO {database}
```

{dataset} : The **.xml**, we have just created. Give the path of the file in the **resources** folder.



{database} : The database we want to operate the script on. Give the name of the .properties file you have created in the **targets** folder (without the .properties extension).

1.1.6 Test that our table contains expected data with a DbUnit dataset

First we will do an incorrect dataset so that the assertion executed by the script fails.

Create a new .xml file in the **resources/dbunit** folder.

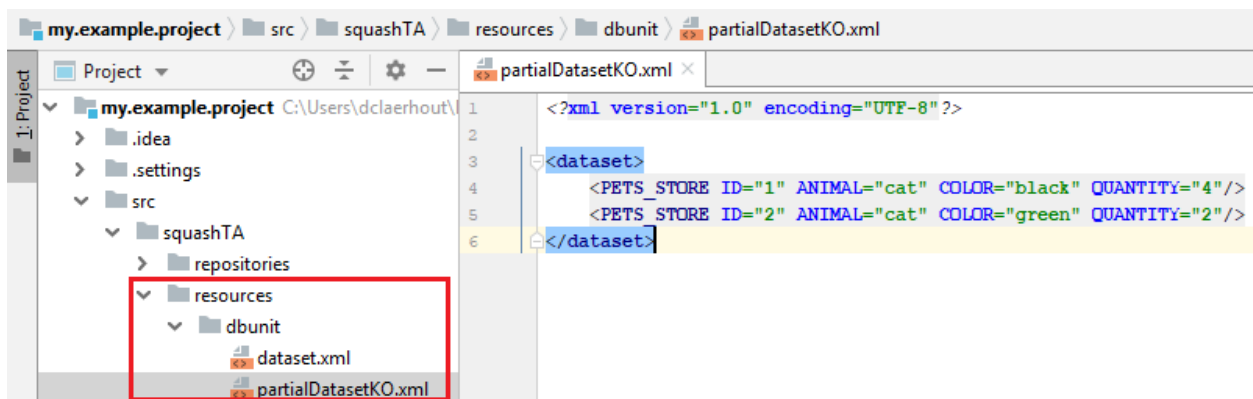
Write down the following dataset :

```

<?xml version="1.0" encoding="UTF-8"?>

<dataset>
  <PETS_STORE ID="1" ANIMAL="cat" COLOR="black" QUANTITY="4"/>
  <PETS_STORE ID="2" ANIMAL="cat" COLOR="green" QUANTITY="2"/>
</dataset>

```



In the SKF script, add the following macro to your **TEST** phase :

```
# ASSERT_DBUNIT TARGET {database} CONTAINS {dataset}
```

Now we are going to execute the script. Use the following maven command to build your project :

```
mvn squash-ta:run
```

```

1  SETUP :
2
3  # EXECUTE_SQL_SCRIPT sql/script.sql ON db AS result
4  # INSERT_DBUNIT dbunit/dataset.xml INTO db
5
6  TEST :
7
8  # ASSERT_DBUNIT TARGET db CONTAINS dbunit/partialDatasetKO.xml

```

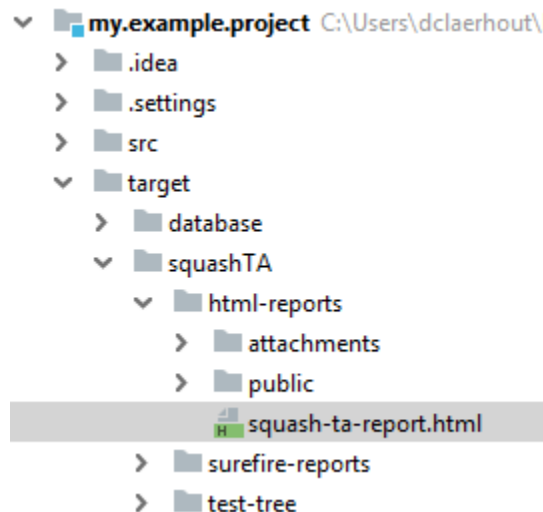
```

16:37:46,816 - [ERROR] The execution failed in the TEST phase of the TA script 'script.ta' with the message: 'The first dataset did not contain the second one.'
16:37:46,842 - [WARN] Potential problem found: The configured data type factory 'class org.dbunit.dataset.datatype.DefaultDataTypeFactory' might cause problems
ht see this message because the list of supported database products is incomplete (list=[derby]). If so please request a java-class update via the forums.If you
bProducts() to specify the supported database products.
16:37:46,848 - [INFO] Exporting results
16:37:47,249 - [INFO] Cleaning resources
16:37:47,250 - [INFO] Squash TF : build complete.
16:37:47,325 - [WARN] All the files from C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190926_163743_0487252961668765742652 could not be deleted.
16:37:47,326 - [INFO] -----
16:37:47,326 - [INFO] BUILD FAILURE
16:37:47,326 - [INFO] -----

```

After the execution, an HTML report is generated. It can give further details about the reason of the failure.

You can access this report in **target/squashTA/html-reports** folder :



Open this report with the web browser of your choice :

You can the differences between the dataset and the database by opening **EXECUTION_REPORT-diff** in the attachments :


Now we are going to create a new **.xml** file with a correct dataset :


```

<?xml version="1.0" encoding="UTF-8"?>


<dataset>
  <PETS_STORE ID="1" ANIMAL="cat" COLOR="black" QUANTITY="4"/>
  <PETS_STORE ID="2" ANIMAL="cat" COLOR="white" QUANTITY="2"/>
</dataset>

```











 tests

Script	Status	Time (s)
 script.ta	Failure	0,232

[\[Back to top\]](#)

 script.ta

Status: Failure
Time (s): 0,232

Test steps	Line
SETUP :	
 # EXECUTE_SQL_SCRIPT sql/script.sql ON db AS result	3
 # INSERT_DBUNIT dbunit/dataset.xml INTO db	4
TEST :	
 <div> <div># ASSERT_DBUNIT TARGET db CONTAINS dbunit/partialDatasetKO.xml</div> <div>  EXECUTE get.all WITH \$() ON db AS __actual_8.dbu  LOAD dbunit/partialDatasetKO.xml AS __temp9.file  CONVERT __temp9.file TO file(param.relateddate) AS __temp_a.file  CONVERT __temp_a.file TO xml(structured) AS __temp_b.xml  CONVERT __temp_b.xml TO dataset.dbunit(dataset) AS __expected_c.dbu  ASSERT __actual_8.dbu DOES contain THE __expected_c.dbu </div> <div>  The first dataset did not contain the second one. </div> </div>	8

Attachments

[EXECUTION_REPORT-diff](#)
[EXECUTION_REPORT-actual](#)
[EXECUTION_REPORT-expected](#)

[\[Back to top\]](#) [\[Back to tests\]](#)

EXECUTION_REPORT-diff

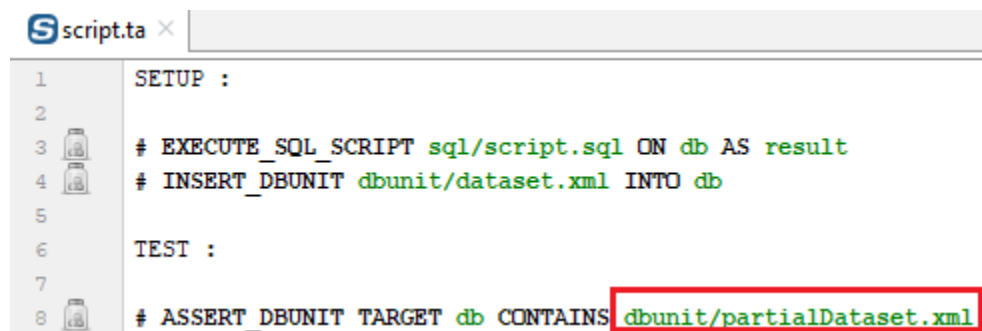
```

1 The dataset did not contain the expected data.
2
3 * Table "pets_store" did not contains the following row(s):
4   - {id='2'} {animal='cat'} {color='green'} {quantity='2'}
5     --> Several matches were found by reducing the search to the primary key column(s). The first match is:
6     - {id='2'} {animal='cat'} {color='white'} {quantity='2'}
7
8 Note : The (pseudo) primary key used for table "pets_store" is: [ID].

```



Don't forget to change the dataset used in the SKF script :



If you execute the script again, you should have a build SUCCESS.

1.1.7 Test that our table contains all the expected data

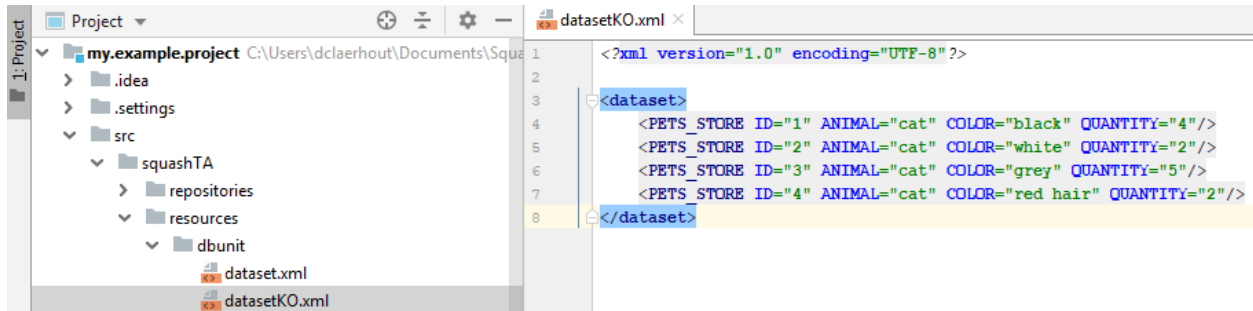
As in the previous example, we will start with an incorrect dataset.

Create a new **.xml** file in the **resources/dbunit** folder and write down the following dataset :

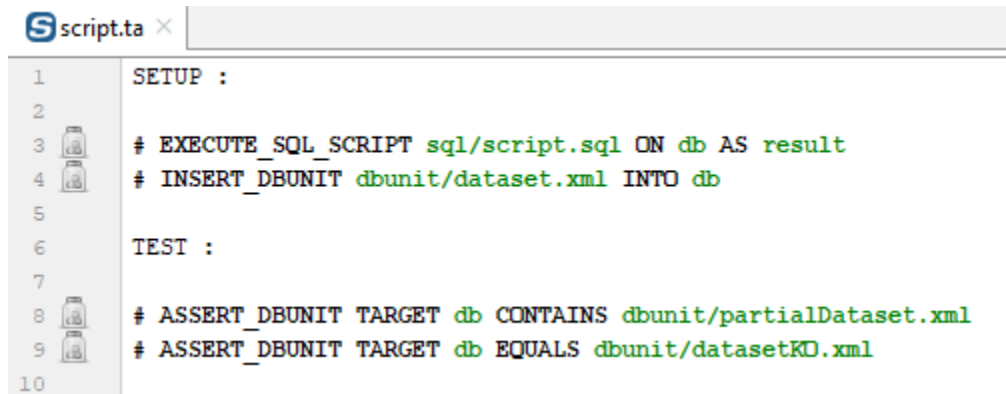
```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <PETS_STORE ID="1" ANIMAL="cat" COLOR="black" QUANTITY="4"/>
  <PETS_STORE ID="2" ANIMAL="cat" COLOR="white" QUANTITY="2"/>
  <PETS_STORE ID="3" ANIMAL="cat" COLOR="grey" QUANTITY="5"/>
  <PETS_STORE ID="4" ANIMAL="cat" COLOR="red hair" QUANTITY="2"/>
</dataset>
```

The invisible cat is missing.

In the SKF script, add the following macro to your **TEST** phase :



```
# ASSERT_DBUNIT TARGET {database} EQUALS {dataset}
```



Execute the script. You should have a build failure with the following error :

```
org.dbunit.dataset.DataSetException: Table 'PETS_STORE' contains 1 more row(s) in the first dataset than in the second one.
```

You can open the HTML report to have more details :

In SKF script, change the dataset in the last macro and use the first one we created to populate the table :

If you execute the script again, you should have a build SUCCESS.

1.1.8 Clean the database

The last thing we want to do is to clean the database after the execution of the test.

In SKF script, add the following macro in **TEARDOWN** phase :

```
# DELETE_DBUNIT {dataset} FROM {database}
```

● tests

Script	Status	Time (s)
● script.ta	Failure	0,264

[\[Back to top\]](#)

● script.ta

Status: Failure

Time (s): 0,264

	Test steps	Line
	SETUP :	
●	# EXECUTE_SQL_SCRIPT sql/script.sql ON db AS result	3
●	# INSERT_DBUNIT dbunit/dataset.xml INTO db	4
	TEST :	
●	# ASSERT_DBUNIT TARGET db CONTAINS dbunit/partialDataset.xml	8
●	<div> <div># ASSERT_DBUNIT TARGET db EQUALS dbunit/datasetKO.xml</div> <div> <div>EXECUTE get.all WITH \$() ON db AS __actual_d.dbu</div> <div>LOAD dbunit/datasetKO.xml AS __tempe.file</div> <div>CONVERT __tempe.file TO file(param.relativeDate) AS __temp_f.file</div> <div>CONVERT __temp_f.file TO xml(structured) AS __temp_10.xml</div> <div>CONVERT __temp_10.xml TO dataset.dbunit(dataset) AS __expected_11.dbu</div> <div> <div>ASSERT __actual_d.dbu IS equal THE __expected_11.dbu</div> <div> <div>⊕ The two compared datasets are different.</div> </div> </div> </div> </div>	9

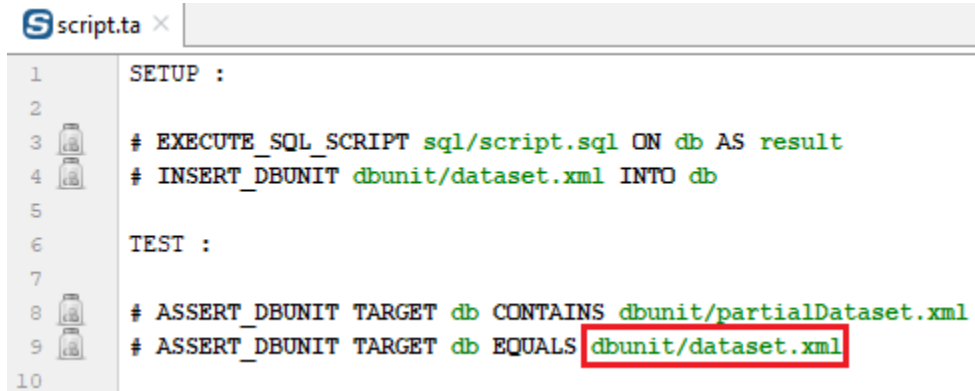
Attachments

[EXECUTION_REPORT-diff](#)

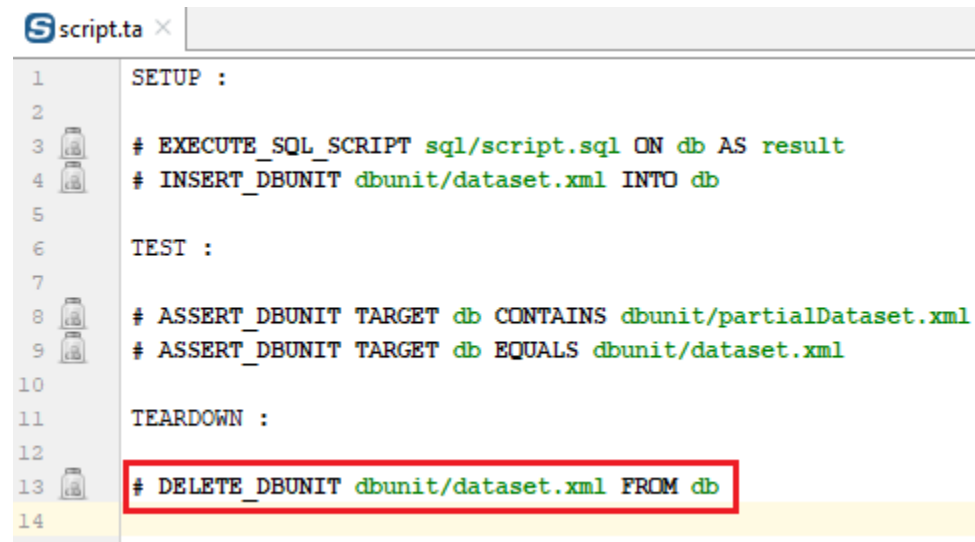
[EXECUTION_REPORT-actual](#)

[EXECUTION_REPORT-expected](#)

[\[Back to top\]](#) [\[Back to tests\]](#)



```
1  SETUP :
2
3  # EXECUTE_SQL_SCRIPT sql/script.sql ON db AS result
4  # INSERT_DBUNIT dbunit/dataset.xml INTO db
5
6  TEST :
7
8  # ASSERT_DBUNIT TARGET db CONTAINS dbunit/partialDataset.xml
9  # ASSERT_DBUNIT TARGET db EQUALS dbunit/dataset.xml
10
```



```
1  SETUP :
2
3  # EXECUTE_SQL_SCRIPT sql/script.sql ON db AS result
4  # INSERT_DBUNIT dbunit/dataset.xml INTO db
5
6  TEST :
7
8  # ASSERT_DBUNIT TARGET db CONTAINS dbunit/partialDataset.xml
9  # ASSERT_DBUNIT TARGET db EQUALS dbunit/dataset.xml
10
11  TEARDOWN :
12
13  # DELETE_DBUNIT dbunit/dataset.xml FROM db
14
```

1.2 An example to go further

Contents:

- *User Story*
- *Pre-requisites*
- *Context*
- *Structure of the test file (or SKF script)*
- *Create a project*
- *Interact with the database*
- *Create an SKF Script*
- *Change host address*
- *Test the webservice*
- *Clean the database*
- *Execution*
- *Reporting*

1.2.1 User Story

We want to test a few web services in a SUT (System Under Test).

In order to test the web services, we'll also need to inject some SQL.

Since we want to be able to use our test wherever we want, we can change the host address easily.

Our web service testing tool will be SoapUI.

To end our test, we should clean the database.

1.2.2 Pre-requisites

You will need [docker](#) and [docker-compose](#) installed on your system.

In our example, we'll be running docker on a Linux system. The rest of the example can be executed on any OS.

1.2.3 Context

The SUT is jacksonviews. It'll be deployed with docker and coupled with a pgSQL 10 database that'll be automatically downloaded during the docker-compose.

Download the compressed image of the jacksonviews, available [here](#). Open a shell in the directory where the compressed image is located and decompress it with the following command (Linux) :

```
tar xvf jackson_image.tar
```

Once untarred, just run the following commands :

```
sudo docker load -i jacksonviews.tar
sudo docker-compose -f docker/app.yml up -d
```

The SUT will then be available on any OS at : `http://{host address}:8080`

1.2.4 Structure of the test file (or SKF script)

First of all, we should think about how to organize our test file.

Let's start with the user story.

Our main goal is to test the web services, so that'll be the **TEST phase**.

In order to test it correctly, we'll need a few dataset and specify the host where the SUT is available. That'll be the **SETUP phase**.

The clean up comes after the test phase, it is used just to rollback to the initial status. That'll be the **TEARDOWN phase**.

Additionally, if the test file is to be automated via a test management platform (for example Squash TM), we'll need a section named **METADATA** to store the credentials.

Phases:	We want to test a few web services in SUT (System Under Test):
META-DATA	To associate this test script with a Test case in a Test Management platform.
SETUP	In order to test the web services, we'll need to inject some SQL. To use our test wherever we want, we can easily change host address.
TEST	Our web service testing tool will be SoapUI.
TEAR-DOWN	To end our test, we should clean the database.

1.2.5 Create a project

Let's start with a clean project. You can use the squash-ta archetype with the latest version.

If you don't know how to generate a maven archetype, you can follow our [guide](#).

You can delete all the samples in the generated project, just keep the structure.

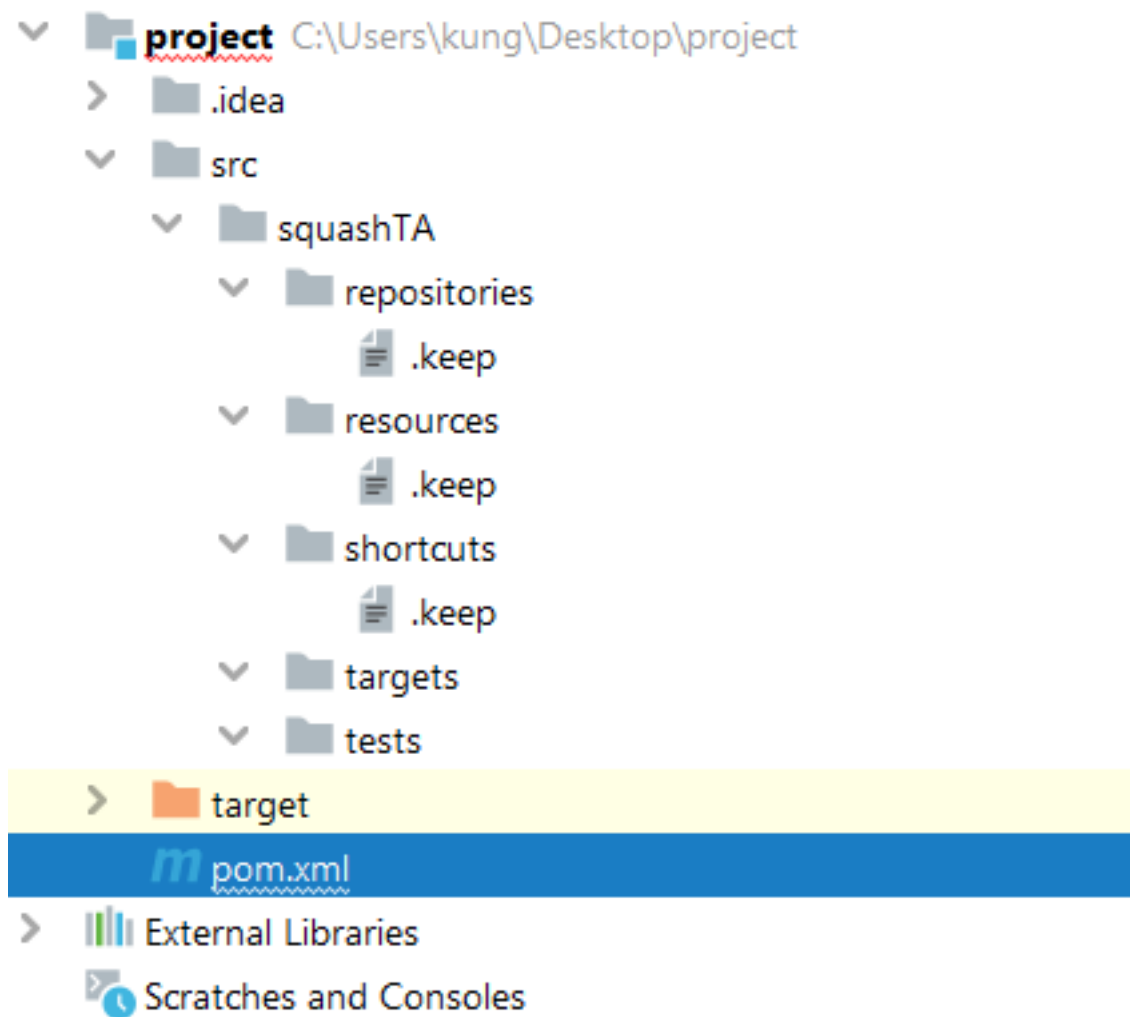


Fig. 1: Fig 1. Clean SKF project

1.2.6 Interact with the database

“In order to test the web services, we’ll need to inject some SQL.”

To interact with a database, we’ll need three things.

The **first one** is a **.properties** file put in the targets folder (be careful about the name, targets != target).

The **.properties** file should have the following properties :

- **#!db** : The shebang to indicate that this file contains informations about a database.
- **squashtest.ta.database.driver** : The driver to be used.
- **squashtest.ta.database.url** : The path to the database.
- **squashtest.ta.database.username** : The username to use to connect to the database.
- **squashtest.ta.database.password** (optional, not used in our example) : The password to use to connect to the database.

In our example, it will be as follow :

```
#!db
squashtest.ta.database.driver=org.postgresql.Driver
squashtest.ta.database.url=jdbc:postgresql://{host_address}:5432/jacksonviews
squashtest.ta.database.username=jacksonviews
```

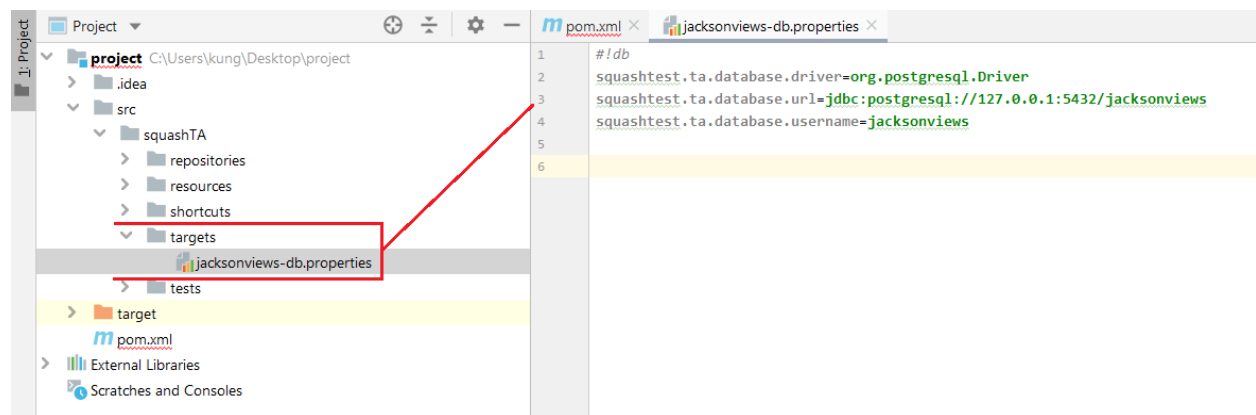


Fig. 2: Fig 2. **.properties** file to connect to a database

The **second one** is of course the query to use in the database.

We can define it inline (directly in the macro) or we can put it in a file that’ll be loaded by the macro.

In general, it’s better to put it in a file for readability and to facilitate changes.

In this case, we’ll use the **.sql** file (both options will be shown later during the **TEST phase**).

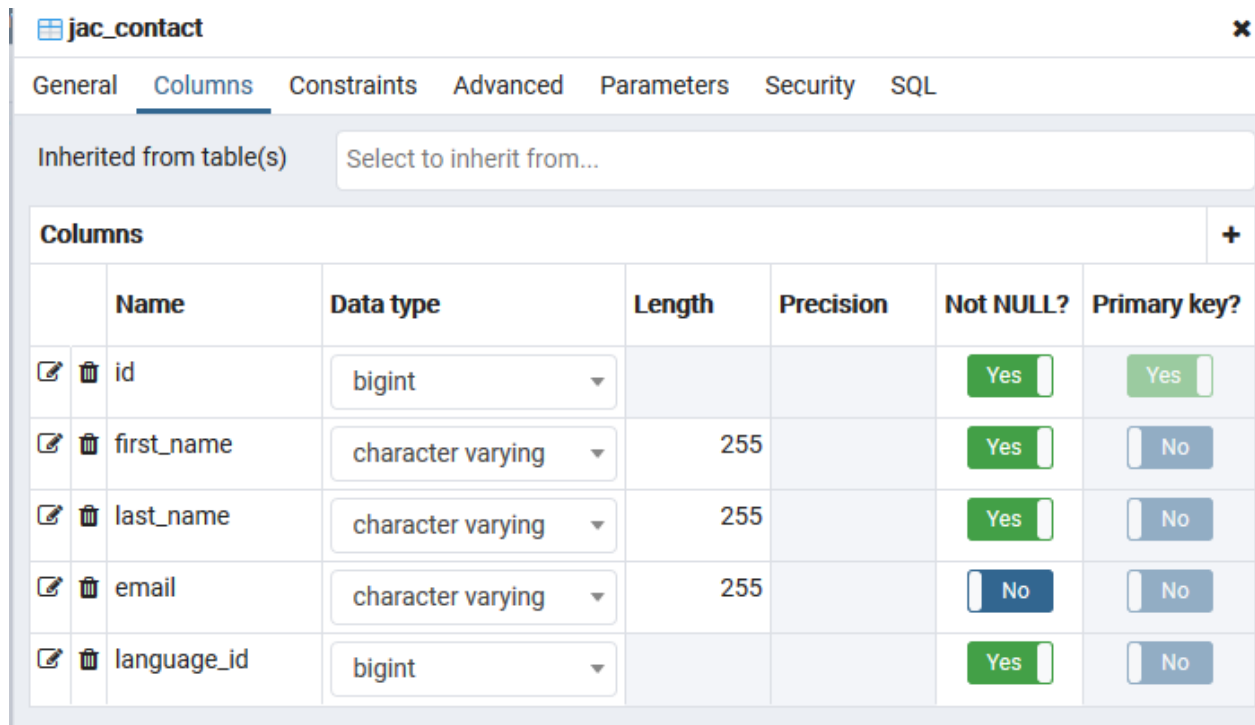
We just need to create a **.sql** file in the resources folder and write all the queries inside.

In our example, we’ll insert a row in the table **jac_contact** of our database by adding the following line to **add_contact.sql** :

```
INSERT INTO jac_contact VALUES (1, 'John', 'Smith', 'foo@foo.foo', 39);
```

It’s good practice to create different subfolders for each type of resources (sql, selenium, soapui, etc).

The **third one** is to add the jdbc driver to the **pom.xml**.



jac_contact						
General Columns Constraints Advanced Parameters Security SQL						
Inherited from table(s) Select to inherit from...						
Columns						
	Name	Data type	Length	Precision	Not NULL?	Primary key?
	id	bigint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	first_name	character varying	255		<input checked="" type="checkbox"/>	<input type="checkbox"/>
	last_name	character varying	255		<input checked="" type="checkbox"/>	<input type="checkbox"/>
	email	character varying	255		<input type="checkbox"/>	<input type="checkbox"/>
	language_id	bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>

Fig. 3: Fig 3. Contact table in the Db

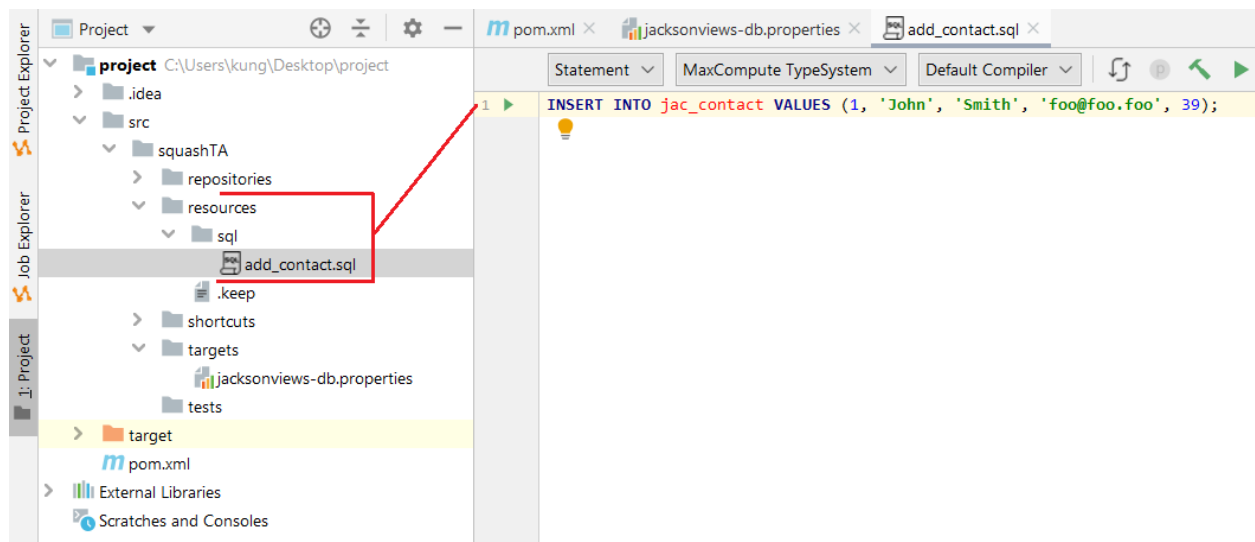


Fig. 4: Fig 4. .sql file containing the query

It's of course dependent of the database. In our case it'll be pgSQL.

The dependency is in the squash-ta-maven-plugin declaration.

```
<dependencies>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.5</version>
  </dependency>
</dependencies>
```

```
<packaging>pom</packaging>
<version>1.0-SNAPSHOT</version>

<!-- Properties definition -->
<properties>
  <!-- Squash-TA framework version used by the project -->
  <ta.framework.version>1.13.0.RC1</ta.framework.version>
</properties>

<build>
  <plugins>
    <!-- Configuration of the Squash TA framework used by the project -->
    <plugin>
      <groupId>org.squashtest.ta</groupId>
      <artifactId>squash-ta-maven-plugin</artifactId>
      <version>${ta.framework.version}</version>
    </plugin>
  </plugins>
  <dependencies>
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.2.5</version>
    </dependency>
  </dependencies>
  <!-- Under here is the Squash TA framework default configuration -->
  <configuration>

    <!--
      Uncomment the line below in order to the build finish in suc
      (functional (assertion) failure), but fail the build if an E
    -->
```

Fig. 5: Fig 5. Dependency to add in the **pom.xml**

1.2.7 Create an SKF Script

Now that we have all the prerequisites, we can create our first **.ta** script.

Since all the test files should be in the tests folder, this one is no exception.

The first thing to do is to add the **SETUP phase** that'll be used before the test itself to add the necessary data.

We can then use the following macro to execute the query on the database :

```
# EXECUTE_SQL sql/add_contact.sql ON jacksonviews-db AS query_result
```

- `sql/add_contact.sql` : The .sql file with the query
- `jacksonviews-db` : The .properties file in the targets folder to specify the database
- `query_result` : The result of the query

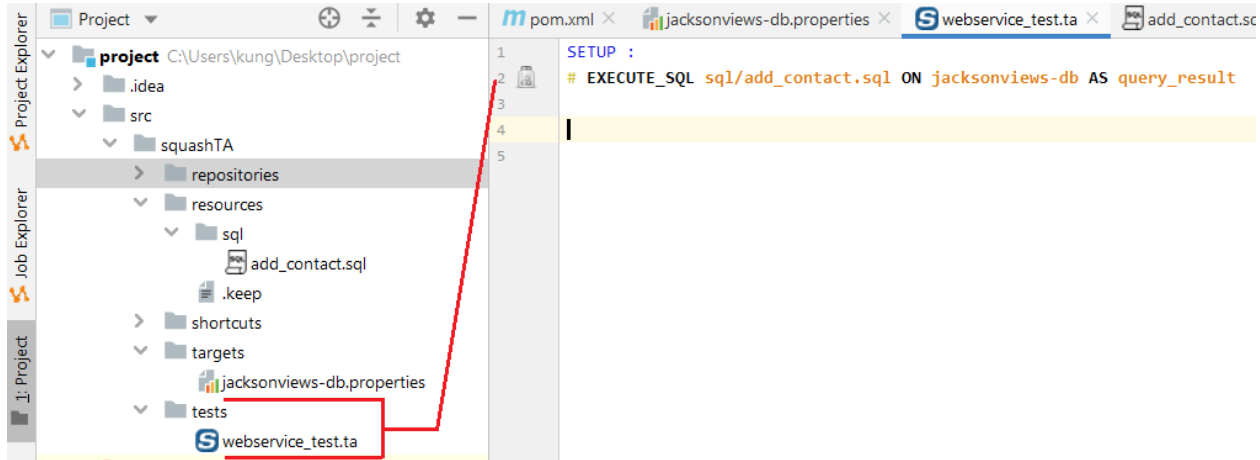


Fig. 6: Fig 6. EXECUTE_SQL macro in .ta file

For more informations on the macro, please check the following [page](#).

1.2.8 Change host address

“To use our test wherever we want, we can easily change host address.”

To change the host address, we will add the following macro to the **SETUP** phase of our script :

```
# SUBSTITUTE_KEYS IN {files} USING {key_value_list} AS {processed_files}
```

For more information, please check the following [page](#).

We’ll need this macro to change the SoapUI XML project that’ll be used later for the **TEST** phase.

Let’s add the SoapUI project to our resources folder. You can copy the content of this [file](#) in a **JacksonviewsAPI-project.xml** (or right-click and save it) and use it for our example.

Just like the .sql, it’s good practice to put it in a subfolder (a **soapui** folder in our example).

To perform the substitution of the host address specified in the project’s xml file, we added placeholders defined with `$(key_name)`, and the value will be stored in a `key=value` list that can be defined inline or in a .properties file.

In our case we’ll make a **host_info.properties** that we’ll place at the base of the resources folder, and specify the host address :

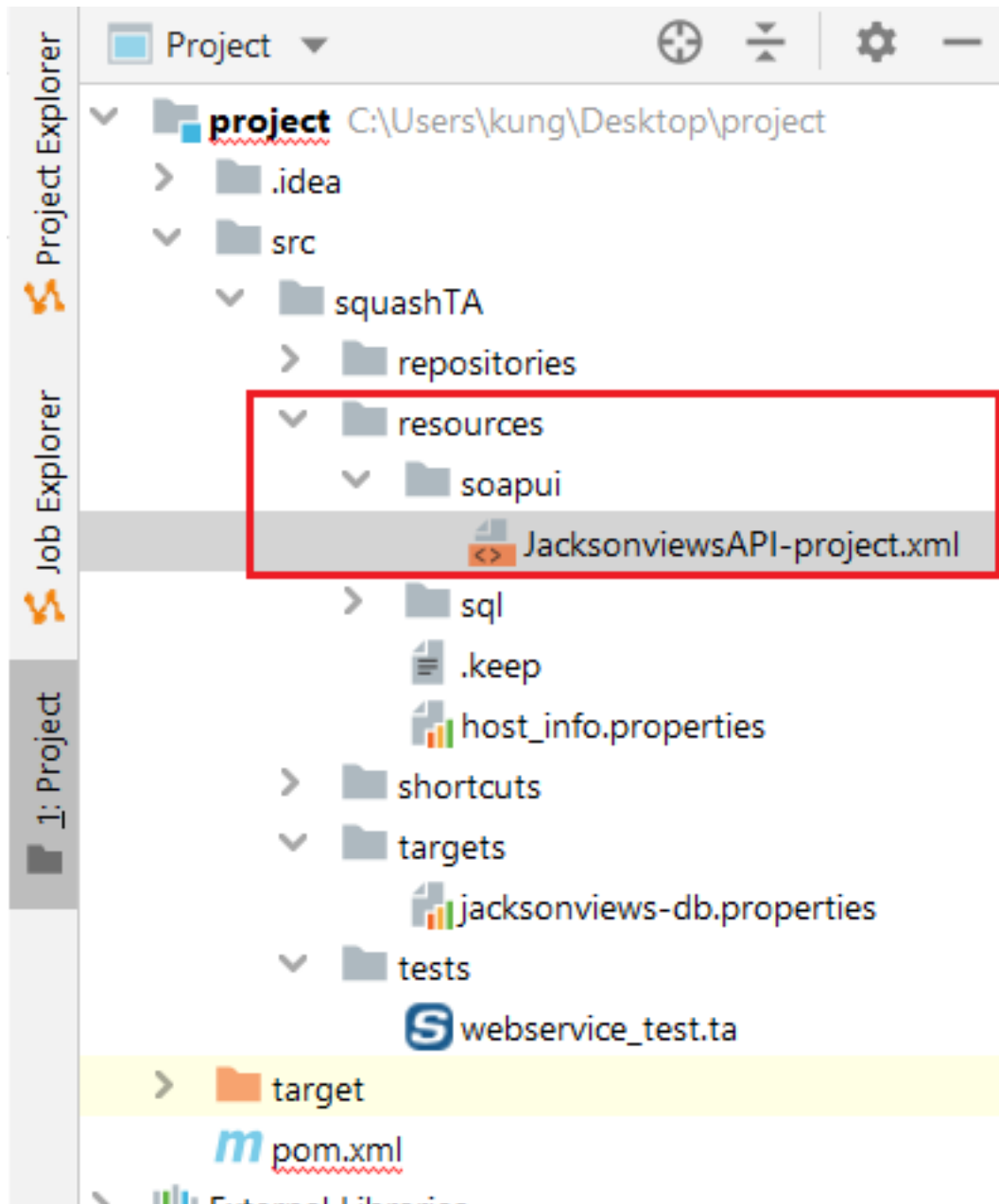


Fig. 7: Fig 7. SoapUI project location

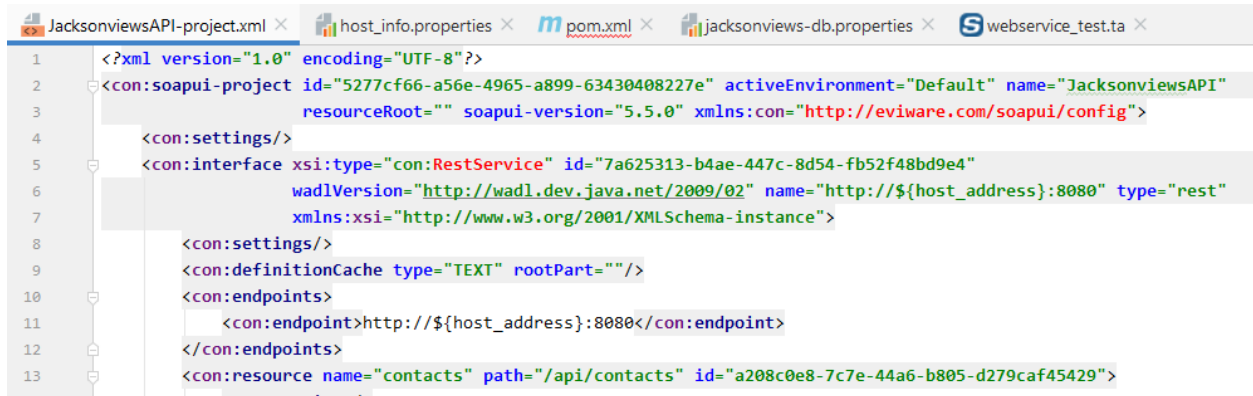


Fig. 8: Fig 8. SoapUI project with placeholders

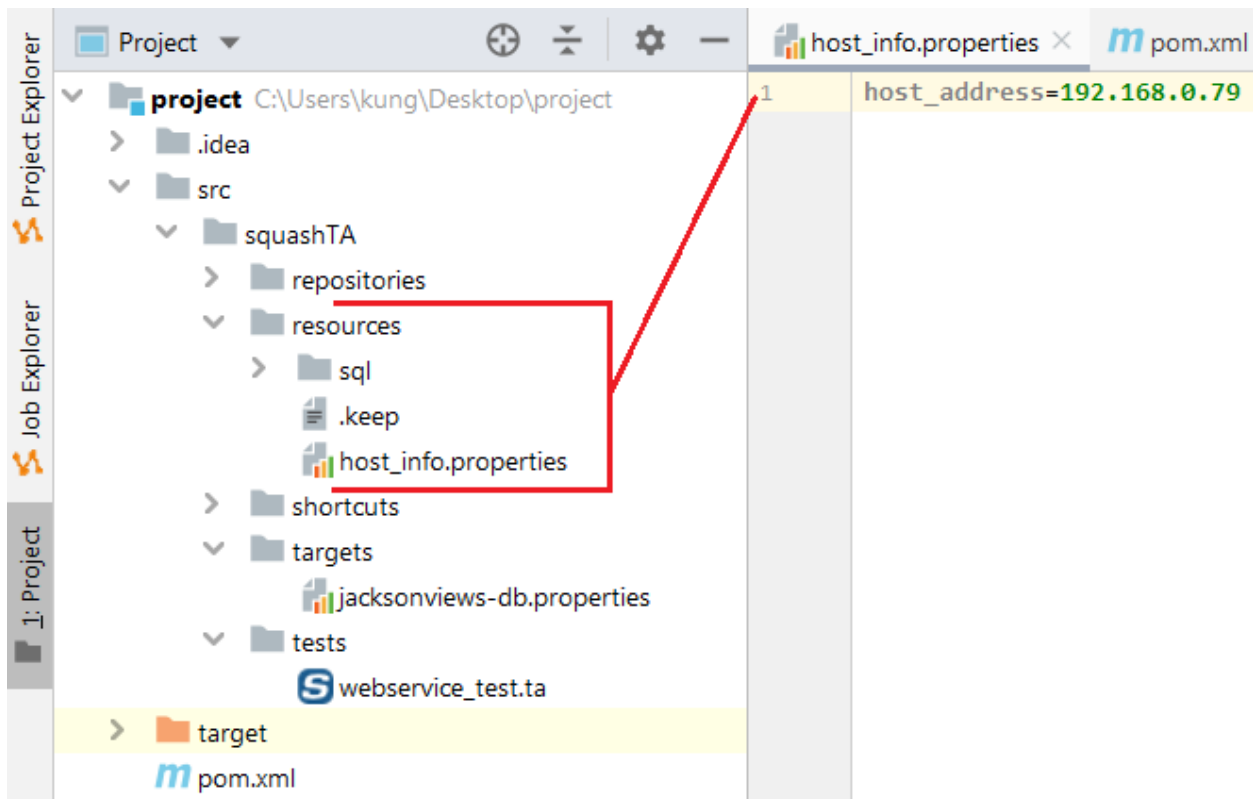


Fig. 9: Fig 9. .properties file with the values needed for the substitute keys macro

You need to change the URL in the **jacksonviews-db.properties** directly, or specify it as an option during the Maven execution with the following syntax :

```
-Dfilename_without_type.property=value
```

For example :

```
-Djacksonviews-db.squashtest.ta.database.url=jdbc:postgresql://192.168.0.178:5432/jacksonviews
```

We can now add the SUBSTITUTE KEYS macro to the **.ta** file, after the execution macro in the **SETUP phase**.

```
# SUBSTITUTE KEYS IN soapui/JacksonviewsAPI-project.xml USING host_info.properties AS modified-soap
```

- `soapui/JacksonviewsAPI-project.xml` : The SoapUI project exported in XML.
- `host_info.properties` : Property file with the key=value to use by the macro.
- `modified-soap` : Result of the instruction.

If you'd rather indicate the key=value inline, you can use the following syntax :

```
# SUBSTITUTE KEYS IN soapui/JacksonviewsAPI-project.xml USING $(host_address=192.168.0.79) AS modified-soap
```

In the case of multiple key=value, we need to add an **"\n"** to indicate each newline.

For example :

```
$(host_address=192.168.0.79\n my_second_key=my_second_value)
```

You should now have the following script :

```
webservice_test.ta
1 SETUP :
2
3 # EXECUTE_SQL sql/add_contact.sql ON jacksonviews-db AS query_result
4
5 # SUBSTITUTE KEYS IN soapui/JacksonviewsAPI-project.xml USING host_info.properties AS modified-soap
6
```

1.2.9 Test the webservice

"Our web service testing tool will be SoapUI."

The **SETUP phase** is finished, we can now begin the **TEST phase**.

We can execute our SoapUI project with the following macro :

```
# EXECUTE_SOAPUI modified-soap
```

- `modified-soap` : The result from the `SUBSTITUTE KEYS` macro with the right URL to connect to the API.

The SoapUI project use the `getContact` API to check that all the informations added through the SQL query are available.

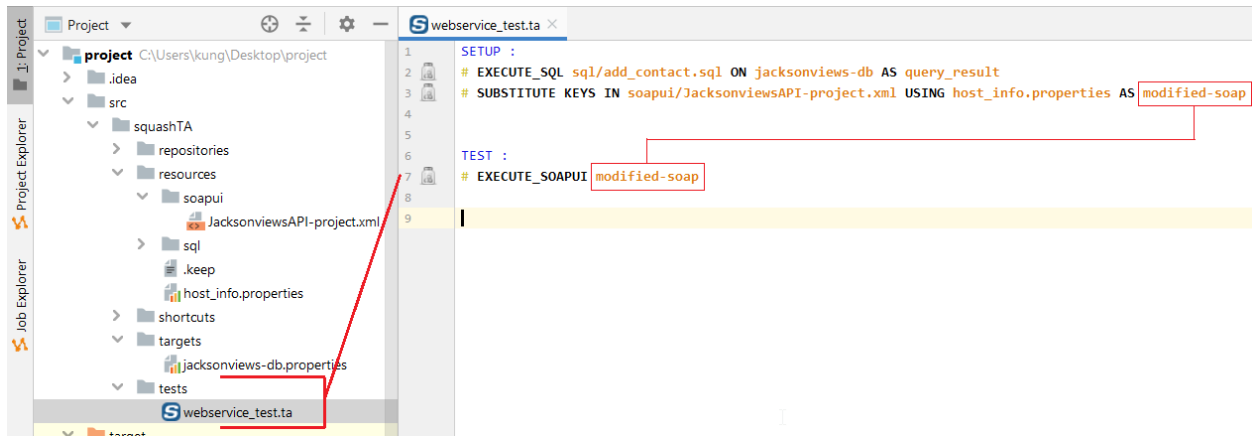


Fig. 10: Fig 10. Test phase with an `EXECUTE_SOAPUI`

For more information, check the following [page](#).

1.2.10 Clean the database

“To end our test, we should clean the database.”

We’re now trying to clean our past operations. That’s the **TEARDOWN** phase.

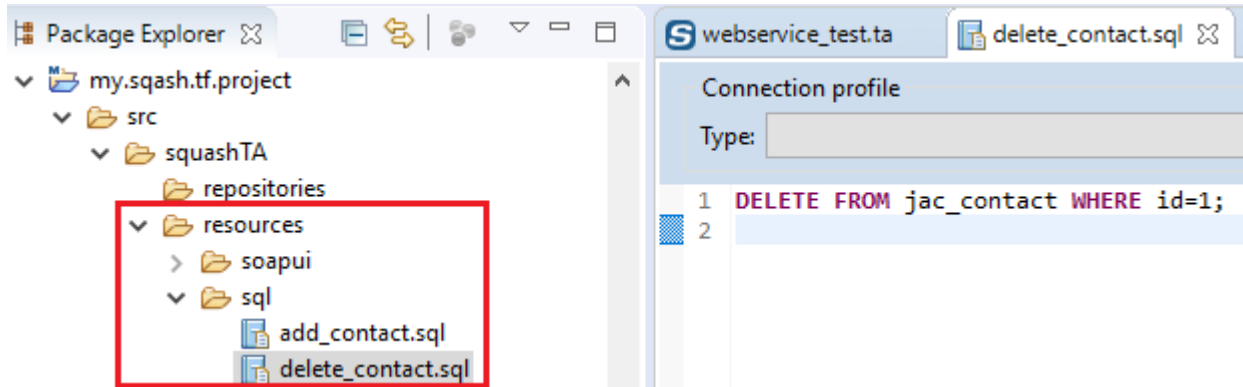
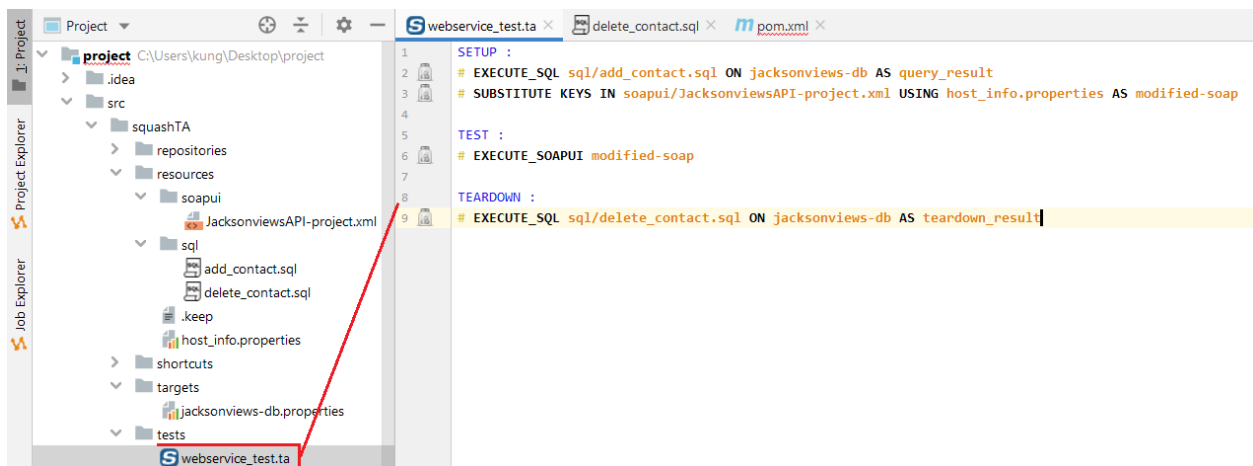
The target property is already created so we just need to create a `.sql` file.

In our example, we’ll delete the row we created in the table `jac_contact` of our database during the **SETUP** phase by adding the following line to `delete_contact.sql` :

```
DELETE FROM jac_contact WHERE id=1;
```

We can then call the macro, just like in the **SETUP** phase :

```
# EXECUTE_SQL sql/delete_contact.sql ON jacksonviews-db AS teardown_result
```

Fig. 11: Fig 11. .sql for the **TEARDOWN** phaseFig. 12: Fig 12. **TEARDOWN** phase in a .ta script

1.2.11 Execution

The test is now finished.

We can start the test by typing the following command in a shell window opened at the base of your project (where the pom.xml is located) :

`mvn squash-ta:run`

We should have a build success.

```
[INFO] Beginning execution of test webservice_test.ta
[WARN] No SLF4J binding, falling back to sysout logger for class org.squashtest.ta.plugin.commons.library.java.LoggingFixer
[WARN] No SLF4J binding, falling back to sysout logger for class org.squashtest.ta.plugin.soapui.library.SoapUiProcessExecutor
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] All the files from C:\Users\kung\AppData\Local\Temp\Squash_TA\20190827_121452_1424069241108427966872 were properly deleted.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.794 s
[INFO] Finished at: 2019-08-27T12:15:00+02:00
[INFO] Final Memory: 23M/435M
[INFO] -----
```

1.2.12 Reporting

The report will then be generated in the target (and not targets) folder, more specifically in **target/squashTA/html-reports/squash-ta-report.html**.

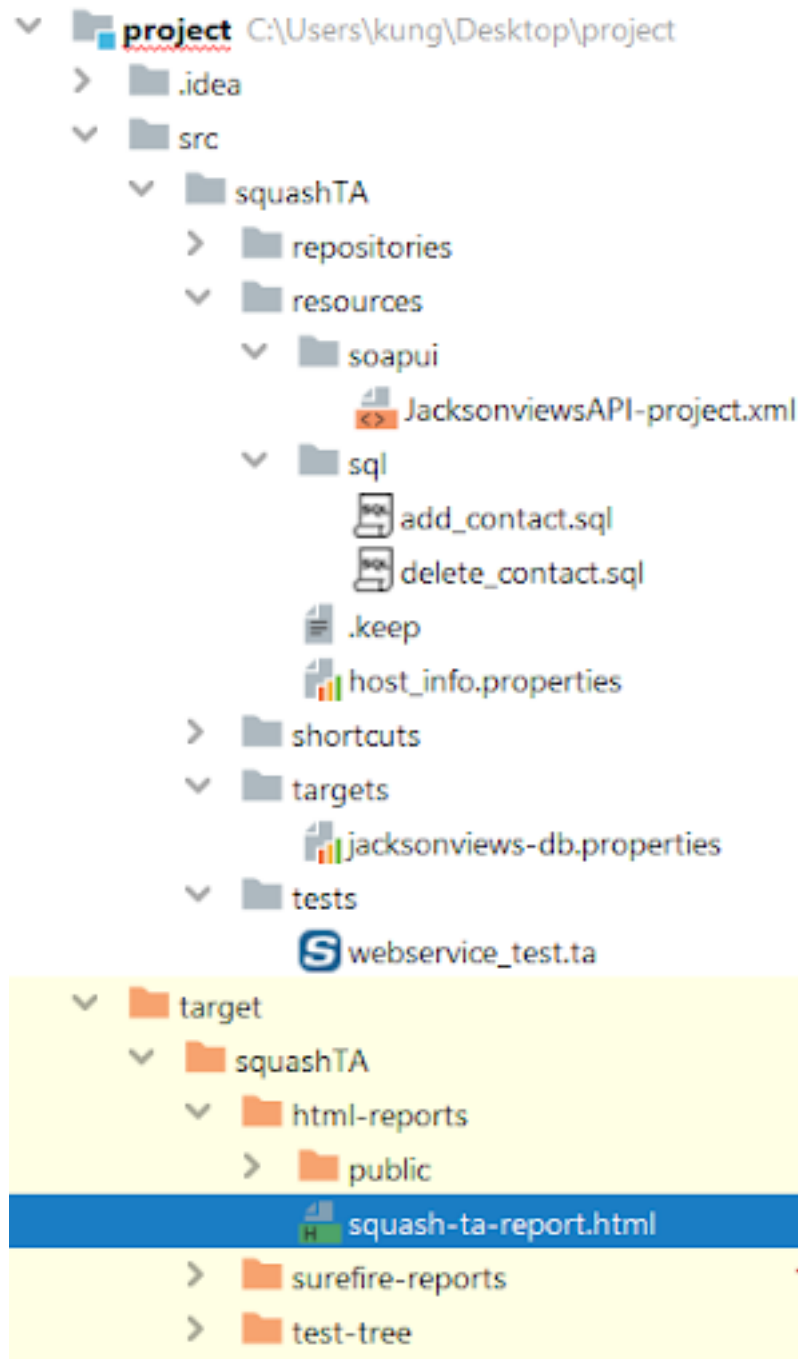


Fig. 13: Fig 13. Reports location

tests		
Script	Status	Time (s)
● webservice_test.ta	Success	6,271
[Back to top]		
● webservice_test.ta		
Status: Success Time (s): 6,271		
Test steps		Line
SETUP :		
● # EXECUTE_SQL sql/add_contact.sql ON jacksonviews-db AS query_result		2
● # SUBSTITUTE KEYS IN soapui/JacksonviewsAPI-project.xml USING host_info.properties AS modified-soap		3
TEST :		
● # EXECUTE_SOAPUI modified-soap		6
TEARDOWN :		
● # EXECUTE_SQL sql/delete_contact.sql ON jacksonviews-db AS teardown_result		9
[Back to top] [Back to tests]		

Fig. 14: Fig 14. HTML Report

Introduction to Squash Keyword Framework (SKF)

Contents :

- *Automated Project Structure*
- *Test Case*
- *Macros*

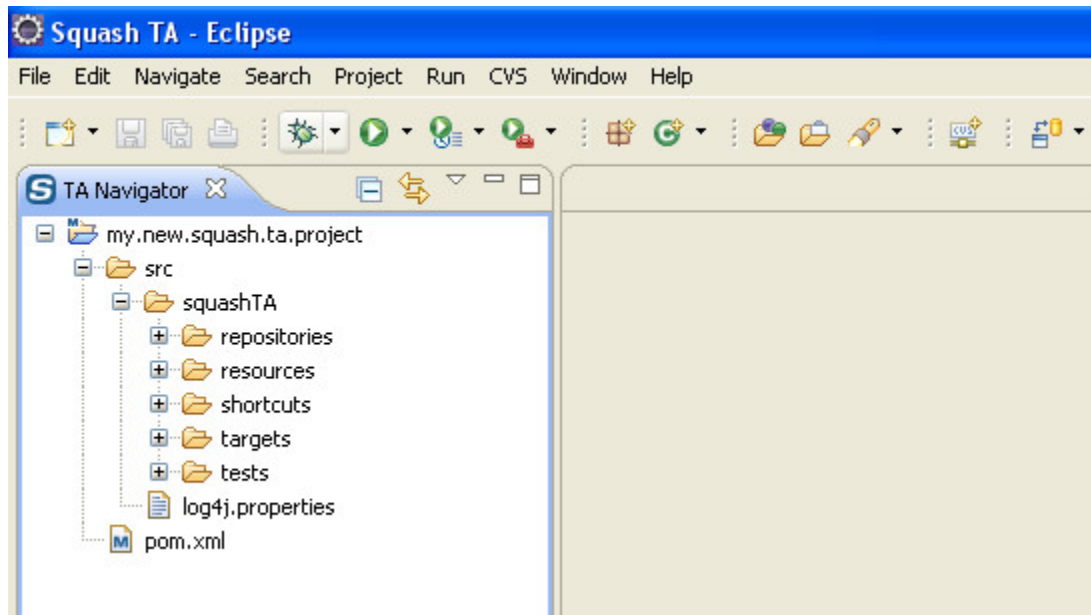
In order to create and maintain your SKF test cases you need several tools. To simplify the installation process, we have bundled these tools into **Squash TA Toolbox**. The installation of **Squash TA Toolbox** is described [here](#).

In order to *execute* your test cases you need **Squash Keyword Framework**, which is NOT part of the toolbox. No need to install it though as it will be automatically downloaded and installed the first time you will try to run an SKF test (**Squash Keyword Framework** is integrated as a maven plugin, as we will see later on).

2.1 Automated Project Structure

An SKF automated project is a **Maven Project** and must have the following structure :

The root of the **Squash TF** files is the `src/squashTA` directory. It is subdivided in 5 subdirectories :



- The `tests` directory contains SKF test scripts (Files `‘.ta’`).
- The `targets` directory contains configuration files to define the tested systems (Files `‘.properties’`).
- The `shortcuts` directory is used to define macros (Files `‘.macro’`).
- The `resources` directory contains all resources used by test scripts (test data, third party tools, configuration files, ...).
- The `repositories` directory contains definitions of the resources library of the automated project (Files `‘.properties’`).

`‘pom.xml’` (Project Object Model) is the configuration file of a Maven Project.

2.2 Test Case

In an SKF automated project all test cases must be in the `tests` directory of the project (or in a subdirectory of this directory).

A test case is described within a file named SKF script. The extension of the file to use is `‘.ta’`. An SKF script represents a test case.

The script names `‘setup.ta’` and `‘teardown.ta’` are reserved names for the ecosystem setup and teardown phases (see [here](#)).

A test case can contain 3 phases :

- A **setup phase (optional)** : Used to initiate the test case and to be sure that all necessary conditions to execute the test are gathered.
- A **test phase (mandatory)** : Contains the different test steps of the test case.

- A **teardown phase (optional)** : Generally used to clean-up the actions of the test case and so to prepare the environment for the next test case.

2.3 Macros

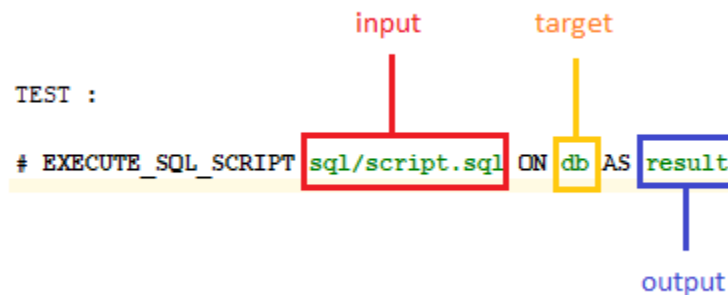
Each phase of a test case is comprised of discreet instruction lines.

Most often these instructions are written as **macros** that contain a sequence of instructions functionally linked.

Macro files all have the extension **‘.macro’**. Some are provided with the SKF but you can always write your own custom macros as described [here](#). User created macros are to be placed in the `shortcuts` directory or its subdirectories.

Macros are parametrized for each use by the user with inputs, and depending on the instructions, an output and other resources needed to execute their instructions set.

For example, the following macro is used to execute an SQL query (written on a **‘.sql’** file in the `resources` directory) on a target database **db**, with the resource **result** defined as the output of the process.



In this example the black parts of the macro are fixed and the green parts are parameters filled by the user :

- The red parameter is an **input**. It is the relative path (to the `‘resources’` folder) to the **‘.sql’** file which will be executed by the macro.
- The yellow parameter is also an **input**. This resource represents a **database** which will be the target of the SQL script.
- The blue parameter is the **output**. The resource will contain the result of the **SQL query** written in the SQL script.

Create a new SKF automation Project

As SKF is a maven plugin, the best way to create a new SKF automation project is to use the SKF project archetype. When you do so, Maven creates a new SKF automation project for you. Later on, when you first run your tests, Maven will download the SKF execution engine and its dependencies.

3.1 Create a Squash TF Project with IntelliJ

Contents :

- *Pre-requisites*
- *Creating your Squash TF project*

3.1.1 Pre-requisites

If you are using a Maven > 3.0.0, you'll need to add the following section to your **settings.xml** (located in the **conf** directory of your Maven folder, or in **\plugins\maven\lib\maven3\conf** in your IntelliJ directory for the bundled version) :

```
<profiles>
  <profile>
    <repositories>
      <repository>
```

(continues on next page)

(continued from previous page)

```

        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
        <id>archetype</id>
        <name>squashTA</name>
        <url>http://repo.squashtest.org/maven2/releases/</url>
    </repository>
</repositories>
<pluginRepositories>
    <pluginRepository>
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
        <id>squashTA</id>
        <name>squashTA</name>
        <url>http://repo.squashtest.org/maven2/releases/</url>
    </pluginRepository>
</pluginRepositories>
<id>squashTA</id>
</profile>
</profiles>
<activeProfiles>
    <activeProfile>squashTA</activeProfile>
</activeProfiles>

```

Add the following to your file :

```

<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.
→apache.org/xsd/settings-1.0.0.xsd">

    <profiles>
        <profile>
            <repositories>
                <repository>
                    <snapshots>
                        <enabled>>false</enabled>
                    </snapshots>
                    <id>archetype</id>
                    <name>squashTA</name>
                    <url>http://repo.squashtest.org/maven2/releases/</url>
                </repository>
            </repositories>
            <pluginRepositories>
                <pluginRepository>
                    <snapshots>
                        <enabled>>false</enabled>
                    </snapshots>
                    <id>squashTA</id>
                    <name>squashTA</name>
                    <url>http://repo.squashtest.org/maven2/releases/</url>
                </pluginRepository>
            </pluginRepositories>
            <id>squashTA</id>
        </profile>
    </profiles>

```

(continues on next page)

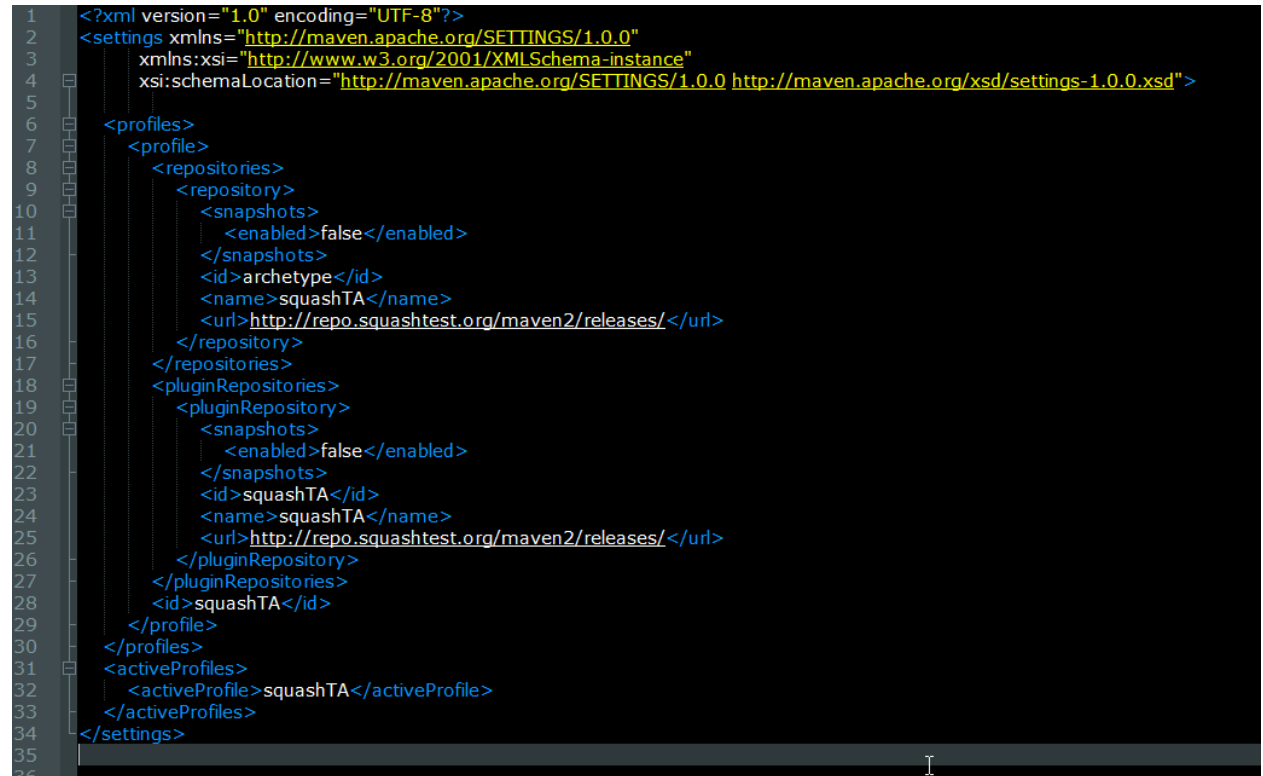
```
6 <profiles>
7   <profile>
8     <repositories>
9       <repository>
10        <snapshots>
11          <enabled>>false</enabled>
12        </snapshots>
13        <id>archetype</id>
14        <name>squashTA</name>
15        <url>http://repo.squashtest.org/maven2/releases/</url>
16      </repository>
17    </repositories>
18    <pluginRepositories>
19      <pluginRepository>
20        <snapshots>
21          <enabled>>false</enabled>
22        </snapshots>
23        <id>squashTA</id>
24        <name>squashTA</name>
25        <url>http://repo.squashtest.org/maven2/releases/</url>
26      </pluginRepository>
27    </pluginRepositories>
28    <id>squashTA</id>
29  </profile>
30 </profiles>
31 <activeProfiles>
32   <activeProfile>squashTA</activeProfile>
33 </activeProfiles>
34 </settings>
35
```

(continued from previous page)

```

</profiles>
<activeProfiles>
  <activeProfile>squashTA</activeProfile>
</activeProfiles>
</settings>

```



```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
5
6    <profiles>
7      <profile>
8        <repositories>
9          <repository>
10             <snapshots>
11               <enabled>false</enabled>
12             </snapshots>
13             <id>archetype</id>
14             <name>squashTA</name>
15             <url>http://repo.squashtest.org/maven2/releases/</url>
16           </repository>
17         </repositories>
18         <pluginRepositories>
19           <pluginRepository>
20             <snapshots>
21               <enabled>false</enabled>
22             </snapshots>
23             <id>squashTA</id>
24             <name>squashTA</name>
25             <url>http://repo.squashtest.org/maven2/releases/</url>
26           </pluginRepository>
27         </pluginRepositories>
28         <id>squashTA</id>
29       </profile>
30     </profiles>
31     <activeProfiles>
32       <activeProfile>squashTA</activeProfile>
33     </activeProfiles>
34   </settings>
35
36

```

3.1.2 Creating your Squash TF project

Launch IntelliJ IDEA then click on “Create New Project” :

Select in the left column the Maven type then check the “Create from archetype” box.

Click on “Add Archetype...” to add the one needed.

Insert the following informations :

- **GroupId** : org.squashtest.ta
- **ArtifactId** : squash-ta-project-archetype
- **Version** : You can check the last version of the Squash Keyword Framework on our [website](#)
- **Repository** : <http://repo.squashtest.org/maven2/releases> (Only if your Maven is < 3.0.0)



IntelliJ IDEA

Version 2019.2

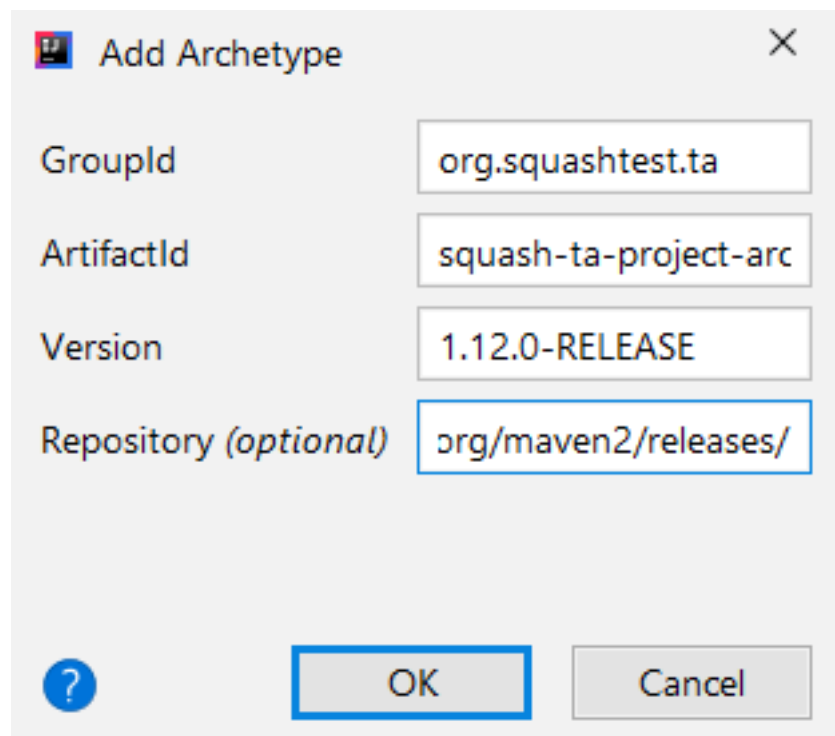
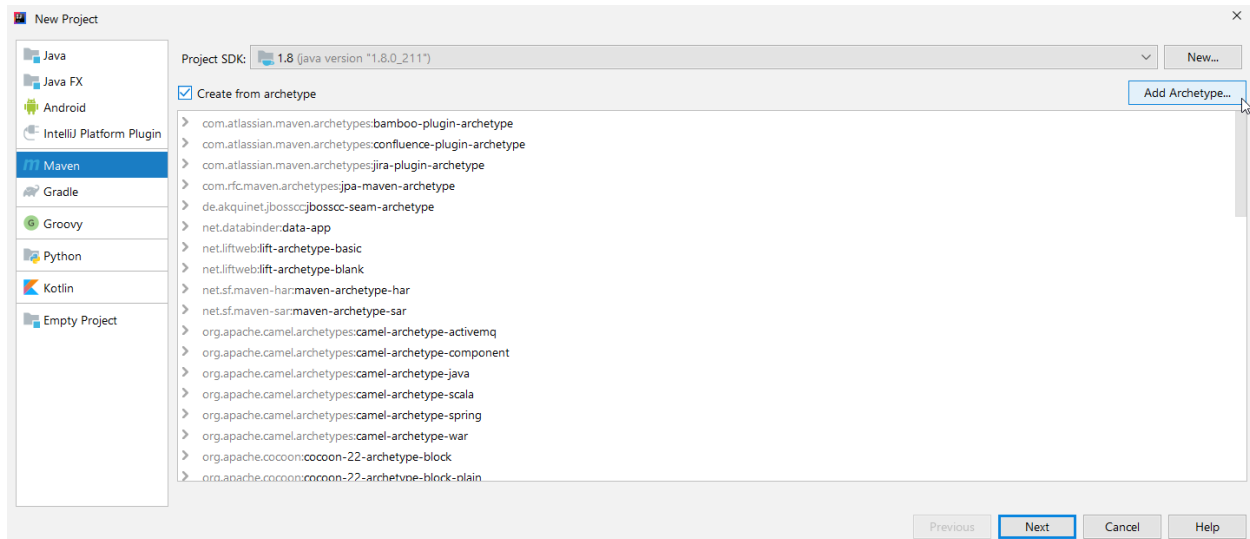
+ Create New Project

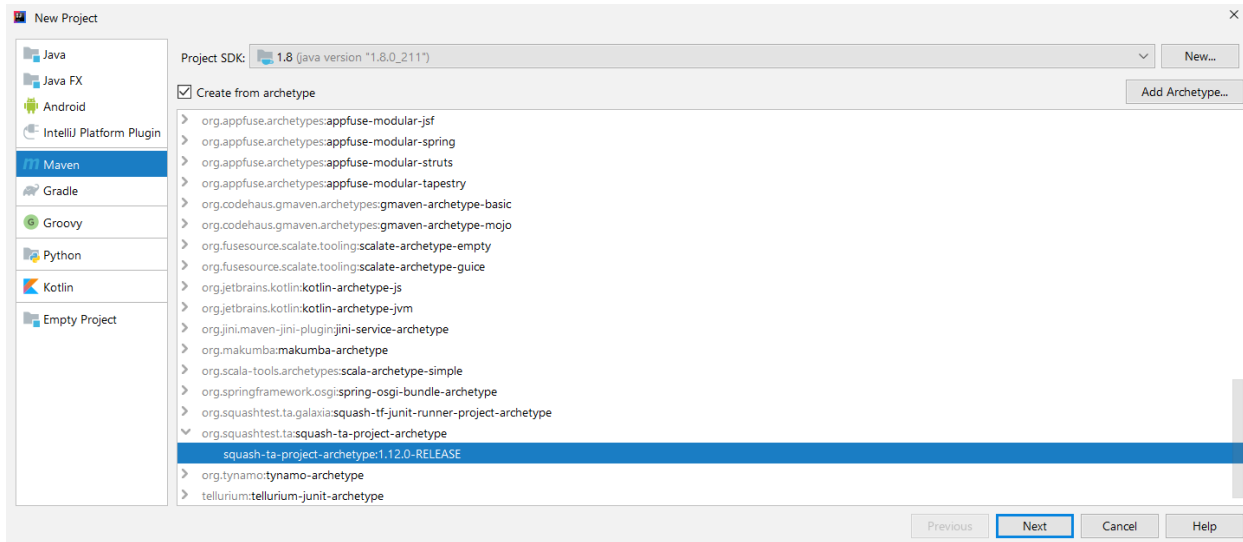
📁 Import Project

📁 Open

📁 Check out from Version Control ▼

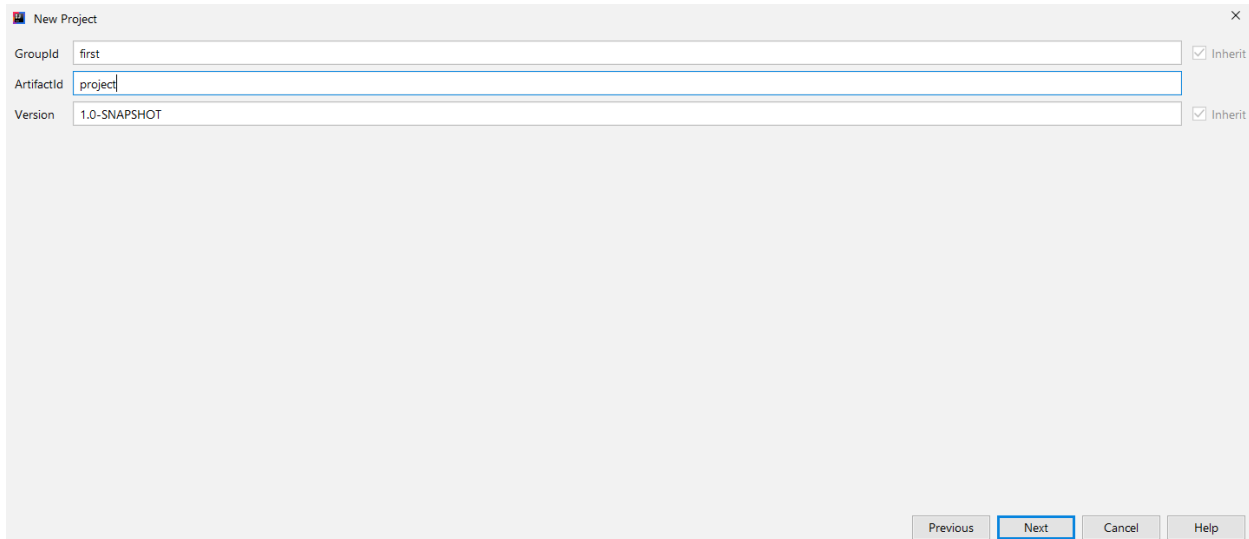
⚙️ Configure ▼ Get Help ▼





Select the newly created archetype and click on “Next”.

Insert your **groupId**, **ArtifactId**, **Version**, and click on “Next”.



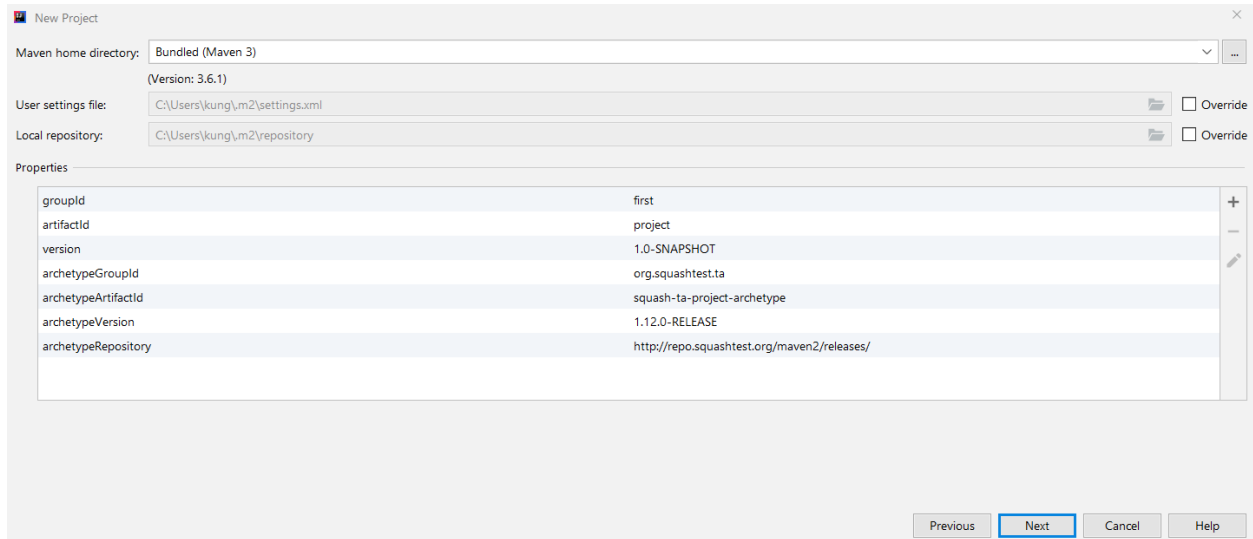
Select your Maven. It should be the one with the repository in the **settings.xml**. Click on “Next”.

Select a project name and location, and click on “Finish”.

You may need to wait a little bit.

You should have a build success and the following structure :

3.2 Create a Squash TF Project with Squash TA Toolbox



New Project

Maven home directory: Bundled (Maven 3) (Version: 3.6.1)

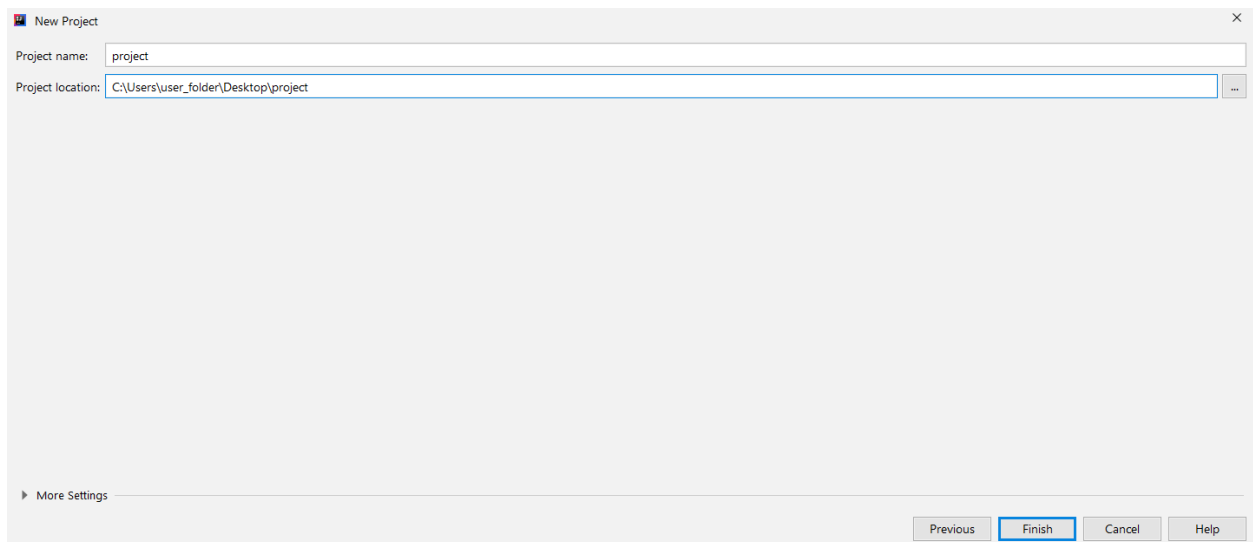
User settings file: C:\Users\kung\m2\settings.xml ☐ Override

Local repository: C:\Users\kung\m2\repository ☐ Override

Properties

groupId	first
artifactId	project
version	1.0-SNAPSHOT
archetypeGroupId	org.squashtest.ta
archetypeArtifactId	squash-ta-project-archetype
archetypeVersion	1.12.0-RELEASE
archetypeRepository	http://repo.squashtest.org/maven2/releases/

Previous Next Cancel Help



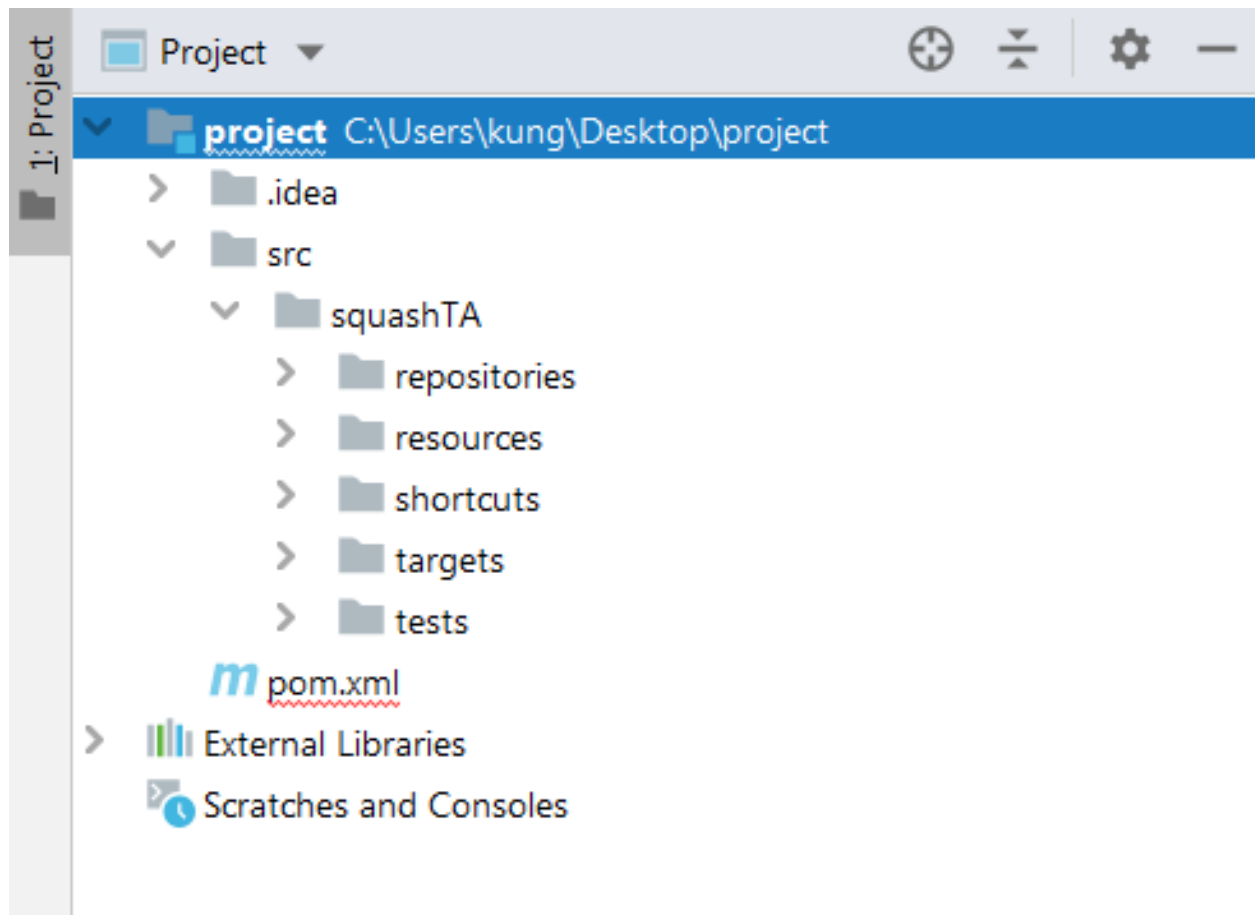
New Project

Project name: project

Project location: C:\Users\user_folder\Desktop\project

More Settings

Previous Finish Cancel Help



Contents :

- *Pre-requisites*
- *Creating your Squash TF project*

3.2.1 Pre-requisites

You need to have the **Squash TA toolbox** installed on your workstation. The toolbox is packaged with Eclipse and the m2eclipse plugin.

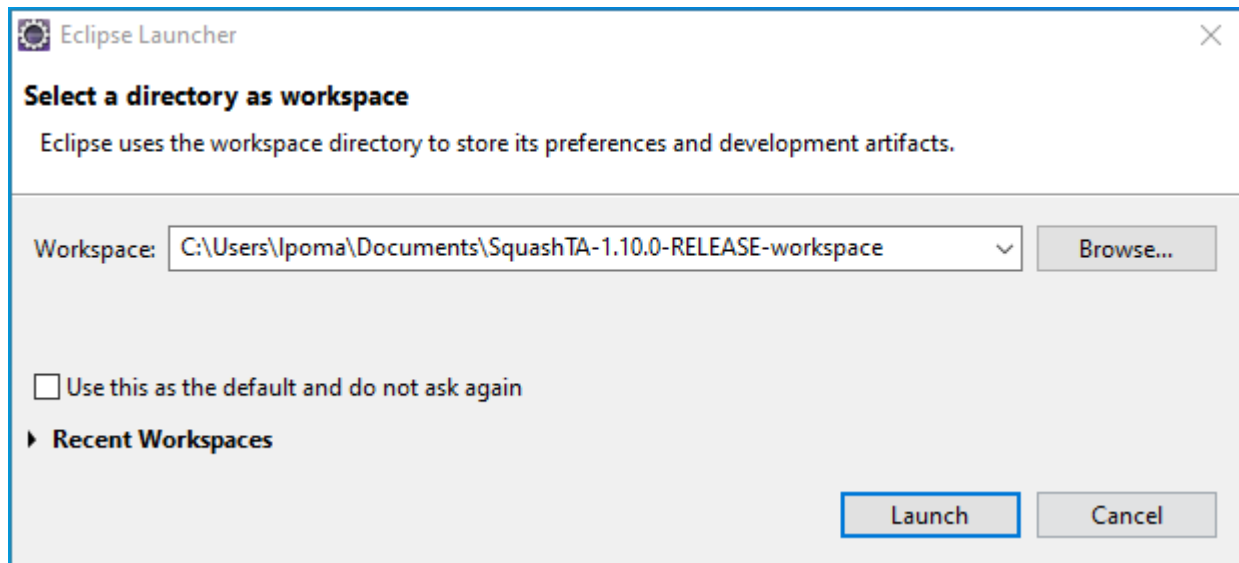
You can download and install it, as described [here](#).

3.2.2 Creating your Squash TF project

Let's start a dedicated Eclipse instance by clicking on the “**Squash-TA Eclipse**” icon on your desktop.

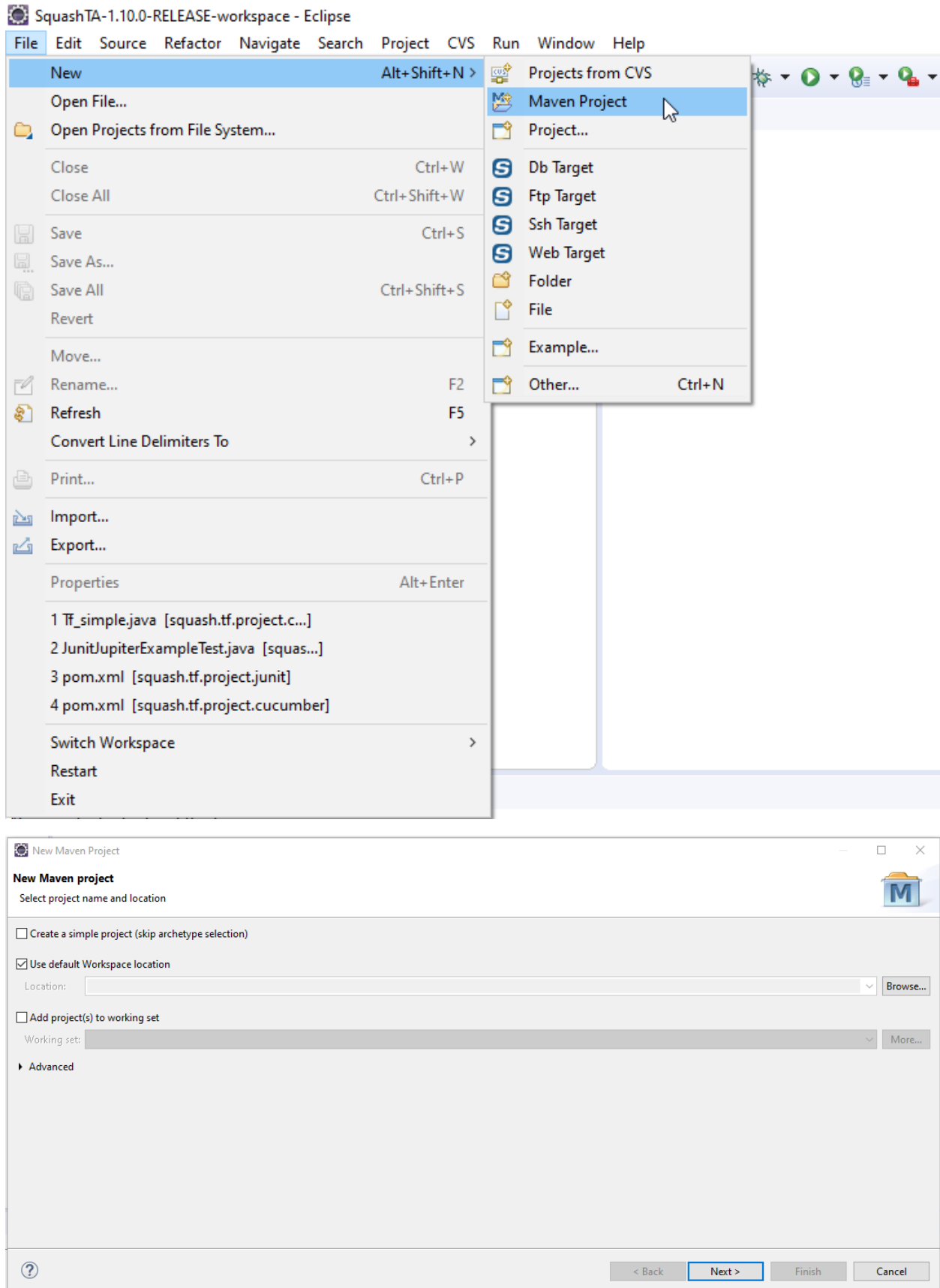
Confirm that you want to use your newly created workspace.

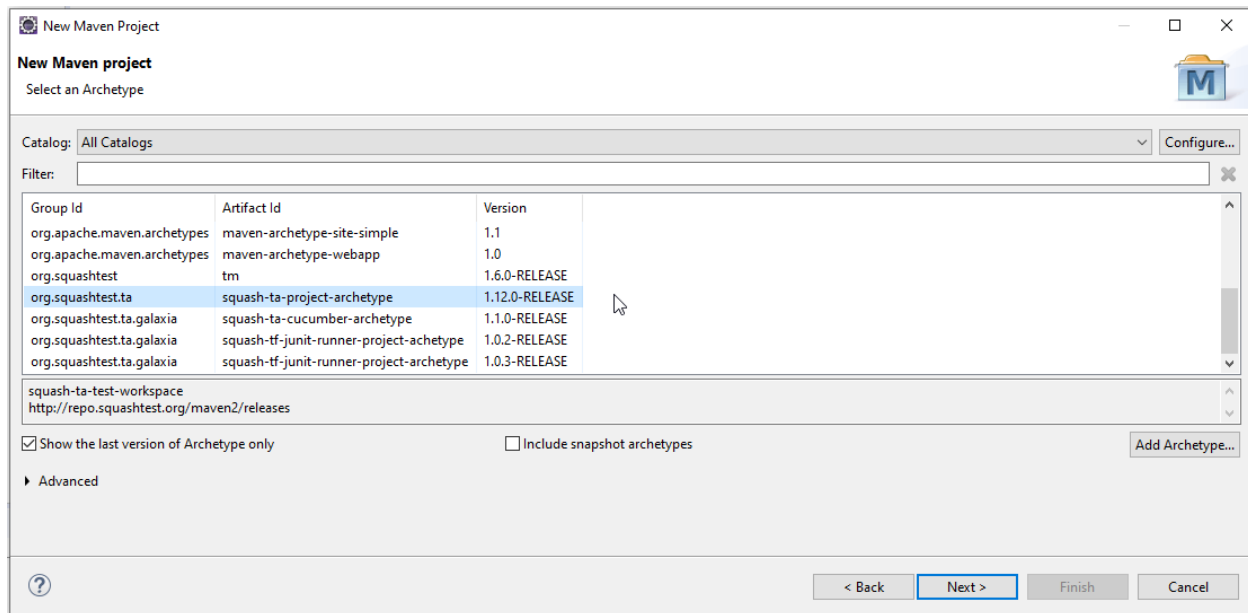
It is important to use this workspace because it contains specific archetypes for your projects and run configurations for your test scripts.



When Eclipse is started, we will create a new **Squash TF** project as follows :

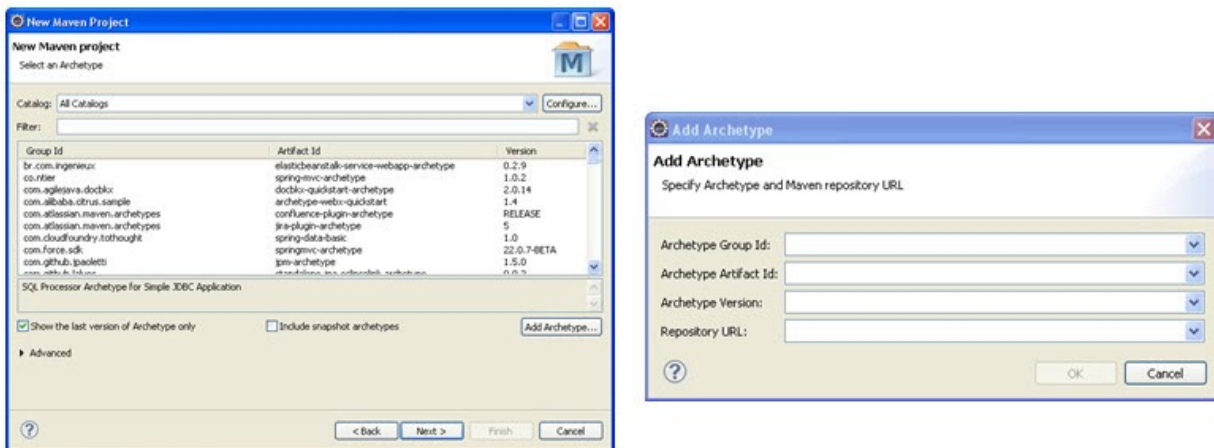
- Select **File > New > Maven Project** :
- In the “New Maven Project” dialog window that opens, click on **Next** :
- Select the Squash-TF project maven archetype with Artifact id **squash-ta-project-archetype** :





- If you can't find this archetype or wish to create your own you can instead add it from scratch as follows :

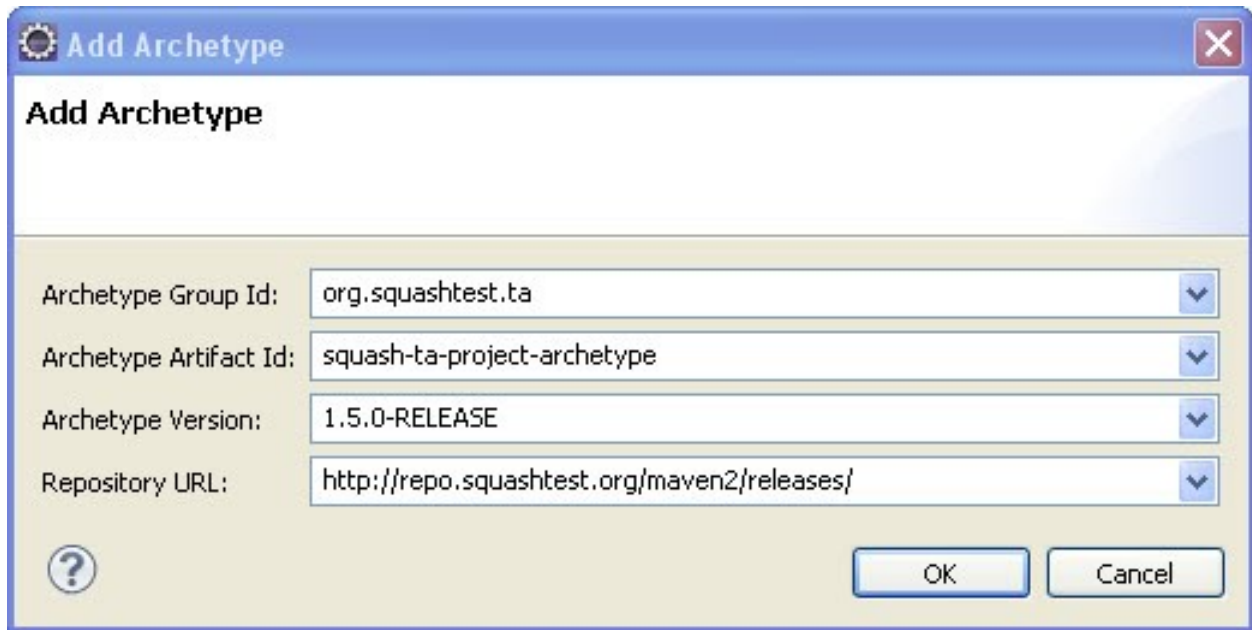
– Click on the **Add Archetype** button :



– Enter the following values :

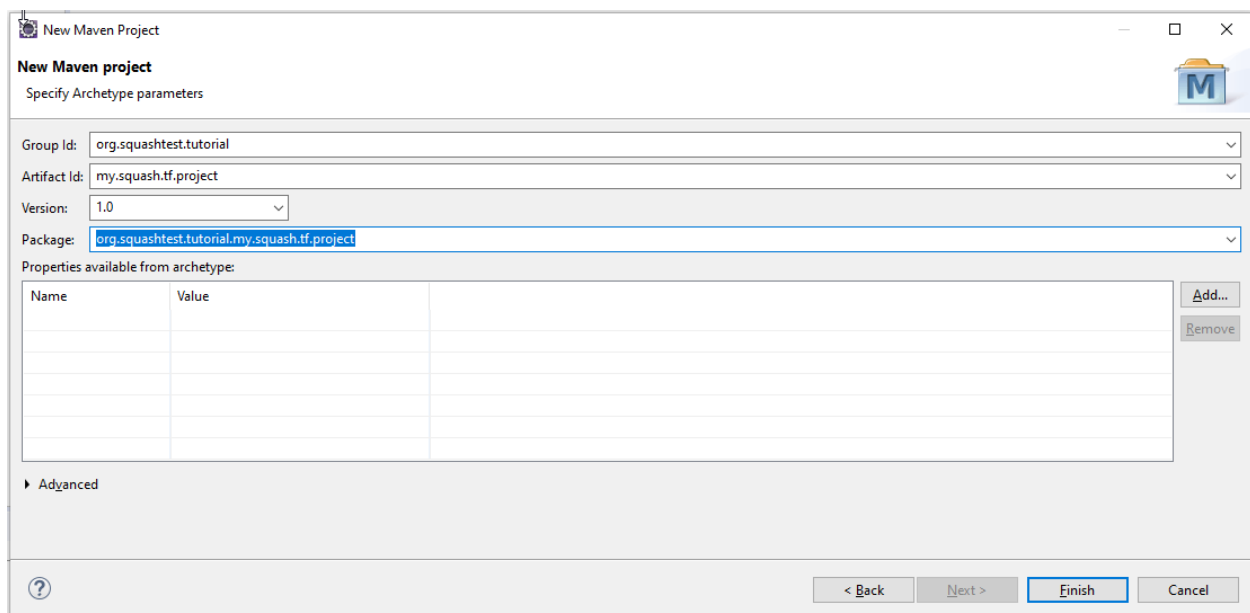
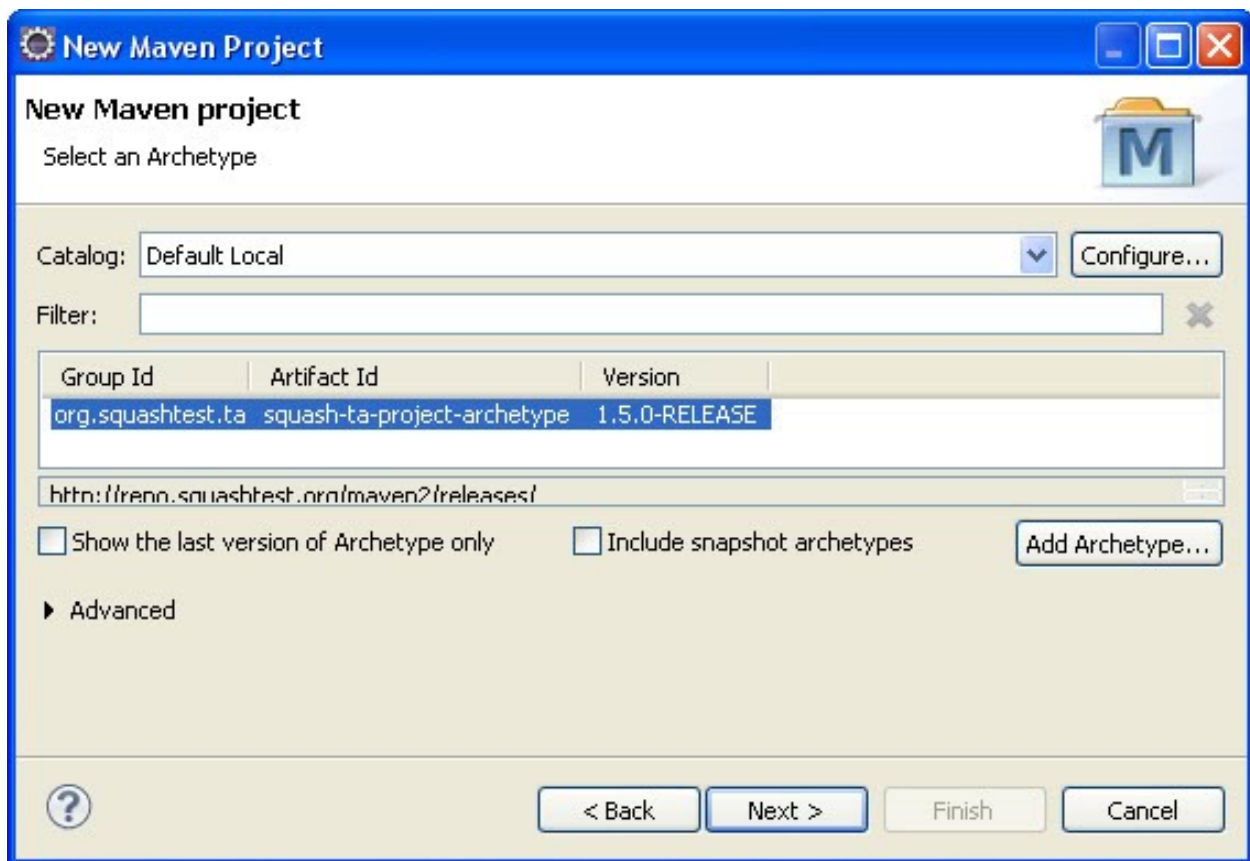
- * **Archetype Group ID** : org.squashtest.ta
- * **Archetype Artifact ID** : squash-ta-project-archetype
- * **Archetype Version** : You can check the last version of the Squash Keyword Framework on our [website](http://repo.squashtest.org/maven2/releases/)
- * **Repository URL** : <http://repo.squashtest.org/maven2/releases/>

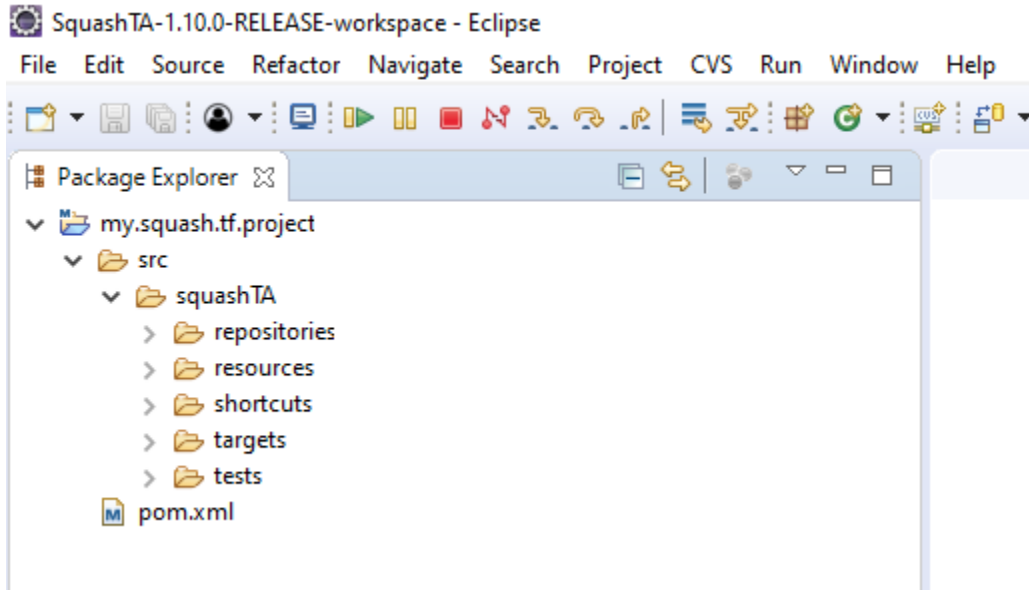
– Confirm by clicking on the **OK** button :



- Then, select the archetype (it has appeared in the archetype list) and uncheck the **Show the last version of Archetype only** option. In the Catalog list at the top of the window, choose **Default local**. Click on **Next** to go to the next page :
- On the next screen, (whether you used a provided or custom archetype), you are describing your new test project :
 - The **Group Id** is the name of the broader software package your test automation project belongs to. In our example we will use **org.squashtest.tutorial** as group Id.
 - The **Artifact Id** is the specific name of your test project. Let's use **my.squash.tf.project** as artifact id.
 - Finally, we will use the **1.0** version number, and ignore the field **package** which is not relevant for Squash-TF projects.
- Now just launch project creation by clicking on the **Finish** button :
- The newly created test project appears in the explorer :

Note: If you encounter some difficulties to create your new TF project through the **Squash TF** project archetype in Eclipse, please try the command-line method.





3.3 Create a Squash TF Project using a command line

You can create the **Squash TF** project with a maven command-line, then import it into your preferred IDE.

Open a shell window at the location where you want your project created and execute the following command line :

```
mvn archetype:generate -DarchetypeGroupId=org.squashtest.ta -
↪DarchetypeArtifactId=squash-ta-project-archetype -DarchetypeVersion={SKF version} -
↪DarchetypeRepository=http://repo.squashtest.org/maven2/releases
```

Note : Do not forget to replace the **{SKF version}** with the last version of the Squash Keyword Framework. You can check it on our [website](#)

- At the prompt, enter the desired **groupId** (in our example, **org.squashtest.tutorial**).
- Next, enter the desired **artifactId** (here, **my.squash.ta.project**).
- Next, enter the **version** (here, **1.0**).
- Skip the next, irrelevant, prompt about java packages and you can check the parameters and confirm them with **y** if it's OK.

After the execution of the archetype, you should have the following :

Now, you can close the shell window and import the project into your IDE (Eclipse in the following example) :

- Select menu **File > Import...**, then **Maven > Existing Maven Projects**. Click on **Browse** and go to the newly created project directory (in our example **C:\Workspace\my.squash.tf.project**) :
- Click on **OK**, then **Finish** to complete the project import operation :

```

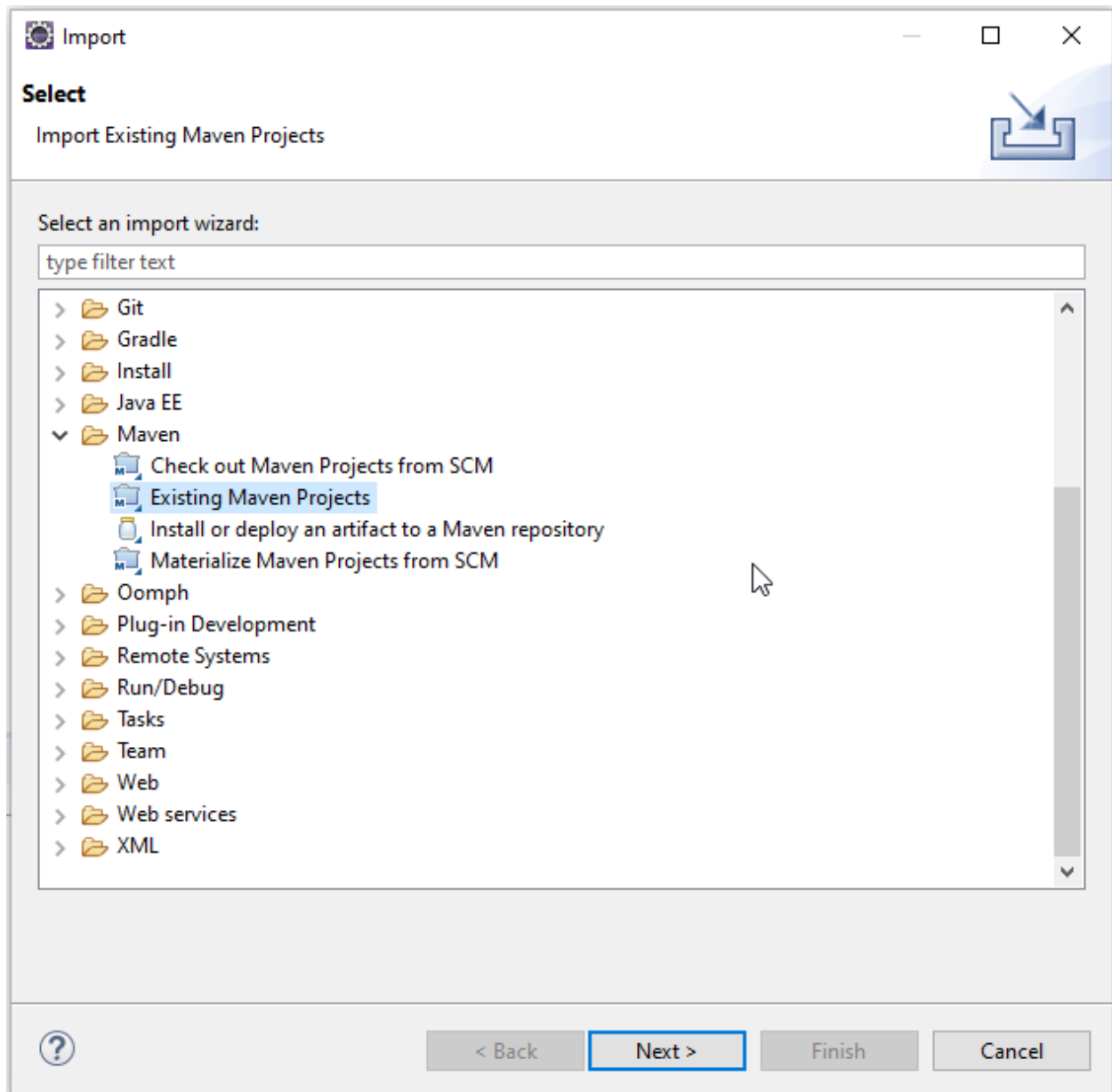
C:\Windows\System32\cmd.exe - mvn archetype:generate -DarchetypeGroupId=org.squashtest.ta -DarchetypeArtifactId=squash-ta-project-archetype ...
09:51:09,762 - [INFO] Scanning for projects...
Downloading: http://repo.squashtest.org/maven2/releases/org/codehaus/mojo/maven-metadata.xml
Downloading: http://repo.squashtest.org/maven2/releases/org/apache/maven/plugins/maven-metadata.xml
Downloaded: http://repo.squashtest.org/maven2/releases/org/codehaus/mojo/maven-metadata.xml (281 B at 2.2 kB/s)
Downloading: http://repo.squashtest.org/maven2/releases/org/apache/maven/plugins/maven-archetype-plugin/maven-metadata.xml
09:51:11,865 - [INFO] -----
09:51:11,865 - [INFO] Building Maven Stub Project (No POM) 1
09:51:11,865 - [INFO] -----
09:51:11,865 - [INFO] >>> maven-archetype-plugin:3.1.1:generate (default-cli) > generate-sources @ standalone-pom >>>
09:51:11,881 - [INFO] <<< maven-archetype-plugin:3.1.1:generate (default-cli) < generate-sources @ standalone-pom <<<
09:51:11,881 - [INFO] --- maven-archetype-plugin:3.1.1:generate (default-cli) @ standalone-pom ---
09:51:15,283 - [INFO] Generating project in Interactive mode
09:51:15,330 - [INFO] Archetype repository not defined. Using the one from [org.squashtest.ta:squash-ta-project-archetype:1.12.0-RELEASE] found in catalog remote
Define value for property 'groupId': org.squashtest.tutorial
Define value for property 'artifactId': my.squash.tf.project
Define value for property 'version': 1.0-SNAPSHOT: : 1.0
Define value for property 'package' org.squashtest.tutorial: :
Confirm properties configuration:
groupId: org.squashtest.tutorial
artifactId: my.squash.tf.project
version: 1.0
package: org.squashtest.tutorial
Y: : y

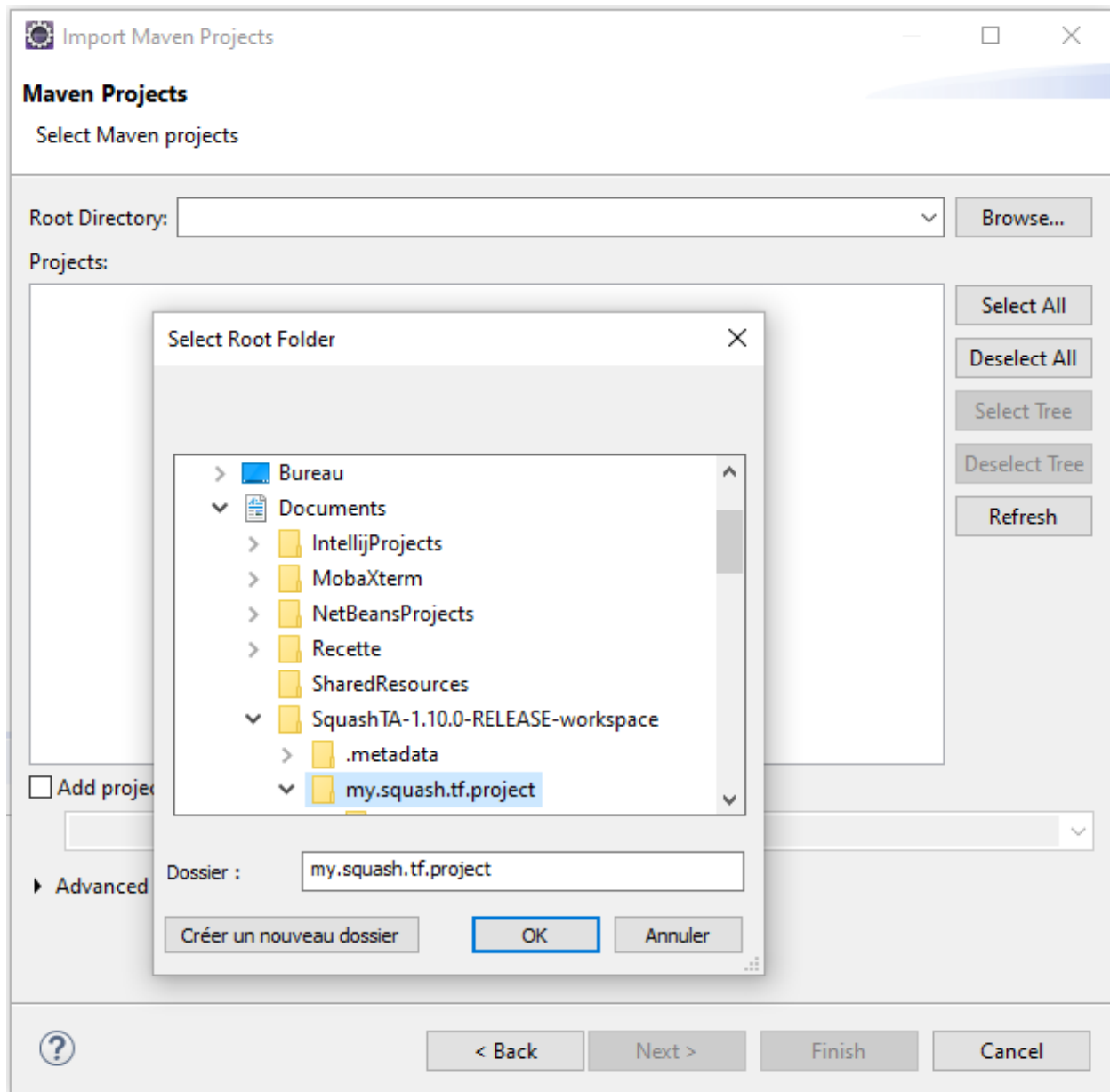
```

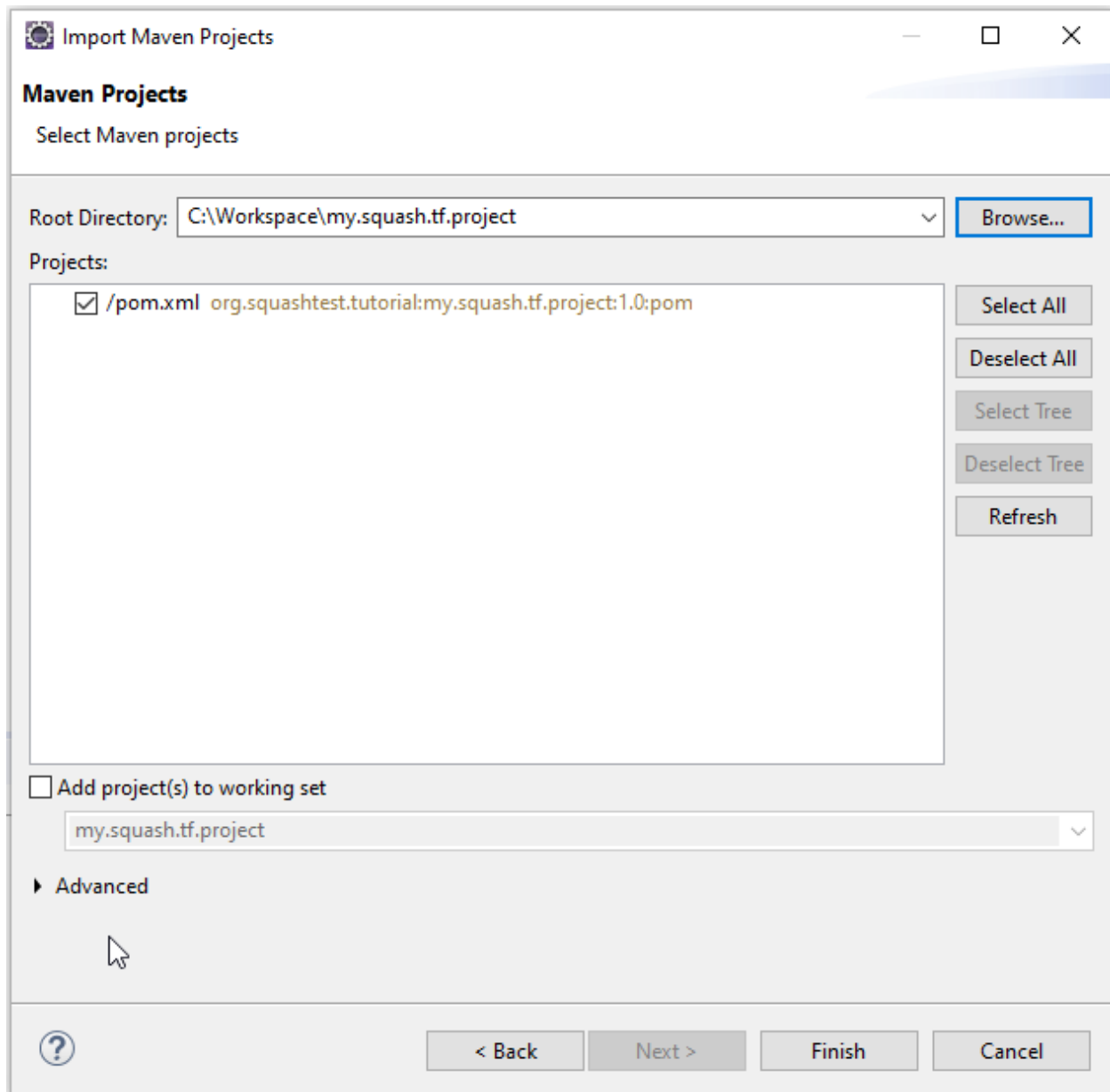
```

C:\Windows\System32\cmd.exe
Define value for property 'package' org.squashtest.tutorial: :
Confirm properties configuration:
groupId: org.squashtest.tutorial
artifactId: my.squash.tf.project
version: 1.0
package: org.squashtest.tutorial
Y: : y
09:52:37,425 - [INFO] -----
09:52:37,425 - [INFO] Using following parameters for creating project from Archetype: squash-ta-project-archetype:1.12.0-RELEASE
09:52:37,425 - [INFO] -----
09:52:37,425 - [INFO] Parameter: groupId, Value: org.squashtest.tutorial
09:52:37,425 - [INFO] Parameter: artifactId, Value: my.squash.tf.project
09:52:37,425 - [INFO] Parameter: version, Value: 1.0
09:52:37,425 - [INFO] Parameter: package, Value: org.squashtest.tutorial
09:52:37,425 - [INFO] Parameter: packageInPathFormat, Value: org/squashtest/tutorial
09:52:37,425 - [INFO] Parameter: package, Value: org.squashtest.tutorial
09:52:37,440 - [INFO] Parameter: version, Value: 1.0
09:52:37,440 - [INFO] Parameter: groupId, Value: org.squashtest.tutorial
09:52:37,440 - [INFO] Parameter: artifactId, Value: my.squash.tf.project
09:52:37,481 - [INFO] Project created from Archetype in dir: C:\Workspace\my.squash.tf.project
09:52:37,481 - [INFO] -----
09:52:37,497 - [INFO] BUILD SUCCESS
09:52:37,497 - [INFO] -----
09:52:37,497 - [INFO] Total time: 01:27 min
09:52:37,497 - [INFO] Finished at: 2019-09-16T09:52:37+02:00
09:52:37,560 - [INFO] Final Memory: 17M/97M
09:52:37,560 - [INFO] -----
C:\Workspace>

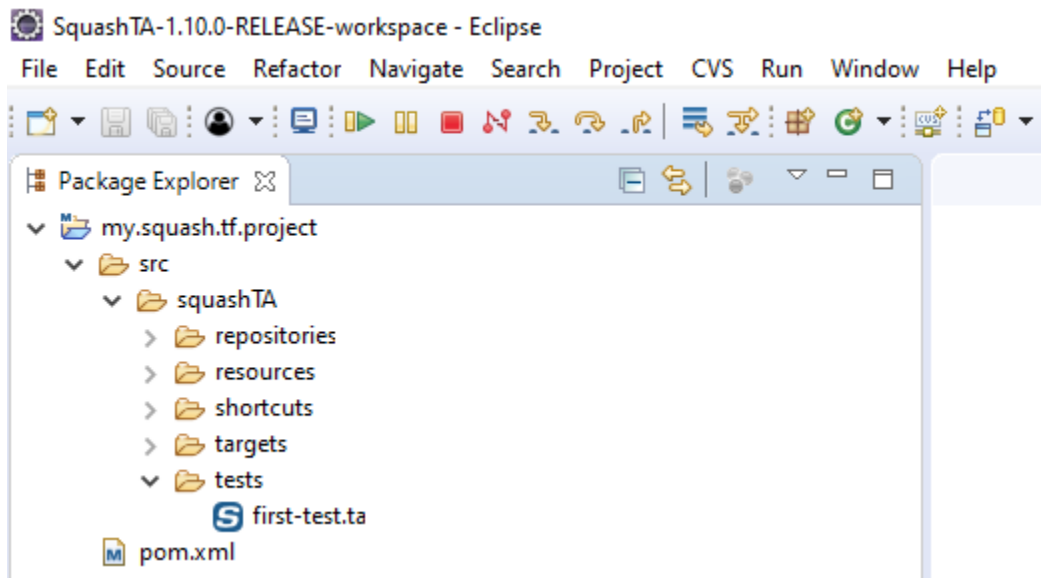
```







- The newly created Test project now appears in the TA Navigator :



3.4 Default SKF automation project pom.xml

3.4.1 Default pom

Here is an example of SKF's default pom. It's the one generated with the maven archetype (for more information, please consult the page about *creating a new project*).

```
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
↪maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance">

  <modelVersion>4.0.0</modelVersion>

  <groupId>squash-project</groupId>
  <artifactId>test</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>

  <!-- Properties definition -->
  <properties>
    <!-- Squash-TA framework version used by the project -->
    <ta.framework.version>1.12.0-RELEASE</ta.framework.version>
```

(continues on next page)

(continued from previous page)

```

</properties>

<build>
  <plugins>
    <!-- Configuration of the Squash TA framework used by the project -->
    <plugin>
      <groupId>org.squashtest.ta</groupId>
      <artifactId>squash-ta-maven-plugin</artifactId>
      <version>${ta.framework.version}</version>

      <!-- Here you can add libraries to the engine classpath, using the
      ↪<dependencies></dependencies> tag -->
      <!-- A sample with the mySql jdbc driver -->
      <!-- <dependencies> -->
      <!--   <dependency> -->
      <!--       <groupId>mysql</groupId> -->
      <!--       <artifactId>mysql-connector-java</artifactId> -->
      <!--       <version>5.1.19</version> -->
      <!--   </dependency> -->
      <!-- </dependencies> -->

      <!-- Under here is the Squash TA framework default configuration -->
      <configuration>

        <!--
        ↪Uncomment the line below in order to the build finish in_
        ↪success even if a test failed
        ↪      (functional (assertion) failure), but fail the build if an_
        ↪ERROR (technical failure) occurred.
        -->
        <!-- <mojoSuccessThreshold>FAIL</mojoSuccessThreshold> -->

        <!-- Define a log configuration file (at log4j format) to_
        ↪override the one defined internally -->
        <!-- If the given file can't be found the engine switch to the_
        ↪internal configuration-->
        <logConfiguration>${logConfFile}</logConfiguration>

        <!-- Define exporters -->
        <exporters>
          <surefire>
            <jenkinsAttachmentMode>${ta.jenkins.attachment.mode}</
            ↪jenkinsAttachmentMode>
          </surefire>
          <html/>
        </exporters>

        <!-- Define configurers -->
        <configurers>
          <tmCallBack>
            <endpointURL>${status.update.events.url}</endpointURL>
            <executionExternalId>${squash.ta.external.id}</
            ↪executionExternalId>

            <jobName>${jobname}</jobName>
            <hostName>${hostname}</hostName>
            <endpointLoginConfFile>${squash.ta.conf.file}</
            ↪endpointLoginConfFile>

```

(continues on next page)

(continued from previous page)

```

        <reportBaseUrl>${ta.tmcallback.reportbaseurl}</
↪reportBaseUrl>
        <jobExecutionId>${ta.tmcallback.jobexecutionid}</
↪jobExecutionId>
        <reportName>${ta.tmcallback.reportname}</reportName>
        </tmCallBack>
    </configurers>
</configuration>

    <!-- Bind the Squash TA "run" goal to the maven integration-test_
↪phase and reuse the default configuration -->
    <executions>
        <execution>
            <goals>
                <goal>run</goal>
            </goals>
        </execution>
    </executions>
</plugin>
</plugins>
</build>

<!-- Squash TA maven repository -->
<repositories>
    <repository>
        <id>org.squashtest.ta.release</id>
        <name>squashtest test automation - releases</name>
        <url>http://repo.squashtest.org/maven2/releases</url>
    </repository>
</repositories>

<!-- Squash TA maven plugin repository -->
<pluginRepositories>
    <pluginRepository>
        <id>org.squashtest.plugins.release</id>
        <name>squashtest.org</name>
        <url>http://repo.squashtest.org/maven2/releases</url>
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
        <releases>
            <enabled>>true</enabled>
        </releases>
    </pluginRepository>
</pluginRepositories>
</project>

```

By default, the following configuration has been set :

- Generation of an html report at the end of the execution.
- Generation of Junit reports at the end of execution with attachment and jenkinsAttachmentMode deactivated.
- **Squash TM** events callback for **Squash TF-TM** link is declared but deactivated.

3.4.2 <exporters>

For more information on the **Squash TF** exporters configuration, please read [this](#).

3.4.3 <configurer>

Currently, there is only one configurer existing in SKF. It is used to configure **Squash TM** events callback for the **TF-TM** link. When activated, this component send progression events to **Squash TM** during the execution of a test suite. If you don't use **TF-TM** link, then you don't need this configurer.

To declare the **Squash TM** events callback in your project :

```
<configuration>
...
<configurers>
  <tmCallBack>
    <endpointURL>${status.update.events.url}</endpointURL>
    <executionExternalId>${squash.ta.external.id}</executionExternalId>
    <jobName>${jobname}</jobName>
    <hostName>${hostname}</hostName>
    <endpointLoginConfFile>${squash.ta.conf.file}</endpointLoginConfFile>
    <reportBaseUrl>${ta.tmcallback.reportbaseurl}</reportBaseUrl>
    <jobExecutionId>${ta.tmcallback.jobexecutionid}</jobExecutionId>
    <reportName>${ta.tmcallback.reportname}</reportName>
  </tmCallBack>
</configurers>
...
</configuration>
```

For automation project using a version before **1.7.0**, you have to use :

```
<configuration>
...
<configurers>
  <configurer implementation="org.squashtest.ta.link.SquashTMCallbackEventConfigurer
  ↪">
    <endpointURL>${status.update.events.url}</endpointURL>
    <executionExternalId>${squash.ta.external.id}</executionExternalId>
    <jobName>${jobname}</jobName>
    <hostName>${hostname}</hostName>
    <endpointLoginConfFile>${squash.ta.conf.file}</endpointLoginConfFile>
    <reportBaseUrl>${ta.tmcallback.reportbaseurl}</reportBaseUrl>
    <jobExecutionId>${ta.tmcallback.jobexecutionid}</jobExecutionId>
    <reportName>${ta.tmcallback.reportname}</reportName>
  </configurer>
</configurers>
```

(continues on next page)

(continued from previous page)

```
...  
</configuration>
```

Note: Since **Squash TA 1.7.0**, the endpointURL has a default value : `file://dev/null`. Moreover this default value has for effect to deactivate the send event mechanism. A valid URL should be given to activate it.

A script is at the basis of an SKF automation test. In this section, we're going to see the different elements of an SKF Script and how it is built.

4.1 Sections

Contents :

- *Test case script*
- *Metadata section*
 - *Declaration*
 - *Content*
 - * *Metadata key*
 - * *Metadata value*
 - *Example*

4.1.1 Test case script

As we have seen in our example, a typical test case script contains 4 sections, including 1 Metadata section and 3 execution phases :

Label	Occurrence
METADATA :	optional
SETUP :	optional
TEST :	exactly one
TEARDOWN :	optional

Note that the space-columns ' :' are part of the label. A script must contain exactly one **TEST :** phase and may contain up to one **METADATA :** section, and up to one **SETUP :** and/or **TEARDOWN :** phase. The phases may be declared in any order, but the **METADATA :** section must come first. When a label is declared, no other element is allowed on the same line.

A section begins with a label and ends when another label begins or when the end of file is reached. Instructions must be placed within the boundaries of a section, and any instructions out of a section (mainly when placed before the first label) will cause an error and stop the processing.

Table 1: Correct test script structure

METADATA : //a comment key1 : value1 key2 : value2
SETUP : //another comment an instruction another instruction
TEST : some instructions //another comment lots of instructions...
TEARDOWN : more instructions //other comment and a last instruction

Table 2: Minimal test script structure

TEST : An instruction Another instruction Some more instructions...
--

4.1.2 Metadata section

Declaration

This section can be declared by writing **"METADATA :"** in a typical Squash Test script, not in an Ecosystem one.

In fact, a typical Test script may or may not contain a Metadata section. However, this section, if any, must be **unique**. The Metadata section must also be placed **before** any execution phases (SETUP, TEST or TEARDOWN).

Content

A Metadata section can contain only empty lines, comment lines and, of course *metadata lines*. While an empty line must contain nothing or only spaces/tabulations and a comment line must start with the symbol '//', a metadata line is always 1 of these 3 types :

- **key**
- **key : value**
- **: value** (allowed only after a metadata line of second type)

The symbol ':' is the *separator*, used between a metadata key and its value.

Note: Spaces/tabulations between the separator and its key or/and value are not mandatory but **strongly advised**.

Metadata key

A metadata key can contain only **alphanumeric characters, dashes, underscores and dots**, and stops at the first space/tabulation found or at the end of the line. Moreover, no inline space/tabulation is allowed before a metadata key.

Metadata key is **case insensitive** and must be **unique** in a Test file.

Metadata value

A metadata value is always placed after a separator ":". It can contain **alphanumeric characters, dashes, underscores, dots and slashes**, and stops at the first space/tabulation found or at the end of the line.

Metadata value is **case sensitive** and must not be empty (i.e. there must be at least a letter/character after the separator ":").

A metadata value must be assigned with a metadata key. If a key has more than one values, the first value will be written with its key as: "**key : value**". Other values will be declared of type : "*[space/tabulation]* **: value**".

Important: Between "**key : value**" line and its following "**: value**" lines, comment lines are allowed, but NOT empty lines.

Example

```
METADATA :
// This is an example of metadata
super_key   : first-value
              : /home/user/test.ta
//comment line allowed here but NOT empty line
: third-value_with.and/and--
another.key : and_its-value

// and another comment line after an empty line
key_without_value
//or an example with Squash TM test case UUID
linked-TC : b7ed5ea4-d4f5-177b-c2b2-47581b4c3188

SETUP :

TEST :

TEARDOWN :
```

4.2 Resource Components

Contents :

- *Resource, repository and target*
 - *Resource*
 - *Repository*
 - *Target*
- *Foreword about repositories and Targets configuration*
- *Test and Ecosystem contexts*
- *Identifiers (names in the context)*
- *Reserved names / identifiers*

4.2.1 Resource, repository and target

Instructions and shortcuts are using resources and targets (Targets to test). Resources are included in repositories (libraries of resources). There are 3 kinds of resource components within SKF :

Resource

An SKF resource is a data. More precisely, it is a container which includes certain content (the data) and a category which qualifies the type of data.

It may come from various origins : your own test data, results from the SUT (System Under Test) or derived from another resource.

SKF has defined one category of resource : the *file* resource. This is the basic category of resource in SKF and is nothing more than a handle on a file.

This type of resource is very common in SKF scripts since any resource loaded from a repository (see below) will first be assumed to be a *file* resource before it can be converted to a more appropriate type (examples of resource categories: *file*, *xml*, *script.sql*...).

When a resource is loaded, it's created in a specific context and is available only for this context (See below *Test context / Ecosystem context*).

Repository

A repository represents a physical resources location where you will be able to access your data and bring them in the Test context as *file* resources.

Repositories are read-only by design : their content cannot be altered by a script execution.

It's defined by :

- A `.properties` file. The name of the file will be the identifier of the resource repository in the test context.
- Only one resource repository must be defined per `.properties` file.
- Repository categories : *URL*, *FTP*, *classpath*.

Here are some examples we can find in the 'repository' directory of an automation project :

Example 1 : ftp_example.properties

```
squashtest.ta.ftp.host=192.168.2.24
squashtest.ta.ftp.username=squash
squashtest.ta.ftp.password=squash
squashtest.ta.ftp.system=windows
squashtest.ta.ftp.port=21
```

Example 2 : url_example.properties

```
squashtest.ta.url.base=http://repo.squashtest.org/maven2/releases/eclipse/jdtcore/
```

Target

Targets represent the SUT (System Under Test). They may stand for a http, a ftp server, a SSH server or a database. They may be in read-write mode. It's defined by :

- A `.properties` file. The name of the file will be the identifier of the target in the test context.

- Only one target must be defined per file.
- Target categories : *database*, *http*, *FTP*, *SSH*.

Here are some examples we can find in the target directory of an automation project :

Example 1 : ‘yahoo.properties’

```
#!http
squashtest.ta.http.endpoint.url=http://www.yahoo.fr
```

Example 2 : ‘webcalendar_DB.properties’

```
#!db
squashtest.ta.database.driver=com.mysql.jdbc.Driver
squashtest.ta.database.url=jdbc:mysql://localhost:3306/webcalendar-db
squashtest.ta.database.username=webcalendar
squashtest.ta.database.password=squash
```

4.2.2 Foreword about repositories and Targets configuration

Every repository and target are configured using a `.properties` file dropped in the ‘repositories’ / ‘targets’ directory of your test project.

Each file will create one instance of the corresponding repository / target.

The name under which they are published in the Test context is the name of their configuration file minus the extension (i.e., if the configuration file is `myrepo.properties` then the name in the Test context will be ‘myrepo’).

Every repository and target in the plugins shipped by default in SKF supports overrides using system properties.

Combined with the configuration file itself, a repository can read its configuration from three levels :

- ‘Prefixed’ system properties.
- The ‘Normal’ configuration file itself.
- ‘Default’ system properties.

Those level are hierarchical : ‘Prefixed’ > ‘Normal’ > ‘Default’. The higher level at which a property was found defines the definitive value of that property.

For example, consider a property: ‘squashtest.ta.my.property’ defined in a file named ‘myrepo.properties’. The following ways to set that property are all valid :

level	property name	declaration location
‘Prefixed’	myrepo. /squashtest.ta.my.property	system properties
‘Normal’	squashtest.ta.my.property	configuration file
‘Default’	squashtest.ta.my.property	system properties

As you can see the ‘Prefixed’ level reads a meta property which simply results from appending the property to the name of the Repository or Target (not the full name of the file, `myrepo.properties`).

4.2.3 Test and Ecosystem contexts

An ecosystem is a succession of test cases which have in common a **SETUP** phase and a **TEARDOWN** phase.

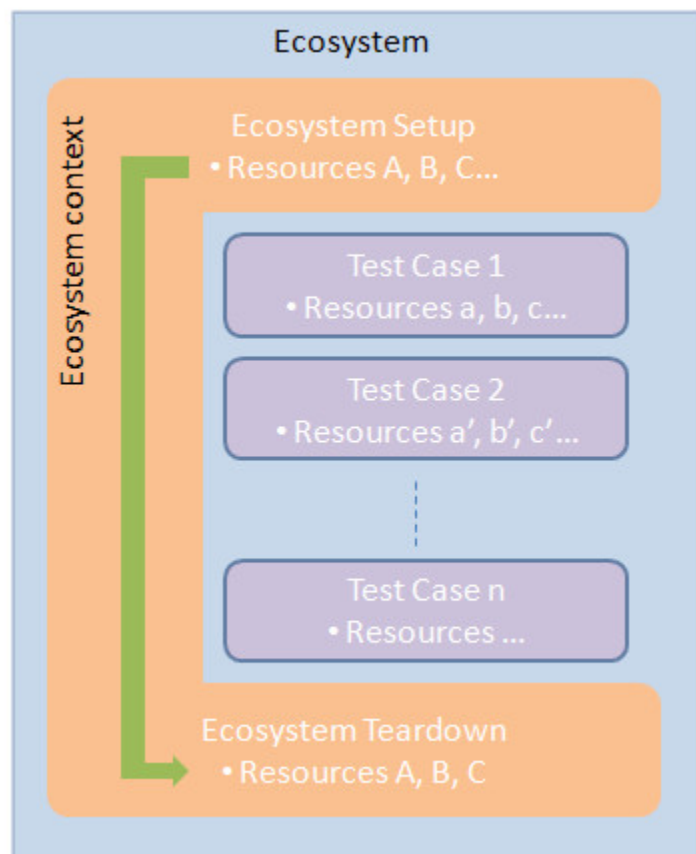
From this, we can figure out two distinctives execution context :

- **Test context**
- **Ecosystem context.**

The **test context** contains the resources for the duration of one single test. Resources can be retrieved from the test context using its name.

The **ecosystem context** contains the resources for both ecosystem's **SETUP** AND **TEARDOWN** phases.

In other words, resources created in the ecosystem setup are also available in the ecosystem teardown whereas those created during a test case are only available for this test case.



For more information, please read the following [page](#).

4.2.4 Identifiers (names in the context)

Almost anything in the test context has an identifier. To be valid, an identifier (for a Resource, a Target or an Engine Component) can be composed of :

- Any lowercase or uppercase a-z letters.
- Digits.
- Dashes '-', underscores '_', dots '.' and slashes '/'.

Examples of valid identifiers : `my_resource05` `Another.Test.File` `testfolder/My_File.txt`

Example of invalid identifier : `<`-no space allowed `$i_c1/2i_c1/2}` `{i_c1/2#` etc `<`-uncommon characters

You should avoid using any file or folder which does not comply with those rules.

When it comes to Resources, a good practice is to mention in the identifiers what kind of resources they actually are.

For instance, when loading a file, a good name would be `my_resource.file`. This is not mandatory but it really helps.

In the rest of this document we will commonly use 'name' as a synonymous for 'identifier'.

4.2.5 Reserved names / identifiers

There are two reserved names / identifiers in SKF (for *context parameters*) :

- `context_script_params`
- `context_global_params`

Note that the framework doesn't prevent you to define your own SKF resource with those context names. If you ever do it, your context parameters will be overwritten (and a warn is logged).

4.3 Macros

Contents :

- *What does a shortcut look like ?*
- *How do I use a macro ?*
- *Variabilized shortcuts*

The basic instructions set covers your need for writing working scripts, but they might be quite verbose. If you find yourself writing the same group of instructions again and again, you will probably find a use for shortcuts (or macros).

A certain number of shortcuts are defined natively in the SKF modules.

4.3.1 What does a shortcut look like ?

A shortcut is simply a sequence of instructions put in a separate file, that defines a hook of your choice that will be replaced by a set of instructions defined below. The syntax is the following :

```
# <macro expression>
=>
<instruction 1>
<instruction 2>
...
<instruction n>
```

The part above the => separator is the **hook** and the part below is the **expansion**.

Spacing characters don't matter : you may put any spaces or tabulations between every word, either in the hook or in the expansion, or before and after the separator.

Please respect the following rules :

- The hook must :
 - Hold in one single line.
 - Be the first line of the shortcut file.
 - Have a space between # and <macro expression>. This is mandatory.
- After the **hook**, the next line must immediately have the separator =>.
- The **expansion** must comply to the rules of the basic instruction set (one instruction per line, etc...).

4.3.2 How do I use a macro ?

The file you just created must land in the `shortcuts` directory (or its subdirectories) of your test project, and the filename must end with a '.macro' extension.

You can now write regular tests using your shortcut just like any other basic instruction. You don't have to respect your hooks to the letter : lowercase and uppercase characters are matched equally, and you may put any extra spaces you like. When your script is executed, any hook encountered will be replaced by the corresponding expansion.

Example : load ‘myfile.xml’ and convert it to type ‘dataset.dbunit’ under the name ‘mydataset.dataset’

```
# load my favourite dataset
=>
//remember that, although the original extension of
//the file is .xml, as a resource in the Test context,
//its initial type is ‘file’, not ‘xml’
LOAD myfile.xml AS myfile.file
CONVERT myfile.file TO xml AS myfile.intermediate.xml
CONVERT myfile.intermediate.xml TO dataset.dbunit AS mydataset.dataset
```

Macro usage :

```
# load my favourite dataset
```

4.3.3 Variabilized shortcuts

You can define variables in the hook and use them in the expansion. When a variable is defined in the hook it won’t be literally matched, the hook will match solely on the basis of the other tokens. The variables defined in the hook may then be used in the expansion. Variables are declared within curly braces ‘{}’, and are used as follow :

Example : Definition of the macro

```
# LOAD {file_name} TO XML DATASET {converted_name}
=>
LOAD {file_name} AS data.file
CONVERT data.file TO xml (structured) AS data.file
CONVERT data.xml TO dataset.dbunit (dataset) AS {converted_name}
```

Macro usage :

```
# LOAD foo.xml TO XML DATASET foo.dataset
```

Corresponding instructions :

```
LOAD foo.file AS data.file
CONVERT data.file TO xml (structured) AS data.file
CONVERT data.xml TO dataset.dbunit (dataset) AS foo.dataset
```

At some point you will probably have to create temporary variables, and thus have to worry about possible conflicting Resource identifiers.

Thankfully there is a mechanism of randomization that helps to tackle the problem, using a special expression in the extension that will generate a random number between -32768 and 32767.

It looks like this :

`{%%whatever}`, where **whatever** is a string of your choice.

When the expression `{%%whatever}` is used in a resource name inside a macro, it's replaced by a string dynamically generated.

If an identical expression `{%%whatever}` is used several times inside a macro, it's replaced each time with the same value.

If two different expressions `{%%whatever}` are used inside a macro (for example `%%data1` and `%%data2`), they're replaced by two different values.

When a script is processed and matches the hook, the variables will be remembered and replaced at their corresponding place in the expression, and placeholders will be filled as well.

Let's rewrite the previous example :

Example : The same shortcut than above, with variables

```
# LOAD {file_name} TO XML DATASET {converted_name}
=>
LOAD {file_name} AS __{%%data1}.file
CONVERT __{%%data1}.file TO xml (structured) AS result{%%data2}.xml
CONVERT result{%%data2}.xml TO dataset.dbunit (dataset) AS {converted_name}
```

Macro usage :

```
# LOAD foo.xml TO XML DATASET foo.dataset
```

Corresponding instructions :

```
LOAD foo.file AS __354.file
CONVERT __354.file TO xml (structured) AS result6345.xml
CONVERT result6345.xml TO dataset.dbunit (dataset) AS foo.dataset
```

4.4 Ecosystem

Contents :

- *Introduction*
- *Workflow*
- *Ecosystems Setup & Teardown scripts*

4.4.1 Introduction

Let's introduce the ecosystem notion in SKF. An ecosystem is a suite of test cases which have in common a setup phase and a teardown phase.

It allows to prepare the environment for a specific bunch of test cases and doing so for as many ecosystems as you need.

Each `tests` directory of an SKF project and its subdirectories correspond to an ecosystem as soon as they contain test cases.

The name of the ecosystem (Such as it will appear in the execution reports) is determined by the name of the directory which contains it.

The ecosystem directory contains :

- **A `setup.ta` file** (optional) : To prepare the environment for the bunch of test cases included in the ecosystem.
- **From 1 to N test files** (`<test_name>.ta`) : Each file corresponding to one test case.
- **A `teardown.ta` file** (optional) : To clean-up the environment after the execution of all test cases included in the ecosystem.

An ecosystem execution takes place in 3 ordered steps (Independent of the order in the directory) :

1. A setup phase where the SKF script **`setup.ta`** is executed (if present). This phase is executed only once.
2. A test cases execution phase during which each test case is executed one after the other.
3. A teardown phase where the SKF script **`teardown.ta`** is executed (if present). This phase is executed only once.

Here is an example with 5 different ecosystems in a **Squash TF** automated project :

4.4.2 Workflow

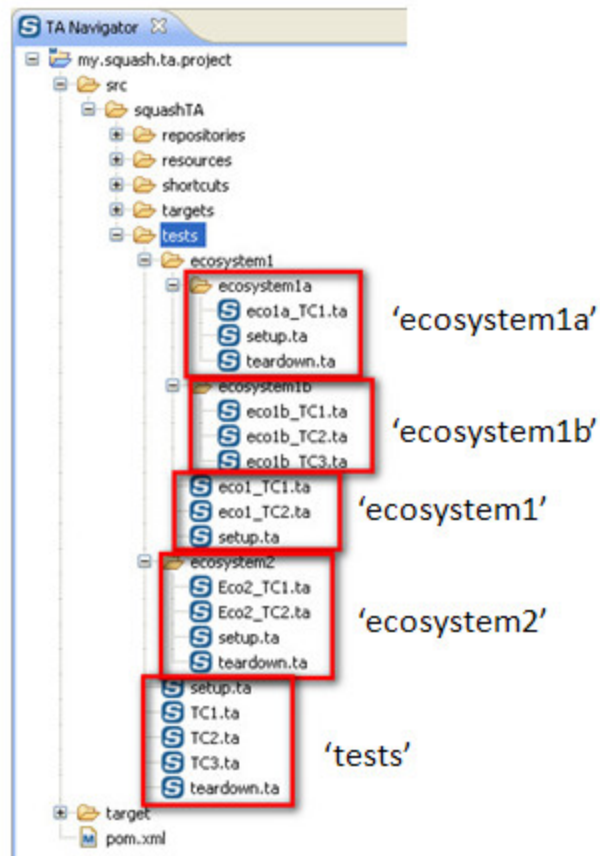
First there is an initialization of contents of the following directories : targets, repositories and shortcuts. At this step the different elements of those directories are verified. If everything is ok these elements are available for the full duration of the execution, therefore for all different ecosystems to execute.

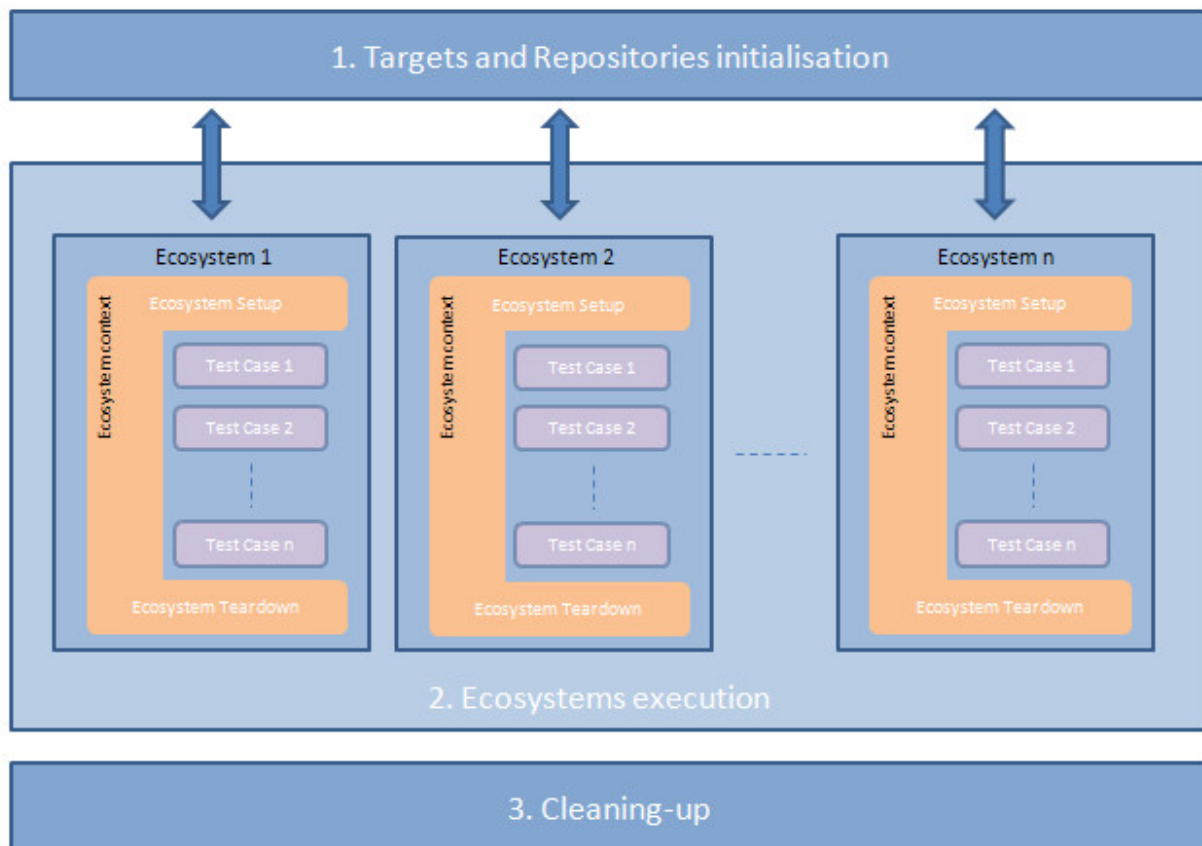
Afterwards comes the execution of the different ecosystems. Each of them with its own *ecosystem context* and for each test case its own *test context*.

Whatever status an ecosystem has after its execution, the next ecosystem of the test suite is launched.

Once all ecosystems have been executed a cleaning-up step occurs, and finally the publication of the execution report available in the 'target' directory.

Here is a schema of the SKF workflow :





4.4.3 Ecosystems Setup & Teardown scripts

As said before, an ecosystem is a suite of test cases which have in common a setup phase and a teardown phase. An ecosystem may - but is not required to - define up to one setup script and one teardown script. They obey to the same rules than regular test scripts, except two points :

- They don't care about phases : A setup or teardown script only contains instructions, no phase should be defined.
- Regular resource names have a special context : the ecosystem context, which is common to BOTH setup and teardown ecosystem script and which is INDEPENDANT (in term of resources) of the test cases included in the ecosystem.

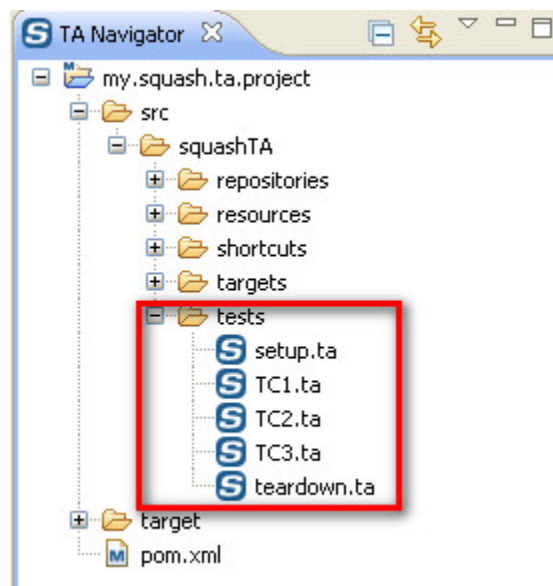
Note that unlike other tests (that may be named freely), setup and teardown scripts **MUST** be named respectively **'setup.ta'** and **'teardown.ta'**.

Example : valid setup/teardown script

```
* - //that's right, no phase needs to be defined
  //instructions are directly written as follow :

instruction
instruction
instruction
```

An example of project containing ecosystems setup and teardown scripts :



4.5 Writing tests - Advanced Users

4.5.1 Instructions

Contents :

- *Syntax convention*
- *Comments*
- *DEFINE instruction / Inlined instruction \$(...)*
- *LOAD instruction*
- *CONVERT instruction*
- *EXECUTE instruction*
- *Assertion instructions (ASSERT / VERIFY)*

The valid instruction set is defined as follow :

```
Blank lines (no instruction)
Comments : starting by a double slash '/' (no instruction)
DEFINE $(raw data) AS {nameInTheContext<Res:File>}
LOAD {path_To_Resource} [ FROM {resourceLibrary<Rep>}] [ AS {nameInTheContext<Res:File>} ]
CONVERT {resourceToConvert<Res>} TO {<Cat:Res>} ( {<Conv>} ) [ USING {config<Res>} ] AS {convertedResource<Res>}
EXECUTE {<Cmd>} WITH {<Res>} ON {<Tar>} [ USING {config<Res>} ] AS {result<Res>}
ASSERT {resourceToTest<Res>} ( IS | HAS | DOES ) {<Asr>} [ ( WITH | THAN | THE ) {expectedResult<Res>} ] [ USING {config<Res>} ]
VERIFY {resourceToTest<Res>} ( IS | HAS | DOES ) {<Asr>} [ ( WITH | THAN | THE ) {expectedResult<Res>} ] [ USING {config<Res>} ]
```

Note: The VERIFY instruction is available since **Squash TA 1.6.0**. It's a new type of assertion instruction.

Syntax convention

- Red words : They represent the language tokens. They are in uppercase and they never change.
- Black words : They represent a physical resource.
- Blue words : Identifiers which point to a resource component. They have the following structure : {name<Type:Category_name>} or {name<Type>} or {<Type>} with :
 - **name** : A name which corresponds to the element that should be pointed by the identifier.
 - **Type** : The component type of the element pointed by the identifier : Res for resources, Tar for targets, Repo for repositories.
 - **Category_Name** : The category of the component which wraps the pointed element.
- Pink words : Identifiers which reference an engine component : {<Cmd>} for commands, {<Asr>} for assertions and {<Conv>} for converters.
- Yellow word : The category of the expected resource after a conversion.

- `[]` : Element inside this square brackets can be omitted in some cases.

Note: For convenience, **name** is often use instead of **identifier** in the documentation.

One instruction per line and one line per instruction. In other words, the end of line means that the instruction ends here and will be parsed as is. The language tokens are case-insensitive and accept inline resource definitions (just like in a `DEFINE` instruction, see below). On the other hand the identifier we discussed above are case-sensitive (i.e. you should respect lowercase and uppercase letters).

An instruction can be divided into clauses. Some are mandatory while others are optional. A clause can be recognized by its language token (uppercased words) and an identifier that immediately follows it.

For each instruction the most obvious mandatory clause is the first one that states which instruction you are referring to. This first clause is also named head clause.

The optional clauses are stated here between enclosing brackets `[]`.

Caution: Those brackets aren't part of the language and just serve the purpose of delimiting those optional clauses.

Except for the head clause which determines the kind of instruction, the order of other clauses is not fixed.

Also note that the DSL does not support nested instructions.

Comments

TA Scripts can contain comments. They start with a `/**`. To write a multiline comment, start each line of the comment with the `/**`. It's not allowed to write a comment on the same line that an instruction.

Example of a not allowed comment :

`LOAD example.txt AS example.file /**This comment is not allowed`

DEFINE instruction / Inlined instruction `$(...)`

`DEFINE $(raw data) AS {nameInTheContext<Res:File>}`

> Input :

- raw data : A string (If there is more than one line, each line must be separate with `"\n"`)

> Output :

- {nameInTheContext<Res:File>} : The identifier of the resource created in the test context.

The DEFINE instruction is rarely used but may come handy. Basically it let you define any text content directly within the script, and binds it to a name.

This content will be stored in the Test context as a *file* resource, under the name supplied in AS clause.

This resource will be available throughout the whole test but won't exist anymore when another test begins.

Example 1 : Simple DEFINE resource

```
DEFINE $(select * from MY_TABLE) AS query.file
```

Example 2 : Structured DEFINE resource

```
DEFINE $(some letters, a tabulation t and n the rest after a linefeed.) AS structured-text.file
```

A more common use for resource definition is to simply inline them within the instruction that will use it.

Example : Resource inlined in a CONVERT instruction

```
CONVERT $(select * from MY_TABLE) TO query.sql AS my_query.query.sql
```

The advantage of explicitly using DEFINE is to bind the newly created *file* resource to a name, thus allowing you to refer to it again later in the script. If you won't need to reuse that resource, an inlined definition is fine.

Inlined resources are notably useful when passing configuration to Engine Components. Engine Components sometimes need a few text to be configured properly, which can be inlined instead of explicitly creating a file for it.

LOAD instruction

```
LOAD {path_To_Resource} [FROM {resourceRepository<Rep>}] [AS {nameInTheContext<Res:File>}]
```

> Input :

- {path_To_Resource} : The path to the resource to load
- {resourceRepository<Rep>} : The name of the resource repository in which is located the resource to load.

> Output :

- {nameInTheContext<Res:File>} : The name of the resource created in the test context.

The LOAD instruction will search for a resource in all of the existing repositories. When it is finally found it will be brought to the test context as a *file* resource. If no AS clause is supplied, the name of this *file* resource will be the name under which it was searched for (including folder hierarchy if it was hidden in a deep file tree).

The path of the resource doesn't need to be a full URL, as that kind of details will be handled by the repositories. In case of a repository looking for the file system it generally have a base directory, you can then omit the full path and only supply a path relative to the base directory.

Also note that the directory separator is a slash '/' regardless of the underlying operating system. More precisely, no backslashes '' needed under Windows. Backslashes aren't a valid character for an identifier and will be rejected anyway.

If by chance two or more repositories could answer the query (i.e. if a given file name exists in two file systems, each of them being addressed by a distinct repository), the *file* resource returned depends on which of them replied first. Consider it as random, and if problems happen you could be interested in the FROM clause (see below).

If the loading fails because the resource was not found, the test will end with a status depending on the phase it was executed in.

The FROM clause is optional. If specified, instead of searching every repository for the resource it will search only the one you specified. It may speed up file retrieval if some of repositories are very slow or busy.

The AS clause is optional. If specified, instead of binding the new *file* resource to the name used in the first clause, the engine will bind it to this alias instead.

Example 1 : Simple file loading

```
LOAD data-folder/myfile // that's it, the file resource will be accessible under the name 'data-folder/myfile'
```

Example 2 : Load with alias

```
LOAD long/path/to/the/resource AS my_resource.file
```

Example 3 : Load from a specific repository

```
LOAD myfile FROM my.repository
```

CONVERT instruction

```
CONVERT {resourceToConvert<Res>} TO {<Cat:Res>} ( <Conv> ) [ USING {config<Res>} ] AS {convertedResource<Res>}
```

> Input :

- {resourceToConvert<Res>} : The name of the resource to convert
- {<Cat:Res>} : The category of the resource expected after the conversion.
- <Conv> : The category of the converter used for the conversion.
- {config<Res>} : The name of the complementary resource needed for the conversion.

> Output :

- {convertedResource<Res>} : The name of the converted resource.

The CONVERT instruction will take an input resource and produce a new resource, that will then be available under the name mentioned in the AS clause. The resource must exist in the Test context beforehand (for instance as resulting from a LOAD instruction).

Remember that no Engine Component will ever modify the input resource, and it will still be available as it was after the conversion is over.

Depending on the invoked converter, a CONVERT instruction will perform at least one of the two operations :

- Produce a resource with the same data than the input resource but wrapped in a different category.
- Produce a resource with new data based on the input resource but the category stays the same.

Some converters do even both. In any case you should refer to the documentation of this converter.

The TO clause is mandatory, as it is where you specify the category of the output (which may be the same than the category of the input resource).

However in some cases, it may happen that two or more converters, accepting the same input and output categories, exist together in the engine, thus leading to an error.

In such cases one should deambiguate the situation by specifying which specific converter you need.

This is the only case where you need to expand the full signature of that converter. You can specify that converter by immediately appending its name to the output category, surrounded by parenthesis ().

Warning: Even in the cases where you don't need to specify the converter name, we highly advise you to do it. Indeed this could prevent you from encountering problems if a new converter with the same input and output is created (making mandatory to specify the converter category).

The optional USING clause lets you specify an arbitrary number of resources that will be treated as configuration for this operation. The category of resources, or which informations they should convey depends on the converter being used. Having a look at the documentation of that converter is certainly useful.

Example 1 : Simple CONVERT from file to CSV

```
CONVERT mydata.file TO csv AS mydata.csv
```

Example 2 : CONVERT with configuration

```
CONVERT my_result.resultset TO dataset.dbunit USING $(tablename : MY_TABLE) AS mydata.csv
```

Example 3 : CONVERT with an inline definition

```
CONVERT $(select * from MY_TABLE) TO query.sql (query) AS my_query.query.sql
```


EXECUTE instruction

```
EXECUTE {<Cmd>} WITH {<Res>} ON {<Tar>} [ USING {config<Res>} ] AS {result<Res>}
```

> Input :

- {<Cmd>} : The command to execute.
- {<Res>} : The name of the resource to use with the command.
- {<Tar>} : The name of the target.
- {config<Res>} : The name of the complementary resource needed to use with the command.

> Output :

- {convertedResource<Res>} : The name of the resource generated by the command.

The EXECUTE instruction will perform an operation involving a resource (WITH clause), on a given target (ON clause). The result of this operation, if any, will be returned as a resource published in the Test context under the name supplied in the AS clause.

If the operation returns some results, the actual type of the resulting resource depends on the command being executed, so you should refer to the documentation of that command to know how to handle it in the rest of the test.

The optional USING clause lets you specify an arbitrary number of resources that will be treated as configuration for this operation. The category of resources, or which informations they should convey depends on the command being used.

You MUST provide an input resource, a target and an alias for the result, even if the command does not actually use all of these features.

Example 1 : Command using a dummy identifier for the result name (because that command doesn't return any)

```
EXECUTE put WITH my_file.file ON my_ftp AS no_result_anyway
```

Example 2 : Command with configuration

```
EXECUTE get WITH $() ON my_ftp USING $(remotepath : data/the-file.txt, filetype : ascii) AS my_new_file.file
```

Note that in the last example we used a dummy inlined resource \$(), since in that case the **get** command doesn't use any input resource.

Assertion instructions (ASSERT / VERIFY)

```
ASSERT {resourceToTest<Res>} ( IS | HAS | DOES ) {<Asr>} [ ( WITH | THAN | THE ) {expectedResult<Res>} ] [ USING {config<Res>} ]
VERIFY {resourceToTest<Res>} ( IS | HAS | DOES ) {<Asr>} [ ( WITH | THAN | THE ) {expectedResult<Res>} ] [ USING {config<Res>} ]
```

> Input :

- {resourceToTest<Res>} : The name of the resource to validate.
- {<Asr>} : The kind of assertion to use.
- {expectedResult<Res>} : The name of the reference resource.
- {config<Res>} : The name of the complementary resource needed for the assertion.

The assertion instructions will perform a test on the supplied resource, optionally compared to another resource.

If the assertion is successful, the test will continue.

If the assertion failed or finished in error :

- In ASSERT mode, the execution of the current test phase is stopped. The teardown test phase is then executed (if it was not already in this teardown test phase).
- In VERIFY mode, the next instructions is executed.

The test final status will always be the most severe status of its instructions.

For details on the execution workflow and test status please see [this page](#).

The VERIFY assertion mode is available since **Squash TA 1.6.0**. Before only the ASSERT mode was available.

Note that, unlike other instructions, an assertion can have multiple choices.

The first multi-token clause is the one identifying the assertion ({<Asr>}, in the syntax above).

The second one is identifying the secondary resource ({expectedResult<Res>}, in the syntax above).

In either case you only need to pick one, and it makes sense to pick the one that fits the most to the grammar of the instruction (see examples below).

The optional (WITH | THAN | THE) clause specifies another resource. In that case, the primary resource will be compared to the secondary resource.

If that clause is used, then we talk of a ‘binary assertion’, and the primary resource usually represents the actual result from the SUT while the secondary result represents the expected result.

If no (WITH | THAN | THE) clause is used, the resource and the assertion are assumed self-sufficient to perform the check. We then talk of a ‘unary assertion’.

The optional USING clause let you specify an arbitrary number of resources that will be treated as configuration for this operation. The category of resources, or which information they should convey depends on the assertion being used, so having a look at the documentation of that assertion is certainly useful.

Example 1 : Simple unary assertion

```
ASSERT my_result.result.sahi IS success
```

Example 2 : Simple binary assertion (awkward)

```
ASSERT actual_result.dataset.dbunit IS contain WITH expected_result.dataset.dbunit
```

In this example, the sentence is grammatically wrong but it will work as expected. You might prefer the following syntax :

Example 3 : Simple binary assertion (better)

```
ASSERT actual_result.dataset.dbunit DOES contain THE expected_result.dataset.dbunit
```

This version does exactly the same thing but is better.

4.5.2 Engine Components

Contents :

- *Converters*
- *Commands*
- *Assertions*
- *Common behavior of an Engine Component*

Scripts are written using a DSL (Domain Specific Language) that provides very basic ways to invoke parts of the SKF engine. Those parts are known as the Engine Components and are divided in three categories : Converters, Commands and Assertions.

Converters

They will help you to modify your resources. Usually a converter will handle one task between the following :

- To modify the content (aka data) of a resource.
 - **For example** : To modify the content of a text file.
- To modify the wrapper (aka type) of a resource.
 - **For example** : To convert a text file to a SQL batch.

In either case the resulting resource will be a new resource. Both the previous and the new version exist and are available in the Test Context once the operation is done.

Commands

A command will perform an operation against a Target, usually using a resource. For instance, execute an SQL query against a database, send a file to a FTP etc.

If the operation carries some results they will be brought back to the Test context as resources.

Assertions

An assertion is a check ran on resource(s). Sometimes the tested resources carry all the information you need to test. In this case we speak of unary assertion.

On the other hand, when you need to compare one resource to another, we speak of binary assertion.

A successful assertion (i.e. the tested resource(s) match(es) the expected criteria) will let the script continue. A failed assertion will stop the script and report the error.

Common behavior of an Engine Component

As resources are immutable, an engine component will never modify a resource. If any new data/result is produced, it will be brought in as a new resource.

An engine component is identified by its name AND by the type of resources and/or targets it manipulates.

For example, let's consider a command named **“put”** that uses SQL batches against databases, and a second command also named **“put”** that uploads files to an FTP.

At runtime when the engine reads an instruction, depending on the name of command, resource and target, it will decide which operation will be ran.

An engine component may take additional resources that will tweak its behavior. Those (mostly optional) resources are called configuration (or parameters).

Example :

Let's imagine you want to upload multiple files to an FTP (e.g. using the command **“put”** above) using the defaults settings associated to the FTP Target. For one of those files, you need to override the settings.

In this case the **“put”** command allows you to mention another resource carrying those overridden values, and will use them for this specific case only.

4.5.3 Example of SKF Script

Contents :

- *What does this SKF script do ?*
- *Some more insights*

In this example, we are going to use an SKF script which contains standard instructions and macros.

A script is a plain text file (with **.ta** extension) containing lists of instructions. All you have to do is to drop it in the 'tests' directory (or anywhere in its sub hierarchy) of your *project*.

Here is the script :

```
// Step 1
SETUP :
// Step 2
LOAD queries/sql/select.sql AS select.file
CONVERT select.file TO query.sql AS query
// Step 3
# LOAD dbunit/resultsets/mytable_simple_select.xml TO XML DATASET expected.dataset
// Step 4
TEST :
// Step 5
EXECUTE execute WITH query ON my_database AS raw_result
// Step 6
CONVERT raw_result TO dataset.dbunit AS actual.dataset
// Step 7
ASSERT actual.dataset IS equal WITH expected.dataset
```

What does this SKF script do ?

This script executes an SQL query and compares the result to the content of a DbUnit dataset.

We can notice several elements :

- Some comments (lines beginning with '//')
- Phases (SETUP : and TEST :)
- Instructions (multi-colored lines)
- Macros (brown lines).

Now we will break down and analyze that script, identify its components and see how they work together.

SETUP phase declaration :

```
// Step 1
SETUP :
```

The SETUP phase groups instructions that will prepare the test. Note that the instructions in that phase could also be set in the main TEST phase.

Differences lie in the handling of test failures : when an instruction fails during the SETUP phase, it means that the script itself is wrong or that the resource is not available. On the other hand, a failure happening during the TEST phase means that the System Under Test (SUT) has a problem.

To sum up, the SETUP phase sets the test prerequisites.

Loading an SQL query :

```
// Step 2
LOAD queries/sql/select.sql AS select.file
CONVERT select.file TO query.sql AS query
```

This pair of instructions will load a file and declare that it contains an SQL query (in that order).

Loading a DbUnit dataset :

```
// Step 3
# LOAD dbunit/resultsets/mytable_simple_select.xml TO XML DATASET expected.dataset
```

The standard way to load a DbUnit dataset requires the same steps than above : load the file and convert it to make it a DbUnit dataset.

However, instead of explicitly writing the corresponding instructions, SKF proposes a shortcut (macro) to achieve that goal.

The line in the script is not syntax colored because it is a macro and not a standard instruction.

TEST phase declaration :

```
// Step 4
TEST :
```

The TEST phase groups the main test instructions. The following instructions will interact with the SUT (System Under Test) and the resources created during the SETUP phase remain available.

If an instruction fails, the test will end and the status displayed in the test result will be set according to the nature of the error.

Execution of the query :

```
// Step 5
EXECUTE execute WITH query ON my_database AS raw_result
```

We use the query created during SETUP and execute it against the database. The name of the database here is **my_database**.

The resulting data are stored in the context under the name supplied at the end of the instruction (**raw_result**).

Data transformation :

```
// Step 6
CONVERT raw_result TO dataset.dbunit AS actual.dataset
```

In step 3 we prepared the (expected) data, formatted as a DbUnit dataset. If we are to compare the actual data with the expected data, we must first convert the actual data to a suitable type. In this case it must be formatted as a DbUnit dataset.

Perform the comparison :

```
// Step 7
ASSERT actual.dataset IS equal WITH expected.dataset
```

Now we're all set to proceed and test the data against each other. The status of the test will depend on the status of that comparison.

If the comparison fails, the test will be flagged as failed, while if the comparison is a success the script continues to the next instruction.

Here there are no further instruction and the test will terminate with SUCCESS status.

Some more insights

So now you should have a glimpse of what an SKF script is made of.

There is a TEARDOWN phase too which is optional (just like SETUP phase) but it was not used in this example.

When looking closer at an instruction (e.g. step 5), we can distinguish two kinds of elements : the tokens (red words) and the variable elements (black, blue, pink and yellow words).

The tokens never change whereas identifiers are variable and refer to elements available in the script, including :

- A file or other physical resource (e.g. step 2 : queries/sql/select.sql).
- The assertion type : a command or a converter (e.g. step 5 : EXECUTE execute WITH query ON my_database AS raw_result).
- The resource type (e.g. step 6 : CONVERT raw_result TO dataset.dbunit).
- The name of a target, a repository, an already loaded resource or a resource to be created (e.g. step 5 : EXECUTE execute WITH query ON my_database AS raw_result).

Basically the tokens tell the engine to execute or activate some components and make them interact with each others.

This section will give you further details about the instructions and the engine components (converters, commands or asserts) of SKF which are used by macros.

An SKF script is a file containing an amount of instructions, resources components, engine components and shortcuts (macros) that will be interpreted by the engine of SKF to execute automation tests.

All those elements form the specific language of SKF to describe automation tests.

It allows to address an heterogeneous panel of tests with a common formalism.

First, we will explain the different *phases* of an SKF script and see what are the *resource components*.

We will also see how *macros* work.

In an *Advanced Users* section, we will explain the *instructions* and *engine components* which are behind the macros.

With those knowledge, you can write your own custom macros if you wish.

If you want to see how macros are used in an SKF script, you can check this *example*.

To see how instructions and engine components are used, please check this other *example of SKF script*.

Execution and Reporting

5.1 Logging

We recommend to patch your maven by using the procedure below for a better logging with our runners :

Note: In all the procedure `$MVN_HOME` is your maven installation directory, and `$MVN_VERSION` your maven version.

- Add in `$MVN_HOME/lib/ext/` the jars :
 - `log4j-slf4j-impl-2.5.jar`
 - `log4j-core2.5.jar`
 - `log4j-api-2.5.jar`
- Create a logging configuration file called `log4j2.xml` in `$MVN_HOME/conf/logging/` and fill it with :

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration>
  <Properties>
    <Property name="maven.logging.root.level">INFO</Property>
  </Properties>
  <Appenders>
    <Console name="console" target="SYSTEM_OUT">
      <PatternLayout pattern="[%p] %msg%n%throwable" />
    </Console>
  </Appenders>
  <Loggers>
    <Root level="${sys:maven.logging.root.level}">
      <Appender-ref ref="console"/>
    </Root>
  </Loggers>
<!-- <logger name="[USER_MESSAGE]" level="DEBUG"/> -->
```

(continues on next page)

(continued from previous page)

```
</Loggers>
</Configuration>
```

- Remove if exists :
 - In the directory `$MVN_HOME/lib` the file `maven-sl4j-provider-$MVN_VERSION.jar`
 - In the directory `$MVN_HOME/conf/logging/` the file `deletesimpleLogger.properties`

5.2 Tests Execution and Reporting

5.2.1 Execute SKF tests

‘run’ goal (squash-ta:run)

The `run` goal of the SKF maven plugin is used to execute one or several tests. By default this goal is associated with the `Integration-test` phase of the maven build life cycle.

To execute ALL tests, you can use one of the command bellow (in the root directory of your project) :

```
mvn integration-test

mvn squash-ta:run
```

Specify the test list to execute

The SKF maven plugin defines a maven property which allows to specify the test list to execute. This option is : `ta.test.suite`.

It’s possible to specify its value by modifying the **pom.xml** of an SKF project :

```
...
<modelVersion>4.0.0</modelVersion>
<groupId>fr.mycompany</groupId>
<artifactId>my-app-automated-tests</artifactId>
<packaging>pom</packaging>
<version>0.0.1-SNAPSHOT</version>
<properties>
  <ta.test.suite>tc1.txt,tc2.txt</ta.test.suite>
</properties>
<build>
...

```

It’s also possible to specify its value in the **command line** :

```
mvn integration-test -Dta.test.suite=tc1.txt,tc2.txt
```

Note:

- If a property is defined in ‘**pom.xml**’ AND via **command line**, the command line value predominates.
- If a property has an empty value, all tests are executed.

There are many ways to define the test list to execute through `ta.test.suite` parameter. More details [here](#).

Manage temporary files

The SKF maven plugin also defines two maven properties which allow to manage temporary files created during execution.

- `ta.temp.directory` : Defines where temporary files should be stored.
- `ta.debug.mode` : Defines if temporary files are deleted or not after the execution (its value must be “true” or “false”).

Same as `ta.test.suite`, it’s possible to define them in the **pom.xml**.

Example :

```
mvn integration-test -Dta.temp.directory=C:Squash_TF_temp
mvn integration-test -Dta.debug.mode=true
mvn integration-test -Dta.temp.directory=C:Squash_TF_temp -Dta.debug.mode=true
```

Note:

- If a property is defined in ‘**pom.xml**’ AND via **command line**, the command line value predominates.
 - `ta.temp.directory` default value is the system temporary directory.
 - `ta.debug.mode` default value is “false”.
-

Attention: If there are syntax-error-metadata in the running test script(s), **warning** message(s) will be displayed in the console. (See [Metadata section](#) for more information about Metadata syntax conventions)

5.2.2 Define Test suite Perimeter

Contents :

- “*ta.test.suite*”: *filters*
 - *Definition*
 - *Usage*
- “*ta.test.suite*”: *json data*
 - *Filtered execution*
 - *Ordered execution*
 - *Usage*

- *Filtered execution vs Ordered execution*
 - *Filtered execution*
 - *Ordered execution*

When you execute your test through the `run` goal, you define the test suite to execute through the `ta.test.suite` parameter (more details on the `run` goal [here](#)).

You can define your test suite by providing to `ta.test.suite` **filters** (for filtered execution) or **data structured in .json** (for filtered or ordered execution).

“ta.test.suite”: filters

Definition

The `ta.test.suite` parameter can be a list of filters separated by comma.

A filter can be :

- The test case path (absolute or relative to the automation project “tests” directory).

The file path `D:/myProject/src/squashTA/tests/foo/bar/baz.ta` can be matched as :

- `D:/myProject/src/squashTA/tests/foo/bar/baz.ta`
- `foo/bar/baz.ta`

- A path using a wildcard characters which selects matching test script inside “tests” directory. Wildcard characters can be used :
 - `**` to replace directories path (one or many levels).
 - `*` to replace 0,1 or many characters.

Examples of file paths matching `foo/bar/baz.ta` using wildcard characters :

- `**/baz.ta`
- `**/bar/baz.ta`
- `foo/*/baz.ta`
- `f*/b*/baz.ta`
- `*/b*z.ta`
- etc.

- A regular expression, using `regex'<myRegex>'` format, which selects matching test script inside “tests” directory.

`regex'<regular_expression>'`
regular_expression : The regular expression to use to select tests.

Usage

In the example below, `ta.test.suite` is composed of two filters :

```
"foo/bar/baz.ta"
```

will select for execution foo/bar/baz.ta file in “tests” directory (if it exists).

```
"sample/**/*.ta"
```

will select for execution all files in “tests/sample” directory and its subdirectories which name finish by “.ta”.

```
mvn squash-tf:run -Dta.test.suite=foo/bar/baz.tf,sample/**/*.ta
```

“ta.test.suite”: json data

Through json data you can do a filtered execution or an ordered execution.

Filtered execution

In the json data you can provide filters (as defined in the previous section) by using the syntax below :

```
{  
  "filter" : "**/*.ta"  
}
```

In addition, you can provide some global parameters :

```
{  
  "filter" : "**/*.ta",  
  "param" : {  
    "property5" : "value13",  
    "property6" : "value14"  
  }  
}
```

Ordered execution

The other possibility, in json format, is to provide the list of tests to execute :

// Path to the test script to execute

// Test execution identifier

// Script parameters

<pre> { "test": [{ "script": "pathToMyscript1", "id": "TestCase1", "param": { "property1": "value1", "property2": "value2" } }, { "script": "pathToMyscript2", "id": "TestCase2", "param": { "property3": "value7", "property4": "value8" } }, { "script": "pathToMyscript1", "id": "TestCase3", "param": { "property1": "value3", "property2": "value4" } }], "param": { "property5": "value13", "property6": "value14" } } </pre>	<pre> - - - <---- Path to the test script to execute <---- Test execution identifier <---- Script parameters - - - - - - - - - - - <---- Global parameters - - - - </pre>
---	---

Where for each test :

- `script` is the the path to the test to execute relative to the “tests” directory. This property is mandatory.
- `id` is the test execution identifier. Only useful for **Squash TM - Squash TF** link. However, if “id” is defined for one test then it should be defined for all tests of test suite.
- `param` is a list of parameters (key/value) associated to the test. This property is optional.

As for json filtered execution, global parameters are optional.

When no `param` and `id` properties are defined for test, it’s possible to use a simplier syntax :

```

{
  "test": [
    "script1Path",
    "script2Path",
    "script3Path"
  ],
  "param": {
    "property5": "value13",
    "property6": "value14"
  }
}

```


Usage

Json data can be provided through a String or a file.

> Json provided through a String

```
mvn squash-ta:run -Dta.test.suite={'test':[{'script':'pathToMyscript1'  
↪','param':{'property1':'value1','property2':'value2'}},{ 'script':  
↪'pathToMyscript2'}]}
```

Note: Note that the double quote surrounding properties and values of the json data has been replaced. You can :

- replace them by simple quote (as it's done in the example)
 - escape the double quote "
-

> Json provided through a file

```
mvn squash-ta:run -Dta.test.suite={file:pathToJsonFile}
```

Where *pathToJsonFile* is the path to the json data file. This path can be absolute or relative to the root directory of the automation project.

Filtered execution vs Ordered execution

Filtered execution

When you do a filtered execution you provide filters. The list of test to execute is composed of all the tests in “tests” directory whose path matches the filter. With this kind of execution :

- A test can only be executed once during an execution.
- There is no execution order.
- You can't provide specific parameters to the script however you can provide global parameters through json data.

Ordered execution

When you do an ordered execution you provide the list of tests to execute through json format. With this kind of execution :

- A test can be executed as many times as needed.
- The tests are executed in the order that they were declared in the json data **if the tests are in the same ecosystem**. If the tests are not in the same ecosystem they are executed ecosystem by ecosystem. That means we execute all the tests of the first ecosystem used (in the order they are declared) then the tests of the second ecosystem are executed, etc.

Given the test tree below :

```
tests
|--foo
|   |--test21.ta
|   |--test22.ta
|--test01.ta
|--test02.ta
```

When this json data is given as input :

```
{
  "test" : [{
    "script" : "foo/test22.ta"
  }, {
    "script" : "test01.ta"
  }, {
    "script" : "test02.ta"
  }, {
    "script" : "foo/test21.ta"
  }]
}
```

Then : test22.ta and test21.ta of ecosystem foo will be executed, then test01.ta and test02.ta will be executed.

- Parameters can be specified for each test.

5.2.3 Reporting

Contents :

- *Configuration*
- *Report Status*
- *SKF HTML Report*
 - *HTML Report Configuration*
 - *HTML Report overview*
- *HTML Summary*
 - *Normal HTML Summary*
 - *Lightened HTML Summary*

- *Surefire Report*
 - *Surefire Report Configuration*
 - *Surefire report overview*

Configuration

All configurations are done in the `<exporters>` section of the `pom.xml` of your project.

A report is defined by the `implementation` attribute of an `<exporter>`. Child tags (of `<exporter>`) are report-dependant.

```
<configuration>
  ...
  <exporters>
    <exporter implementation="org.squashtest.ta.commons.exporter.surefire.
↪SurefireSuiteResultExporter">
      <exportAttached>true</exportAttached>
      <jenkinsAttachmentMode>true</jenkinsAttachmentMode>
    </exporter>
    <exporter implementation="org.squashtest.ta.commons.exporter.html.
↪HtmlSuiteResultExporter"></exporter>
  </exporters>
  ...
</configuration>
```

Report Status

Element :	Status :
Instruction :	<ul style="list-style-type: none">• NOT RUN : Instruction was not executed.• SUCCESS : Instruction was successfully executed.• FAILURE : An assertion has failed.• ERROR : An instruction raised an unexpected error
Test case :	<ul style="list-style-type: none">• NOT_RUN : A phase of a test case was not executed.• Otherwise its status is the severest one among status of its instructions.
Script TF : (Test)	<ul style="list-style-type: none">• NOT_RUN : Script was not executed.• Otherwise its status is the severest one among status of its instructions.
Ecosystem :	<ul style="list-style-type: none">• Its status is the severest one among status of its scripts.

SKF HTML Report

HTML Report Configuration

In our maven archetype, HTML reports are already enabled. Otherwise, you can do it with the following implementation :

```
org.squashtest.ta.commons.exporter.html.HtmlSuiteResultExporter
```

HTML report configuration sample :

```
<configuration>
  ...
  <exporters>
    <exporter implementation="org.squashtest.ta.commons.exporter.html.
    ↪HtmlSuiteResultExporter"></exporter>
  </exporters>
  ...
</configuration>
```

Since 1.7.0 version, it's also possible to enable it with the following configuration :

```
<configuration>
...
  <exporters>
    <html/>
  </exporters>
...
</configuration>
```

Note: Both solutions are equivalent and work since 1.7.0.

HTML Report overview

The HTML Report (squash-ta-report.html) is created post-execution in directory : `my.squash.ta.project/target/SquashTA/html-reports`.

Here are some screenshots of an html report :

- **Test suite summary :**



Automated Test Execution Report

GroupId: org.squashtest

ArtifactId: execution-report-test-sample

Version: 1.0-SNAPSHOT

Execution date: 06/17/2013 11:15:53




















Test Set	Tests	● Successes	● Failures	● Errors	⊙ Not run	Success Rate	Time (s)
● tests.F01.ScriptsTA.Set1	17	2	6	9	0	11,765%	1,574
● tests.F01.ScriptsTA.Set2	1	0	0	0	1	0%	0,017
● tests.F01.ScriptsTA.Set3	1	0	0	0	1	0%	0,01
● tests.F01.ScriptsTA.Set4	1	0	0	1	0	0%	0,043
● tests.F01.ScriptsTA.Set5	1	0	1	0	0	0%	0,034
Total:	21	2	7	10	2	9,524%	1,678


Note:

- "failures" are anticipated and checked for assertions whereas "errors" are unanticipated.
- "not run" means that the test case was not run, due to an error / failure occurred during the ecosystem setup phase.
- The ecosystem might end up with an "error" or "failure" status, even if all its test cases were successful. It means an error / failure occurred during the ecosystem teardown phase.

- **Ecosystem summary :**
- **Target Initialisation summary :** (@since Squash-TA framework 1.7.0)
- **Test script result :**

A full sample is also available here : [Execution report - execution-report-test-sample.pdf](#)

tests.F01.ScriptsTA.Set1		
Script	Status	Time (s)
 setup.ta	Success	0,007
 UC01.01_Success.ta	Success	0,024
 UC01.02_Error_in_setup_phase.ta	Error	0,013
 UC01.03_Error_in_test_phase.ta	Error	0,011
 UC01.04_Error_in_teardown_phase.ta	Error	0,012
 UC01.05_Failure_in_setup_phase.ta	Failure	0,01
 UC01.06_Failure_in_test_phase.ta	Failure	0,009
 UC01.07_Failure_in_teardown_phase.ta	Failure	0,01
 UC01.08_Error_in_setup_phase_verify.ta	Error	0,018
 UC01.09_Error_in_test_phase_verify.ta	Error	0,015
 UC01.10_Error_in_teardown_phase_verify.ta	Error	0,017
 UC01.11_Failure_in_setup_phase_verify.ta	Failure	0,017
 UC01.12_Failure_in_test_phase_verify.ta	Failure	0,013
 UC01.13_Failure_in_teardown_phase_verify.ta	Failure	0,101
 UC01.14_Mix_up.ta	Error	0,033
 UC01.15_syntax_error.ta	Success	0,002
 UC01.16_sahi_error.ta	Error	1,147
 UC01.17_shortcut.ta	Error	0,028
 teardown.ta	Success	0
[Back to top]		

Targets Initialisation	Nature	Status
 unknown-target	unknown	Not found
 target-ok	database	OK
 wrongly-configured-target	database	Error

Note:

- "not found" means the target was not found or was not properly configured. Please check your configuration again.
- "error" means an error occurs during the target initialisation.
- "ok" means the target was properly configured.

UC01.02_Error_in_setup_phase.ta

Status: Error
Time (s): 0,013

Test steps	Line
SETUP :	
LOAD file.txt AS file	2
ASSERT wrongId DOES contain WITH \$(Hello) \$(Hello) --> {__temp709} ASSERT wrongId DOES contain WITH {__temp709} ASSERT wrongId DOES contain WITH {__temp709}: Cannot apply assertion : . SCOPE_TEST:wrongId does not exist in this test context: you must load it first. <pre> org.squashtest.ta.backbone.exception.ResourceNotFoundException: ASSERT wrongId DOES contain WITH {__temp709}: Cannot apply assertion : . SCOPE_TEST:wrongId does not exist in this test context: you must load it first. at org.squashtest.ta.backbone.engine.instructionrunner.AbstractDefaultInstructionRunner.throwResourceNotFound(AbstractDefaultInstructionRunner.java:143) at org.squashtest.ta.backbone.engine.instructionrunner.AbstractDefaultInstructionRunner.throwResourceNotFound(AbstractDefaultInstructionRunner.java:143) at org.squashtest.ta.backbone.engine.instructionrunner.DefaultBinaryAssertionRunner.fetchResourceOrFail(DefaultBinaryAssertionRunner.java:105) at org.squashtest.ta.backbone.engine.instructionrunner.DefaultBinaryAssertionRunner.getActualResult(DefaultBinaryAssertionRunner.java:130) at org.squashtest.ta.backbone.engine.instructionrunner.DefaultBinaryAssertionRunner.doRun(DefaultBinaryAssertionRunner.java:85) at org.squashtest.ta.backbone.engine.instructionrunner.AbstractDefaultInstructionRunner.run(AbstractDefaultInstructionRunner.java:69) at org.squashtest.ta.backbone.engine.impl.TestRunnerImpl.runInstruction(TestRunnerImpl.java:138) at org.squashtest.ta.backbone.engine.impl.TestRunnerImpl.runInstructionList(TestRunnerImpl.java:139) at org.squashtest.ta.backbone.engine.impl.TestRunnerImpl.runMetaInstruction(TestRunnerImpl.java:161) at org.squashtest.ta.backbone.engine.impl.TestRunnerImpl.runInstructionList(TestRunnerImpl.java:138) at org.squashtest.ta.backbone.engine.impl.TestRunnerImpl.runPhase(TestRunnerImpl.java:135) at org.squashtest.ta.backbone.engine.impl.TestRunnerImpl.runTest(TestRunnerImpl.java:105) at org.squashtest.ta.backbone.engine.impl.EcosystemRunnerImpl.runAllTests(EcosystemRunnerImpl.java:103) at org.squashtest.ta.backbone.engine.impl.EcosystemRunnerImpl.run(EcosystemRunnerImpl.java:104) at org.squashtest.ta.backbone.engine.impl.SuiteRunnerImpl.execute(SuiteRunnerImpl.java:64) at org.squashtest.ta.backbone.engine.impl.EngineImpl.execute(EngineImpl.java:54) at org.squashtest.ta.maven.SquashTA Mojo.executeImpl(SquashTA Mojo.java:105) at org.squashtest.ta.maven.AbstractSquashTA Mojo.execute(AbstractSquashTA Mojo.java:114) at org.apache.maven.plugin.DefaultBuildPluginManager.executeMojo(DefaultBuildPluginManager.java:101) at org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:209) at org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:155) at org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:145) at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(LifecycleModuleBuilder.java:84) at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(LifecycleModuleBuilder.java:59) at org.apache.maven.lifecycle.internal.LifecycleStarter.execute(LifecycleStarter.java:183) at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:328) at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:156) at org.apache.maven.cli.MavenCli.execute(MavenCli.java:537) at org.apache.maven.cli.MavenCli.doMain(MavenCli.java:196) at org.apache.maven.cli.MavenCli.main(MavenCli.java:181) at sun.reflect.NativeMethodAccessorImpl.invoke(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:59) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25) at java.lang.reflect.Method.invoke(Method.java:597) at org.codehaus.plexus.classworlds.launcher.Launcher.launchEnhanced(Launcher.java:298) at org.codehaus.plexus.classworlds.launcher.Launcher.launch(Launcher.java:239) at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:489) at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:552) </pre>	3
ASSERT file DOES contain WITH \$(Hello)	4
TEST :	
ASSERT file DOES contain WITH \$(Hello)	7
TEARDOWN :	
ASSERT file DOES contain WITH \$(Hello)	10
Attachments	
EXPECTED RESULT-{{ __temp709 }}	

[\[Back to top\]](#) [\[Back to tests.F01_ScriptsTA_Set1\]](#)

HTML Summary

The HTML Summary is a less detailed version of the HTML report.

Normal HTML Summary

If you want to have an HTML Summary report, you have to add the following lines in `<exporters>` :

```
<exporters>
  <htmlSummary>
    <outputDirectoryName>directoryOfHtmlSummary</outputDirectoryName>
  </htmlSummary>
</exporters>
```



Rapport d'exécution des tests automatisés

Composant de test: test:project

Version: 1.0-SNAPSHOT

Date d'exécution: 09/04/2019 14:43:43

Total	● (Succès)	● (Échec)	● (Erreur)	● (Non exécuté)	● (Non trouvé)	Taux de succès
1	0	1	0	0	0	0%

Groupe de tests	Test	Statut
tests	first-test.ta	●

● tests/first-test.ta

The pattern '<?xml version="1.0"?>' was not found in the file.

Pièces attachées

[EXECUTION_REPORT-detectedForbiddenContext.txt](#)

[\[Retour en-tête\]](#)

Lightened HTML Summary

You can lighten the report if you add to the `<htmlSummary>` configuration the tag :


```
<includeHeader>>false</includeHeader>
```

Which gives :

```
<exporters>
  <htmlSummary>
    <outputDirectoryName>directoryOfHtmlSummary</outputDirectoryName>
    <includeHeader>>false</includeHeader>
  </htmlSummary>
</exporters>
```



Rapport d'exécution des tests automatisés

● tests/first-test.ta

The pattern '<?xml version="1.0"?>' was not found in the file.

Pièces attachées

[EXECUTION_REPORT-detectedForbiddenContext.txt](#)

Note: It's possible to have both **HTML report** and **HTML Summary** together : Put the two tags in <exporters>...</exporters>

```
<exporters>
  <htmlSummary>
    <outputDirectoryName>directoryOfHtmlSummary</outputDirectoryName>
    <includeHeader>>false</includeHeader>
  </htmlSummary>
  <html/>
</exporters>
```

Surefire Report

Surefire Report Configuration

In our maven archetype, Surefire reports are already enabled. Otherwise, you can enable it with the following implementation :

```
org.squashtest.ta.commons.exporter.surefire.SurefireSuiteResultExporter
```

The Surefire report uses 2 properties :

- `exportAttached` : If you want to generate the attachments in the `surefire-reports-directory` :
 - Default value : `true` (since 1.7.0).
 - Values accepted : `true/false`.
- `jenkinsAttachmentMode` : Used with the **Squash TF** server for the Jenkins JUnit attachment plugin.

Surefire report configuration sample :

```
<configuration>
...
<exporters>
  <exporter implementation="org.squashtest.tf.commons.exporter.surefire.
↪SurefireSuiteResultExporter">
    <exportAttached>true</exportAttached>
    <jenkinsAttachmentMode>true</jenkinsAttachmentMode>
  </exporter>
</exporters>
...
</configuration>
```

Since 1.7.0 version, there is an easy solution to activate the Surefire report :

```
<configuration>
...
<exporters>
  <surefire>
    <exportAttached>true</exportAttached>
    <jenkinsAttachmentMode>true</jenkinsAttachmentMode>
  </surefire>
</exporters>
...
</configuration>
```

Note: Both solutions are equivalent and works since SKF 1.7.0.

Surefire report overview

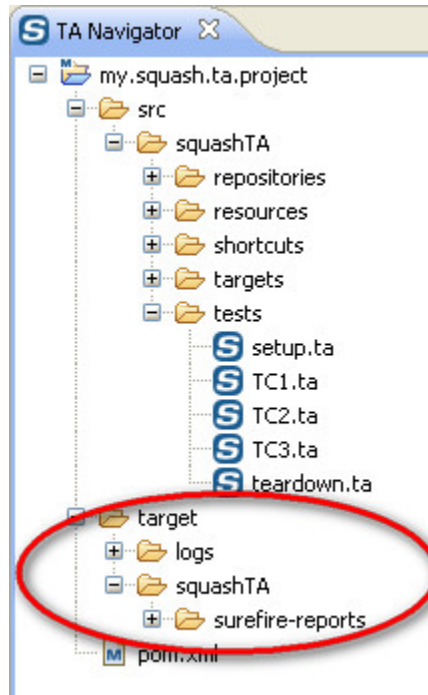
A Surefire report is an `.xml` file which contains all results of the scripts execution.

Each `<testsuite>` element represents an ecosystem and each `<testcase>` element a `.ta` script.

Surefire reports are created in directory : `my.squash.ta.project/target/squashTA/surefire-reports`

Here is an example of a Surefire report :

Three kind of results are possible :



```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite time="0.055" failures="1" tests="1" name="tests" >
  <testcase time="0.055" classname="tests" name="first-test.ta" >
    <failure message="The pattern <?xml version='1.0'> was not found in the file." type="org.squashtest.ta.framework.exception.BinaryAssertionFailedException">
      Summary
      SETUP :
      [SUCCESS] LOAD sample-dataset.xml AS xmlResource
      // here we test the macro defined in shortcuts/sample_shortcut.macro. Note that none of the instructions here are case sensitive :
      TEST :
      [SUCCESS] CONVERT xmlResource TO file(param) USING context_script_params, context_global_params AS convertedXml
      // # SUBSTITUTE KEYS IN xmlResource USING context_script_params, context_global_params AS rololo
      [SUCCESS] LOAD convertedXml AS brlbrlbrl
      [FAIL] ASSERT xmlResource IS contain THE xmlResource
      org.squashtest.ta.framework.exception.BinaryAssertionFailedException: The pattern <?xml version='1.0'> was not found in the file.
        at org.squashtest.ta.plugin.filechecker.assertions.FileContains.test(FileContains.java:60)
        at org.squashtest.ta.backbone.engine.wrapper.BinaryAssertionHandler.test(BinaryAssertionHandler.java:121)
        at org.squashtest.ta.backbone.engine.instructionrunner.DefaultBinaryAssertionRunner.doRun(DefaultBinaryAssertionRunner.java:91)
    </failure>
  </testcase>
</testsuite>

```

- **Passed** : In this case the Surefire report provides only the name and the execution time of the test case :

```
<testcase time="0.017" classname="tests.F01.ScriptsTF.Set1" name="UC01.
↳01_Success.ta" />
```

- **Failed (because of a false assertion)** : In this case the Surefire report provides the name and the execution time of the test case. It also indicates that a **‘failure’** occurred and gives the associated trace.

```
<testcase time="0.0090" classname="tests.Set1" name="Failure_in_setup.ta" >
  <failure message="The pattern rubbish was not found in the file."
    type="org.squashtest.ta.framework.exception.BinaryAssertionFailedException
↳" >
Summary
SETUP :
[SUCCESS] LOAD file.txt AS file
[FAIL] ASSERT file DOES contain WITH $(rubbish)
      [SUCCESS]      $(rubbish) --> {{__temp306}}
      [FAIL] ASSERT file DOES contain WITH {{__temp306}}

Trace here

[NOT_RUN] ASSERT file DOES contain WITH $(Hello)
      [NOT_RUN]      $(Hello) --> {{__temp152}}
      [NOT_RUN]      ASSERT file DOES contain WITH {{__temp152}}
TEST :
[NOT_RUN] ASSERT file DOES contain WITH $(Hello)
      [NOT_RUN]      $(Hello) --> {{__temp597}}
      [NOT_RUN]      ASSERT file DOES contain WITH {{__temp597}}
TEARDOWN :
[SUCCESS] ASSERT file DOES contain WITH $(Hello)
      [SUCCESS]      $(Hello) --> {{__temp529}}
      [SUCCESS]      ASSERT file DOES contain WITH {{__temp529}}
  </failure>
</testcase>
```

- **Failed (because of a technical error)** : In this case the Surefire report provides the name and the execution time of the test case. It also indicates that an **‘error’** occurs and give the associated trace.

```
<testcase time="0.013" classname="tests.Set2" name="Error_in_setup.ta" >
  <error message="ASSERT wrongId DOES contain WITH {{__temp238}}: Cannot
↳apply assertion :
  SCOPE_TEST:wrongId does not exist in this test context: you must load it
↳first."
  type="org.squashtest.tf.backbone.exception.ResourceNotFoundException" >
Summary
SETUP :
[SUCCESS] LOAD file.txt AS file
[ERROR] ASSERT wrongId DOES contain WITH $(Hello)
      [SUCCESS]      $(Hello) --> {{__temp238}}
      [ERROR] ASSERT wrongId DOES contain WITH {{__temp238}}

Trace was here

[NOT_RUN] ASSERT file DOES contain WITH $(Hello)
      [NOT_RUN]      $(Hello) --> {{__temp362}}
      [NOT_RUN]      ASSERT file DOES contain WITH {{__temp362}}
```

(continues on next page)

(continued from previous page)

```

TEST :
[NOT_RUN] ASSERT file DOES contain WITH $(Hello)
    [NOT_RUN]      $(Hello) --> {__temp345}
    [NOT_RUN]      ASSERT file DOES contain WITH {__temp345}
TEARDOWN :
[SUCCESS] ASSERT file DOES contain WITH $(Hello)
    [SUCCESS]      $(Hello) --> {__temp875}
    [SUCCESS]      ASSERT file DOES contain WITH {__temp875}
</error>
</testcase>

```

Inside the target directory, there are also attached documents produced by SKF scripts to make easier diagnosis when an error occurs (snapshots, diff-reports, detailed logs...).

5.2.4 Context Parameters

The purpose of context parameters is :

- To provide a list of key/value through *json data* (at script or at global level).

For example :

```

{
  "test" : [{
    "script" : "pathToMyScript1",
    "param" : {                                // Script context
      ↪parameters
        "my_cuf" : "value1",
        "property2" : "value2"
      }
    },
  ],
  "param" : {                                // Global context
    ↪parameters
      "property2" : "value13",
      "property6" : "value14"
    }
  }

```

- To transform the parameters as a properties resource and then use it in test script through *file to file (using param converter)* (in USING clause).
 - For script context parameters the resource is available in the test with : `context_script_params`.
 - For global context parameters the resource is available in the test with : `context_global_params`.

In the sample below, in processedCommandFile, `#{my_cuf}` is replaced by “value1” :

```

{
  "test" : [{
    "script" : "pathToMyScript1",
    "param" : {
      "my_cuf" : "value1",
      "property2" : "value2"
    }
  },
  ],
  "param" : {

```

(continues on next page)

(continued from previous page)

```
        "property2" : "value13",
        "property6" : "value14"
    }
}
```

```
DEFINE $(monShell.sh -param1=${my_cuf}) AS commandFile
CONVERT commandFile TO file (param) USING context_script_params AS processedCommandFile
CONVERT processedCommandFile TO query.shell (query) AS commandLine
EXECUTE local WITH commandLine AS result
```

`context_script_params` and `context_global_params` can be used together but be wary of multiple definitions of the same parameter.

Only the latest parameter sent will be used.

For example :

```
{
  "test" : [{
    "script" : "pathToMyscript1",
    "param" : {
      "my_cuf" : "value1",
      "property2" : "value2"
    }
  },
  {
    "param" : {
      "property2" : "value13",
      "property6" : "value14"
    }
  }
}]
```

If you send the parameters in the following order :

```
CONVERT xmlResource TO file (param) USING context_global_params, context_script_params AS convertedXml
```

Then : **property2** will be replaced by **value2**.

On the other hand, if you send them in the reverse order :

```
CONVERT xmlResource TO file (param) USING context_script_params, context_global_params AS convertedXml
```

Then : **property2** will be replaced by **value13**.

Note: The framework doesn't prevent you from defining your own SKF resource with those context names. If you ever do it, your context parameters will be overwritten (and a warn is logged).

In the sample below, `context_script_params` corresponds to `sample.properties` :

```
LOAD sample.properties AS sample.file
CONVERT sample.file TO properties (structured) AS context_script_params
```

5.2.5 SKF behaviour if an exception is raised during execution

Phases :	SKF behaviour if an exception ¹ is raised :
Ecosystem SETUP (setup.ta)	General behaviour : <ul style="list-style-type: none"> • Stops the execution of this setup.ta script. • Launches the teardown.ta script. Using VERIFY² instruction : <ul style="list-style-type: none"> • Executes the next instruction in the setup.ta script. • When setup.ta script is finished, launches test cases of the ecosystem.
Test case SETUP	General behaviour : <ul style="list-style-type: none"> • Stops the execution of this test case setup phase. • Launches the test case teardown phase. • When test case teardown phase is finished, launches next SKF script³. Using VERIFY² instruction : <ul style="list-style-type: none"> • Executes the next instruction in this test case setup phase. • When test case setup phase is finished, launches test case test phase.
Test case TEST	General behaviour : <ul style="list-style-type: none"> • Stops the execution of this test case test phase. • Launches the test case teardown phase. • When test case teardown phase is finished, launches next SKF script³. Using VERIFY² instruction : <ul style="list-style-type: none"> • Executes the next instruction in this test case test phase. • When test case test phase is finished, launches test case teardown phase.
Test case TEARDOWN	General behaviour : <ul style="list-style-type: none"> • Stops the execution of this test case teardown phase. • Launches the next SKF script (test case or teardown.ta). Using VERIFY² instruction: <ul style="list-style-type: none"> • Executes the next instruction in this test case teardown phase. • When test case teardown phase is finished, launches the next SKF script³.
Ecosystem TEARDOWN (teardown.ta)	General behaviour : <ul style="list-style-type: none"> • Stops the execution of this teardown.tf script. • Launches the next ecosystem. Using VERIFY² instruction: <ul style="list-style-type: none"> • Executes the next instruction in this teardown.ta script. • When this teardown.ta Script is finished, launches the next ecosystem.

¹ Exception could be an assertion **failure** or an unexpected **error**.

² For more information about VERIFY instruction, please check the following [page](#).

³ test case or teardown.ta

5.3 List tests in an SKF project

Contents :

- *'list' goal (squash-ta:list)*
 - *Listing test JSON report*
- *'list' goal with Metadata*
 - *Listing test JSON report with Metadata*
 - *Disable Metadata when test listing*

5.3.1 'list' goal (squash-ta:list)

The `list` goal generates a json file representing the test tree of the current project. To generate this list, run at the root of your project (where the pom.xml of your project is located) the command :

```
mvn squash-ta:list
```

The generated json file is named `testTree.json` and is created in `<root_project_path>/target/squashTA/test-tree` directory.

Listing test JSON report

```
{
  "timestamp": "2014-06-17T09:48:19.733+0000",
  "name": "tests",
  "contents": [
    {
      "name": "sample",
      "contents": [
        {
          "name": "test-OK.tf",
          "contents": null
        }
      ]
    },
    {
      "name": "sample2",
      "contents": [
        {
          "name": "placeholder.tf",
          "contents": null
        },
        {
          "name": "test-OK.tf",
```

(continues on next page)

(continued from previous page)

```

        "contents": null
    }
  ],
  {
    "name": "placeholder.tf",
    "contents": null
  },
  {
    "name": "test-KO-db-verification.tf",
    "contents": null
  },
  {
    "name": "test-KO-sahi.tf",
    "contents": null
  },
  {
    "name": "test-OK.tf",
    "contents": null
  }
]
}

```

5.3.2 'list' goal with Metadata

If there are Squash Metadata in the current test project, the goal **"list"** searches and checks if all metadata in a SKF project respect the conventions for writing and using **Squash TF** metadata. (See [Metadata section](#) for more information about Metadata syntax conventions)

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

```

[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building quan.ta.tests 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- squash-ta-maven-plugin:1.13.0-SNAPSHOT:list (default-cli) @ quan.ta.tests ---
[WARNING] Oops! Vousa utiliser JarJar version 1.13.0-SNAPSHOT nousa pas sur que vousa avoir bons r sultats !!!
[WARNING] Using non-official version 1.13.0-SNAPSHOT may bring you more fonctionalit s, but also more trouble!
se, you are welcome to it if you see fit.
[INFO] Squash TF : listing...
[INFO] Squash TF : Metadata checking...
[INFO] Squash TF : Metadata checking successfully completed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.080 s
[INFO] Finished at: 2019-09-23T11:03:40+02:00
[INFO] Final Memory: 19M/220M
[INFO] -----

```

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

```

[INFO] Scanning for projects...
[INFO] -----
[INFO] Building quan.ta.tests 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- squash-ta-maven-plugin:1.13.0-SNAPSHOT:list (default-cli) @ quan.ta.tests ---
[WARNING] Oops! Vousssa utiliser JarJar version 1.13.0-SNAPSHOT noussa pas sur que vousssa avoir bons rÚsulats !!!
[WARNING] Using non-official version 1.13.0-SNAPSHOT may bring you more functionalitites, but also more trouble!
se, you are welcome to it if you see fit.
[INFO] Squash TF : listing...
[INFO] Squash TF : Metadata checking...
[ERROR] [Mantis_1958_ftp.ta] Error while parsing Metadata at line 2: 'true* : false' - syntax convention(s) viola
[ERROR] [Mantis_1958_ftp.ta] Error while parsing Metadata at line 3: 'true : value11*' - syntax convention(s) vi
[ERROR] [Mantis_1958_ftp.ta] Error while parsing Metadata at line 4: ' : value12' - Metadata line KEY NOT found
[ERROR] [Mantis_1958_ftp.ta] Error while parsing Metadata at line 6: ' : value13' - Metadata line KEY NOT found
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 1.915 s
[INFO] Finished at: 2019-09-23T11:14:25+02:00
[INFO] Final Memory: 17M/222M
[INFO] -----
[ERROR] Failed to execute goal org.squashtest.ta:squash-ta-maven-plugin:1.13.0-SNAPSHOT:list (default-cli) on pro
an.ta.tests: Execution default-cli of goal org.squashtest.ta:squash-ta-maven-plugin:1.13.0-SNAPSHOT:list failed:
Test Metadata checking failed.
[ERROR] Please refer to Squash TF wiki/doc for further details.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/PluginExecutionException

```

Listing test JSON report with Metadata

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

```

{
  "timestamp": "2014-06-17T09:48:19.733+0000",
  "name": "tests",
  "contents": [
    {
      "name": "sample",
      "metadata" : {},
      "contents": [
        {
          "name": "test-OK.ta",
          "metadata" : {
            "linked-TC": ["guid-1", "guid-2"],
            "key2": null,
            "key3": ["value"]
          },
          "contents": null
        }
      ]
    },
    {
      "name": "test-KO.ta",

```

(continues on next page)

(continued from previous page)

```
    "metadata" : {},
    "contents": null
  }
]
```

Disable Metadata when test listing

If there are Metadata in your project but you want to ignore them during the project test listing, then insert *tf.disableMetadata* property after the goal “list”

```
mvn squash-ta:list -Dtf.disableMetadata=true
```

The generated report (JSON file) will then NO LONGER contain the metadata.

```
{
  "timestamp": "2014-06-17T09:48:19.733+0000",
  "name": "tests",
  "contents": [
    {
      "name": "sample",
      "contents": [
        {
          "name": "test-OK.ta",
          "contents": null
        }
      ]
    },
    {
      "name": "test-KO.ta",
      "contents": null
    }
  ]
}
```

Note: SKF has also a deprecated `test-list` goal. It generates the test list in the console / log and through the exporters configured in `pom.xml` (html, surefire)

5.4 Check TF metadata in project

Contents:

- ‘check-metadata’ goal (*squash-ta:check-metadata*)
 - ‘check-metadata’ goal with Unicity checking
 - ‘check-metadata’ goal with Unicity checking for specific Keys

5.4.1 ‘check-metadata’ goal (squash-ta:check-metadata)

As goal “list”, the goal “**check-metadata**” searches and checks if all metadata in a SKF project respect the conventions for writing and using **Squash TF** metadata (see [Metadata section](#) for more information about Metadata syntax conventions).

```
mvn squash-ta:check-metadata
```

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

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building quan.ta.tests 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- squash-ta-maven-plugin:1.13.0-SNAPSHOT:check-metadata (default-cli) @ quan.ta.tests ---
[WARNING] Oops! Vousssa utiliser JarJar version 1.13.0-SNAPSHOT noussa pas sur que vousssa avoir bons r sultats !!!
[WARNING] Using non-official version 1.13.0-SNAPSHOT may bring you more functionalit s, but also more trouble! Of course,
you are welcome to it if you see fit.
[INFO] Squash TF : checking metadata...
[WARNING] [simple.ta] Warning while parsing Metadata VALUE: 'valu-E_1/here.a' at line 13 - a same VALUE is already assigned
to the current Metadata KEY: 'Metadata0-_' at line 9.
[WARNING] [simple2.ta] Warning while parsing Metadata VALUE: 'valu-E_1/here.a' at line 13 - a same VALUE is already assigne
d to the current Metadata KEY: 'Metadata01-_' at line 9.
[INFO] Squash TF : Metadata checking successfully completed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.845 s
[INFO] Finished at: 2019-09-23T11:59:29+02:00
[INFO] Final Memory: 17M/220M
[INFO] -----
```

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

```
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 7: '(third_key) : third_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 8: '[fourth_key] : fourth_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 9: '{fifth_key} : fifth_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 10: '<sixth_key> : sixth_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 11: 'séventh~këy& : seventh_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 12: '"octo?"key@; : eight' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 13: 'éëâëiïë : accent_only_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 14: '%üä"µkeyëüikey : random_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 15: 'a/// : slash_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 16: 'a\\ : anti-slash_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 18: 'first_key : flr$t_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 19: 'second_key : $€cond_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 20: 'third_key : (third_value)' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 21: 'fourth_key : [fourth_value]' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 22: 'fifth_key : {fifth_value}' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 23: 'sixth_key : <sixth_value>' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 24: 'seventh_key : séventh~valuë&' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 25: 'eight : "octo?"key@;' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 26: 'accent_only_key : éëâëiïë' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 27: 'random_key : %üä"µkeyëüi_value' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 28: 'anti-slash-key : a\\' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 30: 'multiline€ : value_one' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 31: ' : value_two' - Metadata line KEY NOT found
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 32: ' : value_three' - Metadata line KEY NOT found
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 34: 'multiline : v@luë_onë' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 35: ' : v@luë_two;' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 36: ' : value_three2' - Metadata line KEY NOT found
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 37: '&&&' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 38: 'éëé' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 39: 'èèè' - syntax convention(s) violated
[ERROR] [KO\unaccepted-char.ta] Error while parsing Metadata at line 40: 'ôôô' - syntax convention(s) violated
[ERROR] [KO\unique_key.ta] Error while parsing Metadata KEY: 'unique' at line 5 - a same KEY is already existed in Test file.
[ERROR] [KO\unique_key.ta] Error while parsing Metadata KEY: 'UnIQue' at line 7 - a same KEY is already existed in Test file.
[ERROR] [KO\EcosystemKO\test-one.ta] Error while parsing Metadata at line 5: 'first key : first value' - syntax convention(s) violated
[ERROR] [KO\EcosystemKO\test-one.ta] Error while parsing Metadata at line 8: ' : second' - Metadata line KEY NOT found
[ERROR] [KO\EcosystemKO\test-one.ta] Error while parsing Metadata at line 10: ' : third' - Metadata line KEY NOT found
[ERROR] [KO\EcosystemKO\test-two.ta] Error while parsing Metadata at line 4: 'solo_këi' - syntax convention(s) violated
[ERROR] [KO\EcosystemKO\test-two.ta] Error while parsing Metadata at line 5: 'fir$tr_këy : fir$tr_v@luë' - syntax convention(s) violated
[ERROR] [KO\EcosystemKO\test-two.ta] Error while parsing Metadata at line 6: 'multiline : fir$tr_value' - syntax convention(s) violated
[ERROR] [KO\EcosystemKO\test-two.ta] Error while parsing Metadata at line 7: ' : s€cond_value' - syntax convention(s) violated
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 1.351 s
[INFO] Finished at: 2019-07-10T17:26:18+02:00
[INFO] Final Memory: 15M/262M
[INFO] -----
[ERROR] Failed to execute goal org.squashtest.ta:squash-ta-maven-plugin:1.13.0.IT3:check-metadata (default-cli) on project metadata-ko: Execution default-cli of
```

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

```
[WARN] Using non-official version 1.13.0.IT3 may bring you more functionalities, but also more trouble! Of course, you are welcome to it if you see fit.
[INFO] Squash TF : checking metadata...
[WARN] [OK\multiline_duplicate.ta] Warning while parsing Metadata VALUE: 'same' at line 6 - a same VALUE is already assigned to the current Metadata KEY: 'duplicate' at line 5.
[WARN] [OK\multiline_duplicate.ta] Warning while parsing Metadata VALUE: 'same' at line 7 - a same VALUE is already assigned to the current Metadata KEY: 'duplicate' at line 5.
[WARN] [OK\multiline_duplicate.ta] Warning while parsing Metadata VALUE: 'same' at line 8 - a same VALUE is already assigned to the current Metadata KEY: 'duplicate' at line 5.
[WARN] [OK\multiline_duplicate.ta] Warning while parsing Metadata VALUE: '1' at line 10 - a same VALUE is already assigned to the current Metadata KEY: 'duplicate_two' at line 9.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.330 s
[INFO] Finished at: 2019-07-10T17:32:23+02:00
[INFO] Final Memory: 14M/197M
[INFO] -----
```

‘check-metadata’ goal with Unicity checking

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

```
mvn squash-ta:check-metadata -Dtf.metadata.check=[valueUnicity]
```

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

```
[WARN] Using non-official version 1.13.0.IT3 may bring you more functionalities, but also more trouble! Of course, you are welcome to it if you see fit.
[INFO] Listing Junit tests in main bundle "classes" and test bundle "test-classes".
[INFO] Squash TF: Metadata checking...
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata KEY syntax error: 'ÚÚ' 'p'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata VALUE syntax error: 'ÚÚÚÚ' 'p'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata KEY syntax error: 'ÚÚÚÚ' 'p'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata KEY syntax error: 'ÚÚ' 'p'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata VALUE syntax error: 'one'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata VALUE syntax error: 'ÚÚtwo'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.accent] Metadata VALUE syntax error: 'ÚÚÚthree'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata KEY syntax error: 'first K-E.Y&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata VALUE syntax error: 'vv&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata KEY syntax error: '22&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata KEY syntax error: 'multi&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata VALUE syntax error: 'first&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata VALUE syntax error: 'second&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.specialChar] Metadata VALUE syntax error: 'third&~#{}+=| $úñ*%e!?'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.sameKeyMultipleAnnot] The current test method contains duplicate Metadata KEY: 'same'
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.noValue] Metadata VALUE syntax error: ''
[ERROR] [maven.main.bundle:junit.metadata.MetadataTestKO.noValue] Metadata VALUE syntax error: ''
[ERROR] For Metadata KEY: {unicity}, VALUE: {bla} has been found in methods:
maven.main.bundle:junit.metadata.MetadataTestKO.unicityError1
maven.main.bundle:junit.metadata.MetadataTestKO.unicityError2
maven.main.bundle:junit.metadata.MetadataTestKO.otherClass.unicityErrorOtherFile
[ERROR] For Metadata KEY: {unicity_multiline}, VALUE: {same} has been found in methods:
maven.main.bundle:junit.metadata.MetadataTestKO.unicityErrorMultiline2
maven.main.bundle:junit.metadata.MetadataTestKO.unicityErrorMultiline1
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
```

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

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

```
mvn squash-ta:check-metadata -Dtf.metadata.check=[valueUnicity] -Dtf.metadata.check.
↳keys=[xxx,yyy,zzz]
```

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

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

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

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

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

```
[ERROR] [KO\EcosystemKO\test-two.ta] Error while parsing Metadata at line 7: 'second_value' - syntax convention(s) violated
[DEBUG] PHASE definition found. Metadata parsing (if any) is finished.
[DEBUG] Metadata section successfully created
[DEBUG] PHASE definition found. Metadata parsing (if any) is finished.
[ERROR] For Metadata KEY: {unicity}, VALUE: {same_value} has been found in tests:
KO\EcosystemUnicity\test-one.ta
KO\unicity-check-other2.ta
KO\unicity-check.ta
KO\unicity-check-other.ta
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
```

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

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building quan.ta.tests 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- squash-ta-maven-plugin:1.13.0-SNAPSHOT:check-metadata (default-cli) @ quan.ta.tests ---
[WARNING] Oops! Vousssa utiliser JarJar version 1.13.0-SNAPSHOT noussa pas sur que vousssa avoir bons r sultats !!!
[WARNING] Using non-official version 1.13.0-SNAPSHOT may bring you more functionalitites, but also more trouble! Of course,
you are welcome to it if you see fit.
[INFO] Squash TF : checking metadata...
[INFO] Squash TF : checking metadata with value unicity...
[WARNING] [simple.ta] Warning while parsing Metadata VALUE: 'valu-E_1/here.a' at line 13 - a same VALUE is already assigned
to the current Metadata KEY: 'Metadata _.' at line 9.
[WARNING] [simple2.ta] Warning while parsing Metadata VALUE: 'valu-E_1/here.a' at line 13 - a same VALUE is already assigne
d to the current Metadata KEY: 'Metadata 1-_' at line 9.
[INFO] Metadata value unicity analysis for keys: {KEY3., key1, true} in test project done successfully.
[WARNING] Some metadata keys not found in any test: {KEY3., key1}.
[INFO] Squash TF : Metadata checking successfully completed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.846 s
[INFO] Finished at: 2019-09-23T12:08:07+02:00
[INFO] Final Memory: 17M/222M
[INFO] -----
```

SKF is a maven plugin. So an SKF automation project is a maven project. You'll need **maven** and a **jdk** installed on your system.

To execute your tests, list your tests, etc ... , you will have to use the maven goals we create to handle them. Currently 3 goals are defined :

- `run` : this goal handles the execution of your tests.
- `list` : this goal handles the listing of your tests.
- `check-metadata` : this goal handles some checks on TF metadata in SKF test scripts.

6.1 Commons component plugin

6.1.1 Commons component plugin - Repositories

Contents :

- *Classpath*
- *URL*

The Repositories give you access to your test data from your script using the LOAD instruction. The present subsection will teach you how to configure and use them.

IMPORTANT :

By default you can always access the Resources in your ‘resources’ directory (a default Repository pointing to that directory always exists). So if all you need is this unique directory you don’t need to declare any Repository.

Classpath

Category-name : *classpath*

What ?

Retrieves files present in the classpath of SKF and is helpful to load Resources embedded in other SKF plugins. If you don't know what it means, then you probably don't need it.

Configuration : A simple .properties file dropped in the 'repositories' directory of your test project. It must contain EXACTLY '**squashtest.ta.classpath**', with any values you like (it doesn't matter). Any other properties present in this file will disqualify it.

Example of valid configuration file :

```
squashtest.ta.classpath=whatever
```

URL

Category-name : *url*

What ?

A generic repository for **files** (and **ONLY FILES**) accessible using an URL. An ideal choice for http or file-system based repositories. Technical note : the supported protocols depends on the protocol handlers available in the jvm at runtime, so adding your own handlers will naturally increase the range of addresses accessible from SKF (For more informations, please consult the [Java Documentation](#)).

Configuration : A simple .properties file dropped in the 'repositories' directory of your test project. The file must contain AT LEAST '**squashtest.ta.url.base**'.

Available properties are :

- **squashtest.ta.url.base** : The base url representing your repository.
- **squashtest.ta.url.useCache** : Whether to use a cache, to speed up future references to recurrent resources.
- **squashtest.ta.url.login** : Specifies a username for http authentication (special, see below).
- **squashtest.ta.url.password** : Specifies the password for the username above (special, see below).

HTTP Authentication : The login and password options above only hold for http authentication (protocols listed [here](#)). It may also fail if the http server implements a custom protocol.

Note: You can also use the URL repository as a cheap replacement for the FTP repository. You must then format the url with respect to the standard : [ftp://user:password@host:port/path/to/basedir](#). In that case the login and password properties are useless (since they're inlined in the url itself).

Example of valid configuration file pointing to a .txt file :

```
# note that the space in 'Program Files' is url encoded.
squashtest.ta.url.base = file:///C:/Program%20Files/fileTest.txt
squashtest.ta.ftp.useCache = false
```

6.1.2 Commons component plugin - Target

Remark

All `.properties` file corresponding to a target file must contain a shebang on the first line (example for a database target the shebang is : `#!db`).

http

Category-name : *http*

What ?

The http target represents a basic http endpoint.

Configuration : A simple `.properties` file dropped in the `'targets'` directory of your test project. The file must contain EXACTLY `'squashtest.tf.endpoint.url'`. It must also include a shebang on the very first line : `'#!http'`.

Example of valid configuration file :

```
#!http
squashtest.ta.http.endpoint.url = http://www.google.com/pacman/
```

6.1.3 Commons component plugin - Resources

Contents :

- *file*
- *bundle*
- *directory*
- *process*
- *properties*
- *script.java*
- *xml*

This subsection introduces you to the Resources shipped with the commons-components basic plugin. The most nitpicking of you will notice that the `file` resource category is part of the core of SKF and doesn't need the plugin to be available. Nevertheless it is stated here for convenience.

Most of the resources here will start their careers as `file` resource, then will be converted to their type using converters. Some of them can also be produced as the result of a command (for instance `result.sql`).

Since most of the resources will be materialized as a `file` in your test project, you might be wondering what `file` corresponds to complex resources (`conf.dbunit` for instance). You will NOT find that information here, because there are potentially unlimited ways to create that resource, not just one file format. In other words, one could imagine that a given resource could be constructed from other resources involving completely different material. A good example is the `dataset.dbunit` resource : the **Database Plugin** provides three ways of creating one of these.

In fact, the question of how to obtain a resource, given another resource, is typically the job of the converters. If you're interested in what you should supply as inputs to the system in order to produce the resources you want, you should check the '**Converters**' section, located in the '**Advanced Users**' section of each plugin. In the case of our `dataset.dbunit` example, the answer you look for is in [this section](#).

file

Category-name : *file*

What ?

file is a resource type representing a plain file or a directory, no assumption made on the content of either of them.

bundle

Category-name : *bundle*

What ?

bundle is a resource type representing a set of files. Basically it is a directory containing more directories or files. The root directory is called the 'base'. However it also have additional attributes. Those attributes give clues on what is the relationships between the files it embeds.

The followig attribute is available :

- `mainpath` : denotes which file is considered to be the main file, and the other files represent its dependencies. The `mainpath` is the path relative to the base of the bundle. When used, the context will decide what to do with that main file, typically when using commands.
-

directory

Category-name : *directory*

What ?

directory is a resource type that represents a whole directory.

process

Category-name : *process*

@See : Since **Squash TA 1.7.0**, this resource moved to the *Local Process Plugin*.

properties

Category-name : *properties*

What ?

properties is a resource type that represents properties, in other words a set of pairs of <key - value >.

script.java

Category-name : *script.java*

What ?

This resource encapsulates a java code bundle, including resources and compiled java classes.

xml

Category-name : *xml*

What ?

xml is a resource type that represents a file or a folder, like *file*. The difference is that the content is trusted to be of XML nature.

6.1.4 Commons component plugin - Macros

Commons component plugin - Macros - Logs

Contents :

- *# DEBUG \$(message)*
- *# ERROR \$(message)*
- *# INFO \$(message)*
- *# WARN \$(message)*
- *# LOG FILE CONTENT FROM {file} WITH LEVEL {level}*
- *# LOG FILE CONTENT FROM {content1} , {otherContent} WITH LEVEL {level}*

DEBUG \$(message)

What ?

This macro allows to write a message in the console with the DEBUG status.

Underlying instruction :

```
EXECUTE log WITH $({message}) USING $(logLevel:DEBUG) AS whatever
```

> Input :

- message : The message to display.

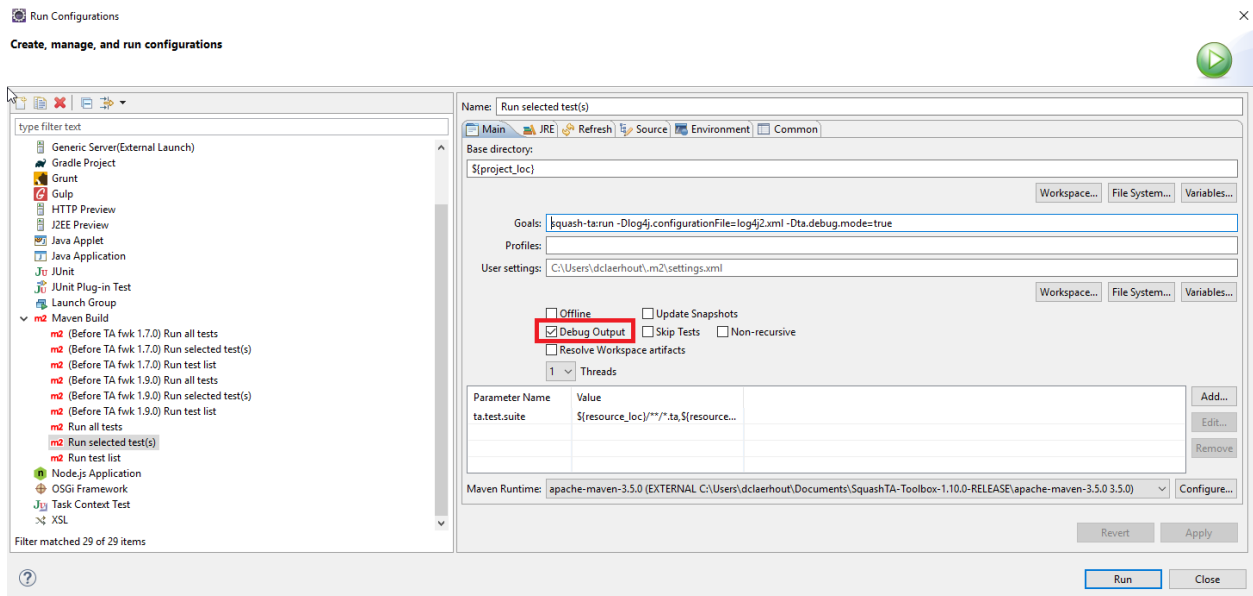
Example :

```
# DEBUG $(This is a debug message logged with a macro)
```

SKF Script :

```
S log_debug.ta ⌘
1 TEST :
2
3 # DEBUG $(This is a debug message logged with a macro)
```

To be able to see the message in the console, you need to activate maven debug output.



Console output :

```
[DEBUG] Creating instance of bean 'writeLogCommand'
[DEBUG] Finished creating instance of bean 'writeLogCommand'
[DEBUG] $(This is a debug message logged with a macro)

[DEBUG] Instruction execution complete.
[DEBUG] Test execution complete.
[DEBUG] Test context cleanup complete.
```

ERROR \$(message)

What ?

This macro allows to write a message in the console with the ERROR status.

Underlying instruction :

```
EXECUTE log WITH ${message}) USING $(logLevel:ERROR) AS whatever
```

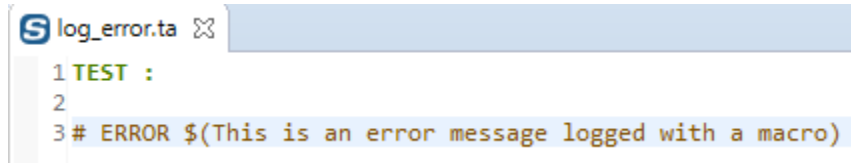
> Input :

- message : The message to display.

Example :

```
# ERROR $(This is an error message logged with a macro)
```

SKF Script :



```
S log_error.ta ⌵
1 TEST :
2
3 # ERROR $(This is an error message logged with a macro)
```

Console output :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test log_error.ta
[ERROR] $(This is an error message logged with a macro)
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCCLAER~1\AppData\Local\Temp\Squash_TA\20190909_111552_0541034169241032793332
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.269 s
[INFO] Finished at: 2019-09-09T11:15:54+02:00
[INFO] Final Memory: 20M/253M
[INFO] -----
```

INFO \$(message)

What ?

This macro allows to write a message in the console with the INFO status.

Underlying instruction :

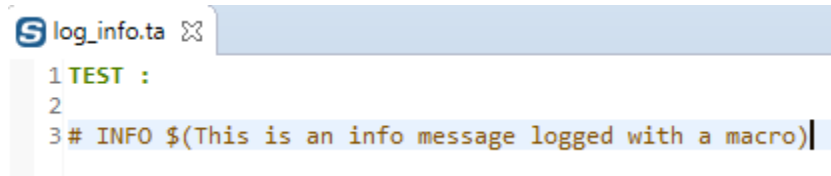

```
EXECUTE log WITH ${message} USING $(logLevel:INFO) AS whatever
```

> Input :

- message : The message to display.

Example :

```
# INFO $(This is an info message logged with a macro)
```

SKF Script :


```
log_info.ta
1 TEST :
2
3 # INFO $(This is an info message logged with a macro)|
```

Console output :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test log_info.ta
[INFO] $(This is an info message logged with a macro)
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190909_111012_5667189481643301183217
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 21.845 s
[INFO] Finished at: 2019-09-09T11:10:15+02:00
[INFO] Final Memory: 20M/248M
[INFO] -----
```

WARN \$(message)**What ?**

This macro allows to write a message in the console with the WARN status.

Underlying instruction :

```
EXECUTE log WITH ${message} USING $(logLevel:WARN) AS whatever
```

> Input :

- message : The message to display.

Example :

```
# WARN $(This is a warn message logged with a macro)
```

SKF Script :

```
S log_warn.ta ⌘  
1 TEST :  
2  
3 # WARN $(This is a warn message logged with a macro)
```

Console output :

```
[INFO] Beginning execution of ecosystem tests  
[INFO] Beginning execution of test log_warn.ta  
[WARN] $(This is a warn message logged with a macro)  
[INFO] Exporting results  
[INFO] Cleaning resources  
[INFO] Squash TF : build complete.  
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190909_111441_5443301856281442717208  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 4.347 s  
[INFO] Finished at: 2019-09-09T11:14:44+02:00  
[INFO] Final Memory: 21M/256M  
[INFO] -----
```

LOG FILE CONTENT FROM {file} WITH LEVEL {level}

What ?

This macro allows to write the content of a file in the console with the status of your choice (DEBUG,INFO,WARN,ERROR).

Underlying instruction :

```
LOAD {file} AS __target{%%r1}  
EXECUTE log WITH __target{%%r1} USING $(logLevel:{level},multiline:yes) AS $()
```

> Input :

- `file` : The file which you want to display the content.

Example :

```
# LOG FILE CONTENT FROM folder/example.txt WITH LEVEL INFO
```

File to log :

SKF Script :

Console output :

```
example.txt  ✖
1 This is my first content.
```

```
S log_file.ta  ✖
1 TEST :
2
3 # LOG FILE CONTENT FROM myResources/example.txt WITH LEVEL INFO
```

LOG FILE CONTENT FROM {content1} , {otherContent} WITH LEVEL {level}

What ?

This macro allows to write the content of multiple files in the console with the status of your choice (DEBUG,INFO,WARN,ERROR).

Underlying instruction :

```
# LOG FILE CONTENT FROM {content1} WITH LEVEL {level}
# LOG FILE CONTENT FROM {otherContent} WITH LEVEL {level}
```

> Input :

- content1 : The selected file which you want to display the content.
- otherContent : Another file which you want to display the content.

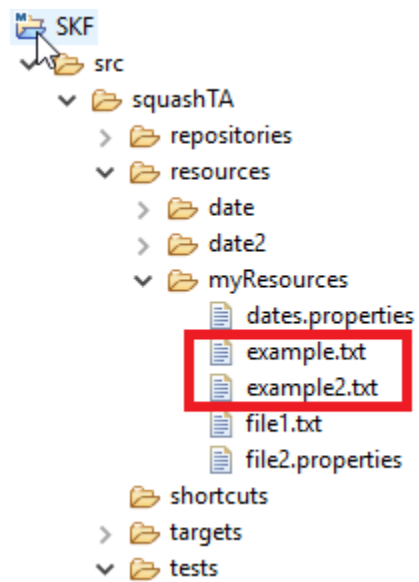
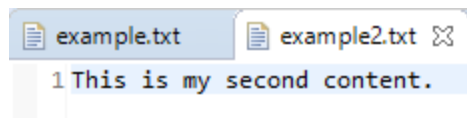
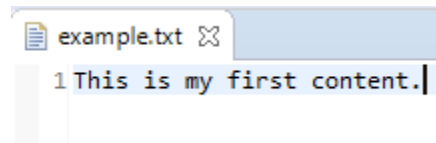
Example :

```
# LOG FILE CONTENT FROM folder/example.txt , folder/example2.txt WITH LEVEL WARN
```

First file to log :

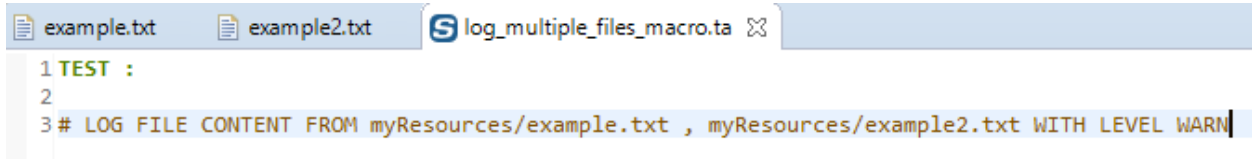
Second file to log :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test log_file.ta
[INFO] This if my first content.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190909_113617_0703515483952569785428
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.337 s
[INFO] Finished at: 2019-09-09T11:36:19+02:00
[INFO] Final Memory: 21M/258M
[INFO] -----
```



Resource folder where files to log are stored :

SKF Script :



```
1 TEST :
2
3 # LOG FILE CONTENT FROM myResources/example.txt , myResources/example2.txt WITH LEVEL WARN
```

Console output :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test log_multiple_files_macro.ta
[WARN] This is my first content.
[WARN] This is my second content.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~2\AppData\Local\Temp\Squash_TA\20191011_100434_7644545534244383950581
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 26.618 s
[INFO] Finished at: 2019-10-11T10:04:37+02:00
[INFO] Final Memory: 23M/198M
[INFO] -----
```

Commons component plugin - Macros - Pause

Contents :

- *# PAUSE {time_in_ms} MILLISECONDS*
- *# PAUSE {time_in_s} SECONDS*

PAUSE {time_in_ms} MILLISECONDS

What ?

This macro makes the test execution sleep for a given time (expressed in milliseconds).

Underlying instruction :

```
EXECUTE pause WITH $({time_in_ms}) AS $()
```

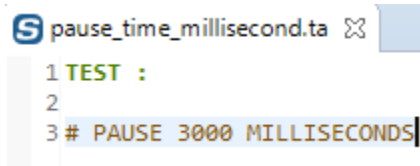
> Input :

- `time_in_ms` : Time in milliseconds.

Example :

```
# PAUSE 3000 MILLISECONDS
```

SKF script :



```
S pause_time_millisecond.ta ✕  
1 TEST :  
2  
3 # PAUSE 3000 MILLISECONDS
```

Console Output :

```
[INFO] Beginning execution of ecosystem tests  
[INFO] Beginning execution of test pause_time_millisecond.ta ← The execution of script will stop during the selected amount of time in milliseconds.  
[INFO] Exporting results  
[INFO] Cleaning resources  
[INFO] Squash TF : build complete.  
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190909_114659_3018240367963645178106  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 7.362 s  
[INFO] Finished at: 2019-09-09T11:47:04+02:00  
[INFO] Final Memory: 21M/255M  
[INFO] -----
```

PAUSE {time_in_s} SECONDS

What ?

This macro makes the test execution sleep for a given time (expressed in seconds).

Underlying instruction :

```
EXECUTE pause WITH $({time_in_s}000) AS $()
```

> Input :

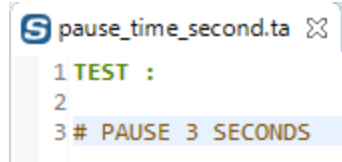
- `time_in_s` : Time in seconds.

Example :

```
# PAUSE 3 SECONDS
```

SKF script :

Console Output :



```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test pause_time_second.ta
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCIAER~1\AppData\Local\Temp\Squash_TA\20190909_113959_1754011901709217749500
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.521 s
[INFO] Finished at: 2019-09-09T11:40:04+02:00
[INFO] Final Memory: 19M/294M
[INFO] -----
```

Commons component plugin - Macros - Substitute dates

Contents :

- *# SUBSTITUTE DATES IN {files} AS {processed_files}*
- *# SUBSTITUTE DATES IN {files} FOR FILES WHOSE NAMES MATCHING {regex} AS {processed_files}*
- *# SUBSTITUTE DATES IN {files} FOR FILES WHOSE NAMES NOT MATCHING {regex} AS {processed_files}*
- *# SUBSTITUTE DATES IN {files} USING {dates} AS {processed_files}*
- *# SUBSTITUTE DATES IN {files} USING {dates} FOR FILES WHOSE NAMES MATCHING {regex} AS {processed_files}*
- *# SUBSTITUTE DATES IN {files} USING {dates} FOR FILES WHOSE NAMES NOT MATCHING {regex} AS {processed_files}*

SUBSTITUTE DATES IN {files} AS {processed_files}

What ?

This macro allows to replace dates in a bundle of files. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativedate) AS {processed_files}
```

> Input :

- `{files}` : The bundle of files where you want to apply the substitution.

> Output :

- {processed_files} : The bundle of files that have been processed.

Example :

```
# SUBSTITUTE DATES IN resources_folder AS result_bundle
```

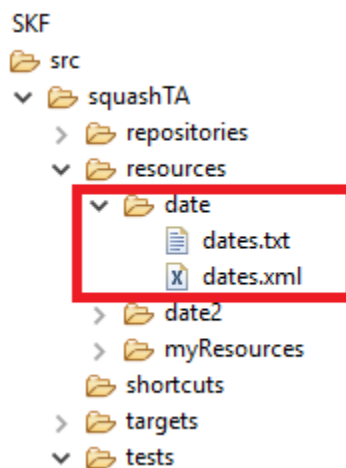
First file to process :

```
dates.txt
1 ${now().addDay(1).format(ddMMyyyy)}
2 ${now().addMonth(2).format(ddMMyyyy)}
3 ${now().addDay(2).addYear(-1).format(yyyyMMdd)}
4 ${now().addDay(-1).addMonth(2).addYear(2).format(dd/MM/yyyy)}|
```

Second file to process :

```
dates.txt  dates.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <webcal_user date="${now().addDay(1).format(ddMMyyyy)}"/>
5   <webcal_user date="${now().addMonth(2).format(ddMMyyyy)}"/>
6   <webcal_user date="${now().addDay(2).addYear(-1).format(yyyyMMdd)}"/>
7   <webcal_user date="${now().addDay(-1).addMonth(2).addYear(2).format(dd/MM/yyyy)}"/>
8 </dataset>
9
```

The folder containing files to process which corresponds to {files} :

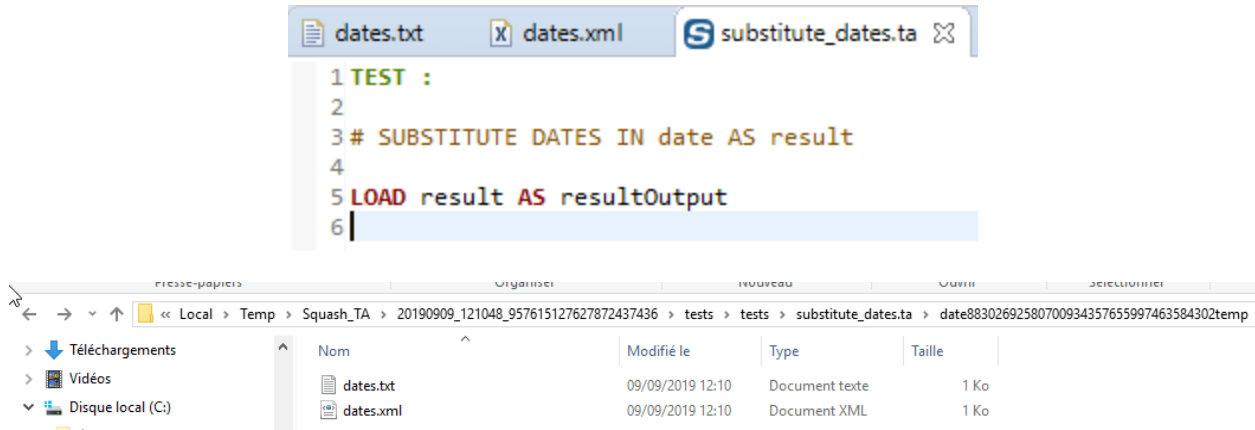


SKF script :

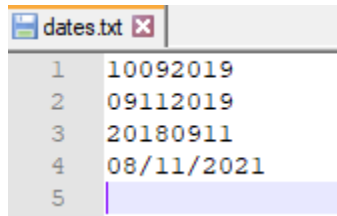
In order to check that the macro worked properly, we added in this example the instruction “**LOAD result AS resultOutput**” to be able to see the result output.

You can access to the result output in the following folder which contains temporary files :

```
C:\Users\*user name*\AppData\Local\Temp\Squash_TA\20190909_121048_957615127627872437436\tests\tests\substitute_da
```

Result output for first file (script executed on 09/09/2019) :



Result output for second file :

SUBSTITUTE DATES IN {files} FOR FILES WHOSE NAMES MATCHING {regex} AS {processed_files}

What ?

This macro allows to replace dates in a bundle of files whose names match a specific regular expression. For more information about the formulas to use in order to replace dates, please check this [page](#).

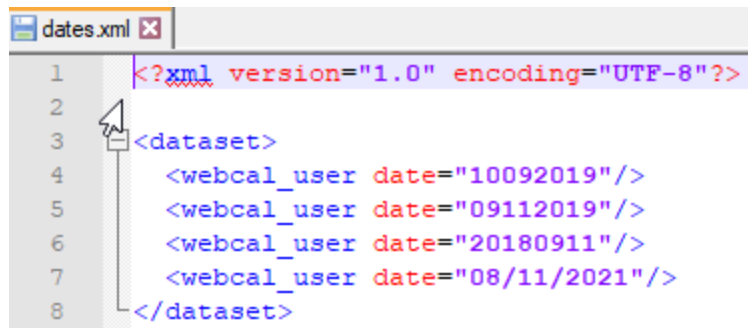
Underlying instruction :

```
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativedate) USING $(squashtest.ta.param.
  ↳include:{regex}) AS {processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {regex} : The regular expression used to filter the files in the bundle.

> Output :



```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <webcal_user date="10092019"/>
5   <webcal_user date="09112019"/>
6   <webcal_user date="20180911"/>
7   <webcal_user date="08/11/2021"/>
8 </dataset>

```

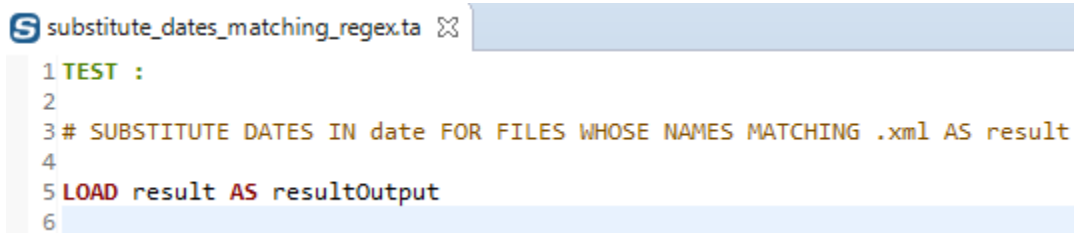
- {processed_files} : The bundle of filtered files that have been processed.

Example :

```
# SUBSTITUTE DATES IN resources_folder FOR FILES WHOSE NAMES MATCHING .xml AS
result_bundle
```

This example is based on the previous one. For more details, please check [here](#).

SKF script :



```

S substitute_dates_matching_regex.ta
1 TEST :
2
3 # SUBSTITUTE DATES IN date FOR FILES WHOSE NAMES MATCHING .xml AS result
4
5 LOAD result AS resultOutput
6

```

Console output :

```

[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test substitute_dates_matching_regex.ta
[INFO] the file dates.txt was properly excluded.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCIAER~1\AppData\Local\Temp\Squash_TA\20190910_160849_1352761239816805003440
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 38.985 s
[INFO] Finished at: 2019-09-10T16:08:52+02:00
[INFO] Final Memory: 23M/196M
[INFO] -----

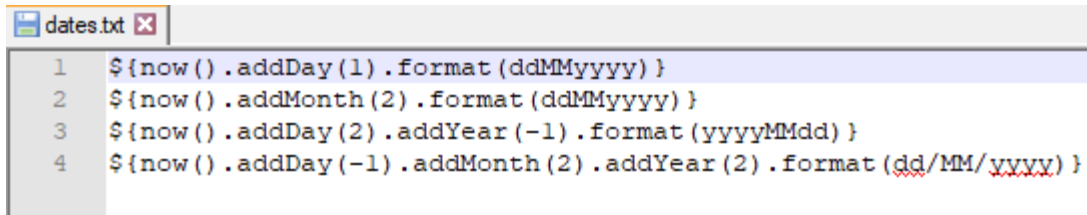
```

The .txt file which does not match the selected regex is properly excluded.

Result output for first file :

Result output for second file :

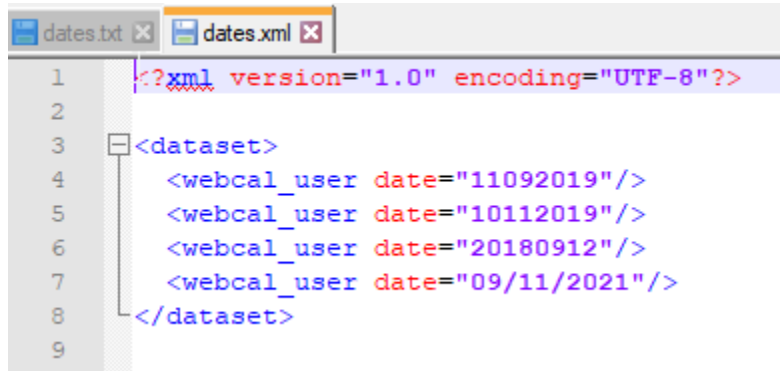
The .txt file is not processed whereas the .xml is.



```

1  ${now().addDay(1).format(ddMMyyyy)}
2  ${now().addMonth(2).format(ddMMyyyy)}
3  ${now().addDay(2).addYear(-1).format(yyyyMMdd)}
4  ${now().addDay(-1).addMonth(2).addYear(2).format(dd/MM/yyyy)}

```



```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <dataset>
4    <webcal_user date="11092019"/>
5    <webcal_user date="10112019"/>
6    <webcal_user date="20180912"/>
7    <webcal_user date="09/11/2021"/>
8  </dataset>
9

```

SUBSTITUTE DATES IN {files} FOR FILES WHOSE NAMES NOT MATCHING {regex} AS {processed_files}

What ?

This macro allows to replace dates in a bundle of files whose names don't match a specific regular expression. For more information about the formulas to use in order to replace dates, please check [this page](#).

Underlying instruction :

```

LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativedate) USING $(squashtest.ta.param.
  ↪exclude:{regex}) AS {processed_files}

```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {regex} : The regular expression used to filter the files in the bundle.

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Example :

```

# SUBSTITUTE DATES IN resources_folder FOR FILES WHOSE NAMES NOT MATCHING .xml
AS result_bundle

```

This example is based on the first one. For more details, please check [here](#).

SKF script :

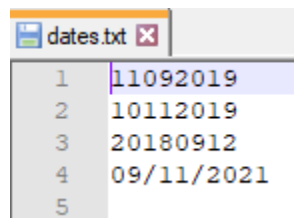
Console output :

```
S substitute_dates_not_matching_regex.ta ⌘
1 TEST :
2
3 # SUBSTITUTE DATES IN date FOR FILES WHOSE NAMES NOT MATCHING .xml AS result
4
5 LOAD result AS resultOutput
6
```

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test substitute_dates_not_matching_regex.ta
[INFO] the file dates.xml was properly excluded.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190910_163924_7894811148128433722018
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.453 s
[INFO] Finished at: 2019-09-10T16:39:27+02:00
[INFO] Final Memory: 21M/254M
[INFO] -----
```

The **.xml** file which match the selected regex is properly excluded.

Result output for first file :



Line	Date
1	11092019
2	10112019
3	20180912
4	09/11/2021
5	

Result output for second file :

The **.xml** file is not processed whereas the **.txt** is.

SUBSTITUTE DATES IN {files} USING {dates} AS {processed_files}

What ?

This macro allows to replace dates in a bundle of files with dates you have specified. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```
LOAD {dates} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳ properties
```

(continues on next page)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <webcal_user date="${now().addDay(1).format('ddMMyyyy')}" />
5   <webcal_user date="${now().addMonth(2).format('ddMMyyyy')}" />
6   <webcal_user date="${now().addDay(2).addYear(-1).format('yyyyMMdd')}" />
7   <webcal_user date="${now().addDay(-1).addMonth(2).addYear(2).format('dd/MM/yyyy')}" />
8 </dataset>

```

(continued from previous page)

```

LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativedate) USING placeholder{%%rand3}.
↳ properties AS {processed_files}

```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {dates} : It can either be a path to a properties file or an inline command with keys and values for the dates you want to specify.

Example of inline command : \$(date1=01/01/2001 \n date2=31/12/2018).

Example of date in file to process : \${date(date1,dd/MM/yyyy).format('dd-MM-yyyy')}

> Output :

- {processed_files} : The bundle of files that have been processed.

Examples :

```
# SUBSTITUTE DATES IN resources_folder USING resources_folder/dates.properties AS result_bundle
```

Or

```
# SUBSTITUTE DATES IN resources_folder USING $(date1=01/01/2001 \n date2=31/12/2018) AS result_bundle
```

This example is based on the first one. For more details, please check [here](#).

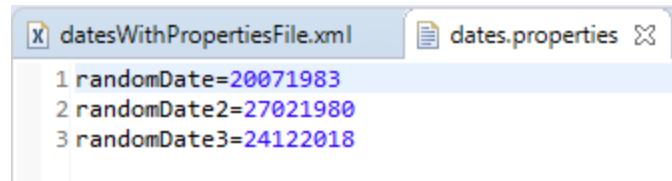
File to process :

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <webcal_user date="${date(randomDate,ddMMyyyy).format('dd/MM/yyyy')}" />
5   <webcal_user date="${date(randomDate2,ddMMyyyy).format('dd-MM-yyyy')}" />
6   <webcal_user date="${date(randomDate3,ddMMyyyy).addDay(1).format('dd-MM-YYYY')}" />
7 </dataset>
8

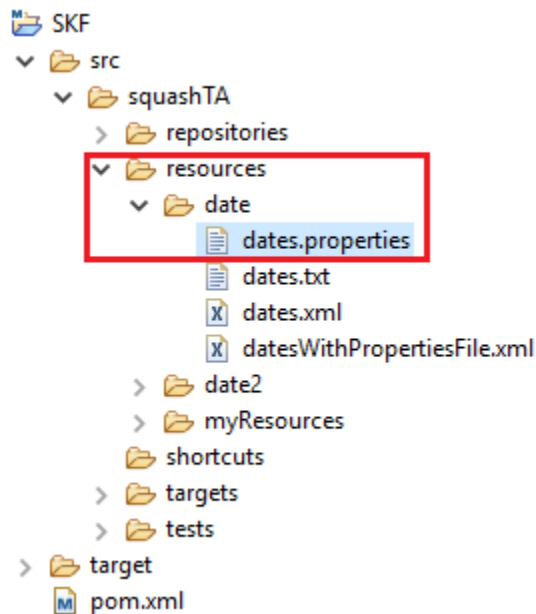
```

.properties File :

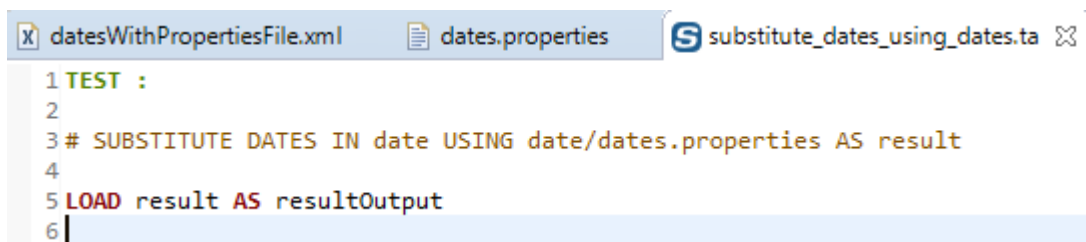


```
1 randomDate=20071983
2 randomDate2=27021980
3 randomDate3=24122018
```

.properties File Location :



SKF script :



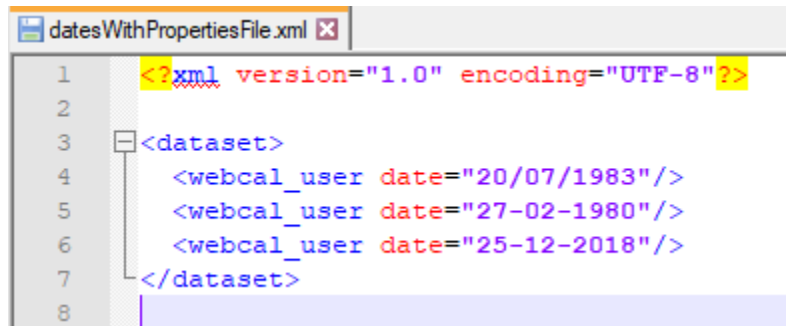
```
1 TEST :
2
3 # SUBSTITUTE DATES IN date USING date/dates.properties AS result
4
5 LOAD result AS resultOutput
6
```

Console output :

Result output :

```
# SUBSTITUTE DATES IN {files} USING {dates} FOR FILES WHOSE NAMES MATCHING {regex} AS {processed_files}
```

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test substitute_dates_using_dates.ta
[INFO] 3 property values used as date data sources.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190910_170724_7031154379990109042514\
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.415 s
[INFO] Finished at: 2019-09-10T17:07:27+02:00
[INFO] Final Memory: 21M/255M
[INFO] -----
```



What ?

This macro allows to replace dates (with dates you have specified) in a bundle of files whose names match a specific regular expression. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```
LOAD {dates} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳properties
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativeDate) USING placeholder{%%rand3}.
↳properties,$(squashtest.ta.param.include:{regex}) AS {processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {dates} : It can either be a path to a properties file or an inline command with keys and values for the dates you want to specify.

Example of inline command : \$(date1=01/01/2001 \n date2=31/12/2018).

Example of date in file to process : \${date(date1,dd/MM/yyyy).format(dd-MM-yyyy)}

- {regex} : The regular expression used to filter the files in the bundle.

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE DATES IN resources_folder USING resources_folder/dates.properties FOR FILES WHOSE
NAMES MATCHING .xml AS result_bundle
```

Or

```
# SUBSTITUTE DATES IN resources_folder USING $(date1=01/01/2001 \n date2=31/12/2018) FOR FILES  
WHOSE NAMES MATCHING .xml AS result_bundle
```

For more information, please check this [example](#) and this [one](#).

**# SUBSTITUTE DATES IN {files} USING {dates} FOR FILES WHOSE NAMES NOT MATCHING {regex}
AS {processed_files}**

What ?

This macro allows to replace dates (with dates you have specified) in a bundle of files whose names don't match a specific regular expression. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```
LOAD {dates} AS placeholder{%%rand2}.file  
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.  
→properties  
LOAD {files} AS __bundle{%%rand1}  
CONVERT __bundle{%%rand1} TO file(param.relativedate) USING placeholder{%%rand3}.  
→properties,$(squashtest.ta.param.exclude:{regex}) AS {processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {dates} : It can either be a path to a properties file or an inline command with keys and values for the dates you want to specify.

Example of inline command : \$(date1=01/01/2001 \n date2=31/12/2018).
--

Example of date in file to process : \${date(date1,dd/MM/yyyy).format(dd-MM-yyyy)}

- {regex} : The regular expression used to filter the files in the bundle.

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE DATES IN resources_folder USING resources_folder/dates.properties FOR FILES WHOSE  
NAMES NOT MATCHING .txt AS result_bundle
```

Or


```
# SUBSTITUTE DATES IN resources_folder USING $(date1=01/01/2001 \n date2=31/12/2018) FOR FILES
WHOSE NAMES NOT MATCHING .txt AS result_bundle
```

For more information, please check this [example](#) and this [one](#).

Commons component plugin - Macros - Substitute keys

Contents :

- *# SUBSTITUTE KEYS IN {files} USING {key_value_list} AS {processed_files}*
- *# SUBSTITUTE KEYS IN {files} FOR FILES WHOSE NAMES MATCHING {regex} USING {key_value_list} AS {processed_files}*
- *# SUBSTITUTE KEYS IN {files} FOR FILES WHOSE NAMES NOT MATCHING {regex} USING {key_value_list} AS {processed_files}*

SUBSTITUTE KEYS IN {files} USING {key_value_list} AS {processed_files}

What ?

This macro allows to replace specific keys by selected values in a bundle of files.

Underlying instruction :

```
LOAD {key_value_list} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳properties
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param) USING placeholder{%%rand3}.properties AS
↳{processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {key_value_list} : It can either be a path to a properties file or an inline command with keys and values.

Example of inline command : \$(key1=value1 \n key2=value2).

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

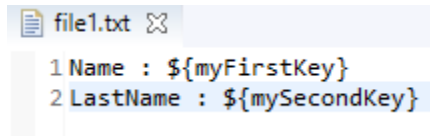
Examples :

```
# SUBSTITUTE KEYS IN resources_folder USING resources_folder/file.properties AS result_bundle
```

Or

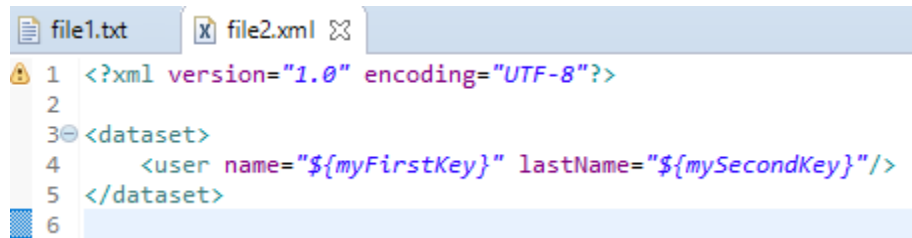
```
# SUBSTITUTE KEYS IN resources_folder USING $(oneKey=oneValue \n anotherKey=anotherValue) AS result_bundle
```

First file to process :



```
file1.txt
1 Name : ${myFirstKey}
2 LastName : ${mySecondKey}
```

Second file to process :



```
file1.txt file2.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <user name="${myFirstKey}" lastName="${mySecondKey}" />
5 </dataset>
6
```

.properties File :



```
file1.txt file2.xml file.properties
1 myFirstKey= John
2 mySecondKey= Snow
3
```

The folder containing files to process which corresponds to {files} :

SKF script :

In order to check that the macro worked properly, we added in this example, the instruction “**LOAD result AS resultOutput**” to be able to see the result output.

You can access to the result output in the following folder which contains temporary files :

```
C:\Users\*user name*\AppData\Local\Temp\Squash_TA\20190909_121048_957615127627872437436\tests\tests\substitute_ke
```

Result output for first file :

Result output for second file :

The screenshot displays the Squash Keyword Framework IDE interface. At the top, the **Package Explorer** shows the project structure:

- SKF
 - src
 - squashTA
 - repositories
 - resources
 - date
 - date2
 - myResources
 - example.txt
 - example2.txt
 - file.properties
 - file1.txt
 - file2.xml
 - ftpUploadExample.txt
 - shortcuts
 - targets
 - tests
 - target
 - pom.xml

The **substitute_keys.ta** test script is open, showing the following content:

```

1 TEST :
2
3 # SUBSTITUTE KEYS IN myResources USING myResources/file.properties AS result
4
5 LOAD result AS resultOutput

```

Below the script, a file list table shows the contents of the `myResources` directory:

Nom	Modifié le	Type	Taille
example.txt	11/09/2019 16:41	Document texte	1 Ko
example2.txt	11/09/2019 16:41	Document texte	1 Ko
file.properties	11/09/2019 16:41	Fichier PROPERTIES	1 Ko
file1.txt	11/09/2019 16:41	Document texte	1 Ko
file2.xml	11/09/2019 16:41	Document XML	1 Ko
ftpUploadExample.txt	11/09/2019 16:41	Document texte	1 Ko

Two file content views are shown below:

file1.txt content:

```

1 Name : John
2 LastName : Snow
3

```

file2.xml content:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <user name="John" lastName="Snow"/>
5 </dataset>
6

```

SUBSTITUTE KEYS IN {files} FOR FILES WHOSE NAMES MATCHING {regex} USING {key_value_list} AS {processed_files}

What ?

This macro allows to replace specific keys by selected values in a bundle of files whose names are matching a selected regular expression.

Underlying instruction :

```
LOAD {key_value_list} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳properties
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param) USING placeholder{%%rand3}.properties,
↳$(squashtest.ta.param.include:{regex}) AS {processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {regex} : The regular expression used to filter the files in the bundle.
- {key_value_list} : It can either be a path to a properties file or an inline command with keys and values.

Example of inline command : \$(key1=value1 \n key2=value2).

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE KEYS IN resources_folder FOR FILES WHOSE NAMES MATCHING .xml USING
resources_folder/file.properties AS result_bundle
```

Or

```
# SUBSTITUTE KEYS IN resources_folder FOR FILES WHOSE NAMES MATCHING .xml USING
$(oneKey=oneValue \n anotherKey=anotherValue) AS result_bundle
```

This example is based on the previous one. For more details, please check [here](#).

SKF script :

```
S substitute_keys_matching_regex.ta ⓘ
1 TEST :
2
3 # SUBSTITUTE KEYS IN myResources FOR FILES WHOSE NAMES MATCHING .xml USING myResources/file.properties AS result
4
5 LOAD result AS resultOutput
6
7
```

Or

```

S substitute_keys_matching_regex.ta
1 TEST :
2
3 # SUBSTITUTE KEYS IN myResources FOR FILES WHOSE NAMES MATCHING .xml USING $(myFirstKey=John \n mySecondKey=Snow) AS result
4
5 LOAD result AS resultOutput

```

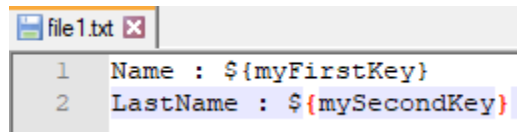
Console output :

```

[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test substitute_keys_matching_regex.ta
[INFO] the file example.txt was properly excluded.
[INFO] the file example2.txt was properly excluded.
[INFO] the file file.properties was properly excluded.
[INFO] the file file1.txt was properly excluded.
[INFO] the file ftpUploadExample.txt was properly excluded.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190912_095758_1615099261162986536592
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 32.359 s
[INFO] Finished at: 2019-09-12T09:58:00+02:00
[INFO] Final Memory: 23M/197M
[INFO] -----

```

The **.txt** and **.properties** files which don't match the selected regex are properly excluded.

Result output for first file :


```

1 Name : ${myFirstKey}
2 LastName : ${mySecondKey}

```

Result output for second file :

The **.xml** file is the only one that has been processed.

```

# SUBSTITUTE KEYS IN {files} FOR FILES WHOSE NAMES NOT MATCHING {regex} USING
{key_value_list} AS {processed_files}

```

What ?

This macro allows to replace specific keys by selected values in a bundle of files whose names are not matching a selected regular expression.

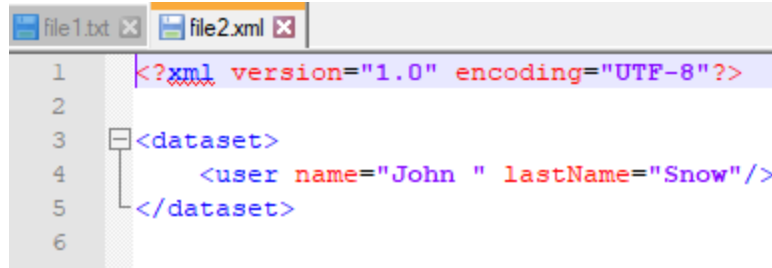
Underlying instruction :

```

LOAD {key_value_list} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↪properties

```

(continues on next page)



(continued from previous page)

```
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param) USING placeholder{%%rand3}.properties,
→$(squashtest.ta.param.exclude:{regex}) AS {processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {regex} : The regular expression used to filter the files in the bundle.
- {key_value_list} : It can either be a path to a properties file or an inline command with keys and values.

Example of inline command : \$(key1=value1 \n key2=value2).

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE KEYS IN resources_folder FOR FILES WHOSE NAMES NOT MATCHING .xml
USING resources_folder/file.properties AS result_bundle
```

Or

```
# SUBSTITUTE KEYS IN resources_folder FOR FILES WHOSE NAMES NOT MATCHING .xml
USING $(oneKey=oneValue,anotherKey \n anotherValue) AS result_bundle
```

This example is based on the previous one. For more details, please check [here](#).

SKF script :

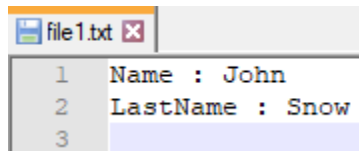
```
S substitute_keys_not_matching_regex.ta
1 TEST :
2
3 # SUBSTITUTE KEYS IN myResources FOR FILES WHOSE NAMES NOT MATCHING .xml USING myResources/file.properties AS result
4
5 LOAD result AS resultOutput
```

Console output :

The .xml file which match the selected regex is properly excluded.

Result output for first file :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test substitute_keys_not_matching_regex.ta
[INFO] the file file2.xml was properly excluded.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190912_103440_6691867721864919663379
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.514 s
[INFO] Finished at: 2019-09-12T10:34:43+02:00
[INFO] Final Memory: 20M/257M
[INFO] -----
```



Result output for second file :

The .xml file is not processed whereas the .txt is.

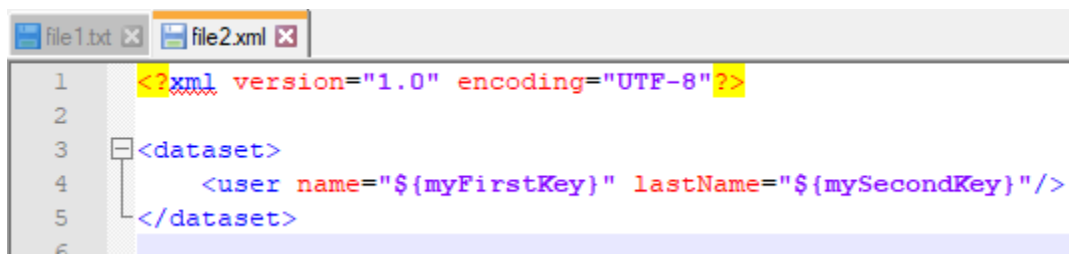
Commons component plugin - Macros - Substitute dates and keys

Contents :

- *# SUBSTITUTE DATES AND KEYS IN {files} USING {key_value_list} AS {processed_files}*
- *# SUBSTITUTE DATES AND KEYS IN {files} FOR FILES WHOSE NAMES MATCHING {regex} USING {key_value_list} AS {processed_files}*
- *# SUBSTITUTE DATES AND KEYS IN {files} FOR FILES WHOSE NAMES NOT MATCHING {regex} USING {key_value_list} AS {processed_files}*

SUBSTITUTE DATES AND KEYS IN {files} USING {key_value_list} AS {processed_files}

What ?



This macro allows to replace dates and keys by values in a bundle of files. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```
LOAD {key_value_list} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳properties
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativeDate) USING placeholder{%%rand3}.
↳properties AS __bundle{%%rand2}
CONVERT __bundle{%%rand2} TO file(param) USING placeholder{%%rand3}.properties AS
↳{processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {key_value_list} : It can either be a path to a properties file or an inline command with keys and values.

Example of inline command : \$(key1=value1 \n key2=value2).

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE DATES AND KEYS IN resources_folder USING resources_folder/file.properties AS re-
sult_bundle
```

Or

```
# SUBSTITUTE DATES AND KEYS IN resources_folder USING $(oneKey=oneValue \n anoth-
erKey=anotherValue) AS result_bundle
```

For more information please check the following sections : [substitute dates macro](#) and [substitute keys macro](#).

SUBSTITUTE DATES AND KEYS IN {files} FOR FILES WHOSE NAMES MATCHING {regex} USING {key_value_list} AS {processed_files}

What ?

This macro allows to replace dates and keys by values in a bundle of files whose names match a selected regular expression. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```

LOAD {key_value_list} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳properties
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativeDate) USING placeholder{%%rand3}.
↳properties, $(squashtest.ta.param.include:{regex}) AS __bundle{%%rand2}
CONVERT __bundle{%%rand2} TO file(param) USING placeholder{%%rand3}.properties,
↳$(squashtest.ta.param.include:{regex}) AS {processed_files}

```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {regex} : The regular expression used to filter the files in the bundle.
- {key_value_list} : It can either be a path to a properties file or an inline command with keys and values.

Example of inline command : \$(key1=value1 \n key2=value2).

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE DATES AND KEYS IN resources_folder FOR FILES WHOSE NAMES MATCHING .xml USING resources_folder/file.properties AS result_bundle
```

Or

```
# SUBSTITUTE DATES AND KEYS IN resources_folder FOR FILES WHOSE NAMES MATCHING .xml USING $(oneKey=oneValue \n anotherKey=anotherValue) AS result_bundle
```

For more information please check the following sections : *substitute dates macro* and *substitute keys macro* for files whose names are matching the given regular expression.

SUBSTITUTE DATES AND KEYS IN {files} FOR FILES WHOSE NAMES NOT MATCHING {regex} USING {key_value_list} AS {processed_files}

What ?

This macro allows to replace dates and keys by values in a bundle of files whose names are not matching a selected regular expression. For more information about the formulas to use in order to replace dates, please check this [page](#).

Underlying instruction :

```
LOAD {key_value_list} AS placeholder{%%rand2}.file
CONVERT placeholder{%%rand2}.file TO properties(structured) AS placeholder{%%rand3}.
↳properties
LOAD {files} AS __bundle{%%rand1}
CONVERT __bundle{%%rand1} TO file(param.relativeDate) USING placeholder{%%rand3}.
↳properties,$(squashtest.ta.param.exclude:{regex}) AS __bundle{%%rand2}
CONVERT __bundle{%%rand2} TO file(param) USING placeholder{%%rand3}.properties,
↳$(squashtest.ta.param.exclude:{regex}) AS {processed_files}
```

> Input :

- {files} : The bundle of files where you want to apply the substitution.
- {regex} : The regular expression used to filter the files in the bundle.
- {key_value_list} : It can either be a path to a properties file or an inline command with keys and values.

Example of inline command : \$(key1=value1 \n key2=value2).

> Output :

- {processed_files} : The bundle of filtered files that have been processed.

Examples :

```
# SUBSTITUTE DATES AND KEYS IN resources_folder FOR FILES WHOSE NAMES NOT MATCHING .txt
USING resources_folder/file.properties AS result_bundle
```

Or

```
# SUBSTITUTE DATES AND KEYS IN resources_folder FOR FILES WHOSE NAMES NOT MATCHING .txt
USING $(oneKey=oneValue \n anotherKey=anotherValue) AS result_bundle
```

For more information please check the following sections : [substitute dates macro](#) and [substitute keys macro](#) for files whose names are not matching the given regular expression.

6.1.5 Commons component plugin - Advanced Users

Commons component plugin - Converters

Contents :

- *From file ...*
 - ... *to bundle*
 - ... *to csv*
 - ... *to directory*

- ... to file (using param.relativeDate)
- ... to file (using param)
- ... to properties
- ... to script.java
- ... to xml

Since converters create resource of B type using a resource of A type, the documentation on converters follows a pattern 'from A to B' (e.g. from `file` to `query.sql`). Most of the time knowing the category of the resources you have and/or you want to obtain will be enough to find which converter you need using the following list. However remember that a converter is fully qualified by his signature : the only way to disambiguate situations where multiple converters consuming and producing the same categories could apply is to specify the name of the converter too.

Like the other engine components, a converter is configurable. It means that you can pass parameters and tweak the operation.

From file ...

A test project is mostly made of files, including the test resources. This is why the `file` category is so important and overly processed by converters.

... to bundle

Category-Name : *unchecked*

What ?

This *unchecked* converter will convert a `file` type resource to a `bundle` type resource. It checks during the conversion that the resource to convert is really pointing to a directory.

```

CONVERT {resourceToConvert<Res:file>} TO bundle (unchecked) AS {converted<Res:bundle>} [ USING
{mainPath<Res:file>} ]

```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references the root directory. This root directory should contains the whole files and directories of the bundle (`file` type resource).
- `mainPath<Res:file>` (OPTIONAL) : The name of the configuration resource. The content of the file should be: `mainpath:relativePathToMainFile` (Note : you could use an inline definition). This path to main file should be relative to the directory given as the root directory.

> Output :

- `converted<Res:bundle>` : The name of the converted resource (`bundle` type resource).

Example (with the USING clause and an inline definition) :

```
LOAD path/to/rootDirectory AS bundle.file
CONVERT bundle.file TO bundle (unchecked) AS bundle.bundle USING $(mainpath:relative/path/to/mainFile.txt)
```

... to csv

Category-Name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a `csv` type resource. It checks during the conversion that the resource to convert is really pointing to a csv file.

```
CONVERT {resourceToConvert<Res:file>} TO csv (structured) AS {converted<Res:csv>} [ USING {main-Path<Res:file>} ]
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references the csv file (file type resource).
- `mainPath<Res:file>` (OPTIONAL) : The name of the configuration resource. given as the root directory.

> Output :

- `converted<Res:csv>` : The name of the converted resource (csv type resource).

Example :

```
LOAD csv1/mycsv.csv AS mycsv.file
CONVERT mycsv.file TO csv (structured) AS mycsv.csv
```

... to directory

Category-Name : *filesystem*

What ?

This *filesystem* converter will convert a `file` type resource to a `directory` type resource. It checks during the conversion that the resource to convert is really pointing to a directory.

`CONVERT {resourceToConvert<Res:file>} TO directory (filesystem) AS {converted<Res:directory>}`

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a directory (`file` type resource).

> Output :

- `converted<Res:directory>` : The name of the converted resource (`directory` type resource).

Example :

`LOAD path/to/Directory AS directory.file
CONVERT directory.file TO directory (filesystem) AS directory.directory`

... to file (using `param.relativedate`)

Category-Name : *param.relativedate*

What ?

This *param.relativedate* converter will convert a `file` type resource to another `file` type resource. In the context of the aging factor a mechanism has been set-up in SKF to manage dates. It consists in replacing dates of the data set with formulas of type :

`${function}`

where function is :

- `now().format(f)` : return the execution date at the 'f' format.
- `now().addDay(n).format(f)` : return the execution date + 'n' days (positive or negative) at the 'f' format.
- `now().addMonth(n).format(f)` : return the execution date + 'n' months (positive or negative) at the 'f' format.
- `now().addYear(n).format(f)` : return the execution date + 'n' years (positive or negative) at the 'f' format.

Table 1: Examples with an SKF script run on 16/05/2012 :

Function	Result
now().format(yyyyMMdd)	20120516
now().addDay(1).format(dd/MM/yyyy)	17/05/2012
now().addDay(-2).addMonth(1).addYear(-3).format(dd MMMM yyyy)	14 June 2009

Since **1.7.0**, you can overwrite the default locale of your date, with a language code or a language and a country :

- now().[...].format(f, l) : return the date at the 'f' format in the 'l' language, where 'l' is a lower-case, two-letter code as defined by ISO-639.
- now().[...].format(f, l, C) : return the date at the 'f' format in the 'l' language of the 'C' country, where 'l' is a lower-case, two-letter code as defined by ISO-639 and 'C' is an upper-case, two-letter code as defined by ISO-3166.

Function	Result
now().addMonth(1).format(dd MMMM yyyy, fr)	16 juin 2012
now().addMonth(1).format(dd MMMM yyyy, de, DE)	16 Juni 2012

Since **1.10.0**, you can manipulate the date in the \${function} with 3 new options :

- The ensureWorkingDay(\$param) function adjusts the computed date to the nearest working day before or after the input date (\$param must be replaced by AFTER or BEFORE).
- The addWorkingDay(n) function allows to add a given number of working days to its input date (n is a positive or negative integer).
- If you want to specify a date which is different from now(), you can use the following method :

Example :

```
LOAD folder/file AS templateData
DEFINE $(date-key=05051978) AS rawProperties
CONVERT rawProperties TO properties(structured) AS prop
CONVERT templateData TO file(param.relateddate) USING prop AS data

Written in the file to process : ${date(date-key, ddMMyyyy).addDay(-1) .
↪format(yyyyMMdd)}
```

Function	Result
now().addMonth(4).ensureWorkingDay(AFTER).format(dd MMM yyyy, fr)	17 Septembre 2012
now().addMonth(4).ensureWorkingDay(BEFORE).format(dd MMM yyyy, fr)	14 Septembre 2012
now().addWorkingDay(10).format(dd MMM yyyy, fr)	30 Mai 2012

The Working days are read from component configuration through the USING clause.

Example of file :

- org.squashtest.tf.plugin.common.parms.data.WorkedSunday=false
- org.squashtest.tf.plugin.common.parms.data.WorkedSaturday=false
- org.squashtest.tf.plugin.common.parms.data.WorkedMonday=true
- org.squashtest.tf.plugin.common.parms.data.nonWorkingDays=2018-05-01,2018-12-25,2019-01-01

The **fourth** parameter list all the non working days, you must declare the date like yyyy-MM-dd and separate them with a comma.

The converter transforms formulas `${function}` in valid dates at the execution :

```
CONVERT {resourceToConvert<Res:file>} TO file  
(param.relativeDate) [USING {properties<Res:properties>}] AS {converted<Res:file>}
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a file. All the dates of the file must have been replaced by formulas like `'${function}'`.
- `properties<Res:properties>` : If the `resourceToConvert` is a bundle containing binary file, this properties file must contain either the `squashtest.tf.param.exclude` OR the `squashtest.tf.param.include` parameter. Both of them use regular expressions. According to the selected parameter, the file(s) matching the regex will or will not be treated by this converter.

> Output :

- `converted<Res:file>` : The name of the converted resource (`file` type resource). The finale resource is the same than the input resource, the difference is that formulas have been replaced with valid dates.

Example :

```
LOAD path/myfile AS myfile.file  
CONVERT myfile.file TO file (param.relativeDate) AS myfile2.file
```

... to file (using param)

Category-Name : *param*

Since 1.6.0 :

What ?

This *param* converter will convert a `file` type resource to another `file` type resource. After the conversion all the placeholder, whose key was found in the properties resource given in the `USING` clause, should have been replace by it's valid value (the value associate to the key in the property resource file). The initial File resource could be a file or a directory. If it's a directory then all the file contained in this directory should be process.

Here are the rule used :

- Placeholder syntax : `${key}`
- Authorized characters for properties key : letters (a-z;A-Z), digits (0-9), underscore (`_`), dot (`.`) and dash (`-`)
- The convert instruction could take one or many properties file AND one or many inline statements
- If a property key is defined many times, then it's the last stated which is take into account

- If a placeholder is surrounding with character @, then the placeholder is escaped. For example if we have in the file to process : @\${test}@, then we will have in the final file : \${test}
- If a placeholder key is not found in the properties key, then the placeholder is escaped.

```
CONVERT {resourceToConvert<Res:file>} TO file (param) USING {properties<Res:properties>} AS {converted<Res:file>}
```

> Input :

- resourceToConvert<Res:file> : The name (in the context) of the resource which references a file.
- properties<Res:properties> : This properties file contains the mapping key-value. If the resourceToConvert is a bundle containing binary file, this properties file must also contain either the squashtest.tf.param.exclude OR the squashtest.tf.param.include parameter. Both of them are regular expression the file contained in the bundle must match to be or not to be treated by this converter.

> Output :

- converted<Res:file> : The name of the converted resource (file type resource). The finale resource is the same than the input resource, the difference is that the placeholder \${...} have been replaced with their valid values.

Example :

```
LOAD sahi/placeholder.properties AS placeholder.file
CONVERT placeholder.file TO properties (structured) AS placeholder.properties

LOAD sahi/main/simple-script.sah AS sahiFile
CONVERT sahiFile TO file (param) USING placeholder.properties AS processedSahiFile

CONVERT processedSahiFile TO script.sahi (script) AS suite
```

Where :

```
* placeholder.properties contains : c3p0.data=Using c3p0
* simple-script.sah contains : _click(_link("${c3p0.data}"));
```

Then :

```
* processedSahiFile should contains : _click(_link("Using c3p0"));
```

... to properties

Category-Name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a `properties` type resource.

```
CONVERT {resourceToConvert<Res:file>} TO properties (structured) AS {converted<Res:properties>}
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references the '.properties' file (list of properties key / value) where the used separator is '='.

> Output :

- `converted<Res:properties>` : The name of the converted resource (`properties` type resource).

Example :

```
LOAD path/myfile.properties AS myfile.file  
CONVERT myfile.file TO properties (structured) AS myfile.prop
```

... to script.java

Category-Name : *compile*

What ?

This *compile* converter will convert a `file` type resource to a `script.java` type resource.

```
CONVERT {resourceToConvert<Res:file>} TO script.java (compile) AS {converted<Res:script.java>} [ USING {mainPath<Res:file>} ]
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references the root directory of the java code bundle which contains resources and the java's source code.
- `mainPath<Res:file>` (OPTIONAL) : The name of the configuration resource. It represents a configuration file containing java compilation options. (Possible options are those of the Java compiler present on the machine). In this file options can be written:
 - In line separated with a space character
 - One option per line
 - A mix of both

> Output :

- `converted<Res:script.java>`: The name of the converted resource (Resource of type `script.java`). It contains the compiled java code.

Example :

```
LOAD path/to/javaBundle AS bundleJava.file
CONVERT bundleJava.file TO script.java (compile) AS bundleJava.compiled USING $(main-path:relative/path/to/compileOptions)
```

... to xml

Category-Name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a `xml` type resource. It checks during the conversion that the resource to convert is really `xml` category.

```
CONVERT {resourceToConvert<Res:file>} TO xml (structured) AS {converted<Res:xml>}
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references the `xml` file.

> Output :

- `converted<Res:xml>` : The name of the converted resource (`xml` type resource).

Example :

```
LOAD myfile.xml AS myfile.file
CONVERT myfile.file TO xml (structured) AS myXMLfile
```

Commons component plugin - Commands

Contents :

- *cleanup*
- *pause*

- *log*
-

cleanup

@See : Since **Squash TA 1.7.0**, this command moved to the *Local Process Plugin*.

pause

Description :

Makes the test execution sleep for a given time (expressed in milliseconds).

EXECUTE pause WITH \$(<n>) AS \$()

> Input :

- <n> : an integer. It represents the time in milliseconds.

log

Description :

This instruction allows writing a message in the log console.

EXECUTE log WITH \$(<message>) USING \$(logLevel: <level>) AS \$()

> Input :

- <message> : The message you want to display in the log console.
- <level> : The log level to use : DEBUG, INFO, WARN or ERROR.

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

In an SKF script, the tokens are the spine and the engine components are the muscles. The package commons-components is a basic plugin shipped with SKF. It provides the platform with some basic Resources, Repositories, Targets, and Engine Components.

6.2 Database Plugin

6.2.1 Database Plugin - Prerequisites

To connect to a database and so to use database targets, an automation **Squash TF** project need an adequate JDBC driver (the driver depends on the database type : mysql, oracle...).

The driver is provided in the form of a maven artifact that we add in the automation project. To do so, we modify the 'squash-ta-maven-plugin' inside the *pom.xml* file :

```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.squashtest.ta</groupId>
      <artifactId>squash-ta-maven-plugin</artifactId>
      <version>squash-ta-maven-plugin version</version>

      <dependencies>
        <dependency>
          <groupId>JDBC driver groupId</groupId>
          <artifactId>JDBC driver artifact ID</artifactId>
          <version>JDBC driver version</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
...
```

Example of JDBC Driver for MySql :

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.17</version>
</dependency>
```

Project's POM File :

.properties File to connect to database :

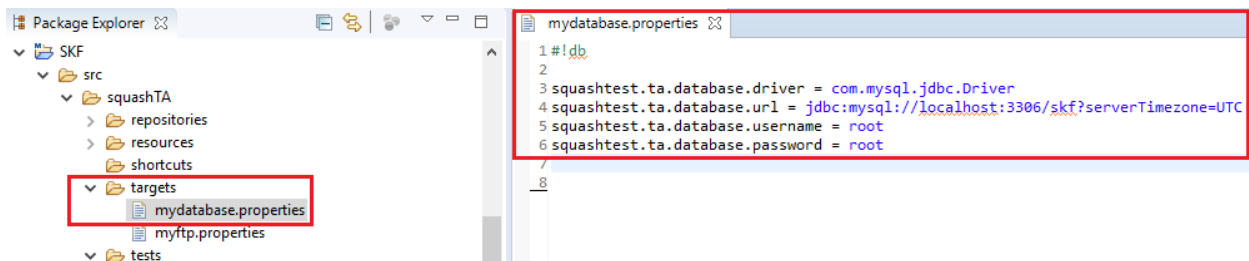
For more information, please check this [section](#).

```

<build>
  <plugins>
    <!-- Configuration of the Squash TA framework used by the project -->
    <plugin>
      <groupId>org.squashtest.ta</groupId>
      <artifactId>squash-ta-maven-plugin</artifactId>
      <version>${ta.framework.version}</version>

      <!-- Here you can add libraries to the engine classpath, using the <dependencies></dependencies> tag -->
      <!-- A sample with the mySql jdbc driver -->
      <dependencies>
        <dependency>
          <groupId>mysql</groupId>
          <artifactId>mysql-connector-java</artifactId>
          <version>8.0.17</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>

```



6.2.2 Database Plugin - Target

Database

Category-name : *database*

What ?

A *database* target represents, well, a database. The file (*.properties*) which allows to define this target to test contains all needed informations to connect to the database.

Configuration (basic) : A simple *.properties* file dropped in the 'targets' directory of your test project. The file must include the shebang on the very first line : *#!/db* and it must contain AT LEAST *squashtest.tf.database.driver* and *squashtest.tf.database.url*.

Available properties :

- *squashtest.ta.database.driver* : the jdbc driver supporting your database
- *squashtest.ta.database.url* : aka connection string, this is the url of your database
- *squashtest.ta.database.username* : the username to connect with
- *squashtest.ta.database.password* : the corresponding password

Remark

If properties *squashtest.tf.database.username* and *squashtest.tf.database.password* are not indicated, then the user name and the user password must be indicated in the *squashtest.tf.database*.

url property. If they are indicated in both, then datas from the properties `squashtest.tf.database.username` and `squashtest.tf.database.password` prime.

Configuration (advanced) : Using the same `.properties` file you can also specify options related to the pooling of the datasource. As for version 1.0.x SKF will create its own datasource - no support for JNDI references for instance (yet).

To this end, SKF is backed by the `c3p0` technology. SKF will transmit to `c3p0` its regular configuration properties (see [here](#)). For the sake of consistency with the rest of the file, each key must be prefixed with `squashtest.ta.database.pool.` + property. For instance `squashtest.ta.database.pool.maxPoolSize` will configure the property `maxPoolSize`.

The only exception to this are the `c3p0` properties `user` and `password`, that already exist as basic configuration. Consequently they will be ignored : namely, `squashtest.ta.database.pool.user` and `squashtest.ta.database.pool.password` will be shunted. Please use the basic keys instead.

Example of valid configuration file :

```
#!/db

# basic configuration
squashtest.ta.database.driver = com.mysql.jdbc.Driver
squashtest.ta.database.url = jdbc:mysql://localhost:3306/my_database
squashtest.ta.database.username = tester
squashtest.ta.database.password = _tester

# advanced configuration
squashtest.ta.database.pool.acquireRetryDelay = 3000
squashtest.ta.database.pool.maxPoolSize = 40
```

6.2.3 Database Plugin - Resources

Contents :

- *conf.dbunit*
- *conf.dbunit.ppk*
- *dataset.dbunit*
- *filter.dbunit*
- *parameter.named.sql*
- *parameter.indexed.sql*
- *query.sql*
- *result.sql*
- *script.sql*

conf.dbunit

Category-name : *conf.dbunit*

What ?

conf.dbunit is a resource type whose role is to configure DbUnit transactions.

How to use it ?

conf.dbunit.ppk

Category-name : *conf.dbunit.ppk*

What ?

conf.dbunit.ppk is a resource type that represents a pseudo-primary key filter for DbUnit transactions. The file format is as follows : for each table, define a property with the name of the table. The value of the property is the comma-separated list of the names of the columns that make up the primary key.

Here is an example of definition file content :

```
employee=employee_id  
company=company_id  
contract=contract_employee_id,contract_company_id
```

Why ?

Usually DbUnit reads directly from the database, information about the tables it needs to know, including their primary keys. However some tables simply have no primary key, which can lead DbUnit to failures for a few operations. The *conf.dbunit.ppk* is a way to provide DbUnit with these extra information.

dataset.dbunit

Category-name : *dataset.dbunit*

What ?

dataset.dbunit is a resource type that represents a DbUnit DataSet.

How to use it ?

filter.dbunit

Category-name : *filter.dbunit*

What ?

filter.dbunit is a resource type that represents a Dbunit Filter. These filters are used in assertions for comparison between DbUnit datasets (*dataset.dbunit*). Their purpose is to exclude / include from the comparison some tables or some columns if you need to narrow the scope of your assertion.

How to use it ?

parameter.named.sql

Category-name : *parameter.named.sql*

What ?

parameter.named.sql is a resource type that represents a map of parameters for parameterized sql queries using named parameters (see *query.sql*).

parameter.indexed.sql

Category-name : *parameter.indexed.sql*

What ?

parameter.indexed.sql is a resource type that represents a list of parameters for parameterized sql queries using positional parameters (see *query.sql*).

query.sql

Category-name : *query.sql*

What ?

query.sql is a resource type that represents a query written in SQL. It can be parameterized either using named parameters or positional (indexed) parameters.

- **Named parameters** : Usually a named parameter appears in a sql query as a column ‘:’ followed by its name. For instance :

```
SELECT * FROM MY_TABLE WHERE id = :value;
with value: the name of the parameter
```

- **Indexed parameters** : Usually indexed parameters appear in a sql query as a question mark ‘?’. For instance :

```
SELECT * FROM MY_TABLE WHERE id = ?;
with '?': the indexed parameter
```

Since the parameters are identified by their position relative to each others, the order in which they are passed in does actually matter (they are matched by their position).

result.sql

Category-name : *result.sql*

What ?

result.sql is a resource type that represents the result of a sql query (or *query.sql* in **TF**).

script.sql

Category-name : *script.sql*

What ?

script.sql is a resource type that represents a script written in SQL. They aren't meant to read data, rather to perform massive operations in bulk like insertion or manipulation of the structure of the database.

6.2.4 Database Plugin - Macros

Database Plugin - Macros - Execute SQL and SQL script

Contents :

- *# EXECUTE_SQL {file} ON {database} AS {result}*
- *# EXECUTE_SQL_SCRIPT {file} ON {database} AS {result} WITH ENCODING {encoding} AND DELIMITER {delimiter}*
- *#EXECUTE_SQL_SCRIPT_BY_REMOVING_SEPARATOR {file} ON {database} AS {result}*

EXECUTE_SQL {file} ON {database} AS {result}

What ?

This macro will load and execute an SQL query against the database, then the result will be returned under the name you typed for the last parameter.

Underlying instructions :

```
LOAD {file} AS __temp{%%rand1}.file

CONVERT __temp{%%rand1}.file TO file(param.relativeDate) AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO query.sql(query) AS __temp_{%%rand3}.query
EXECUTE execute WITH __temp_{%%rand3}.query ON {database} AS {result}
```

> Input :

- {file} : A SQL query ('.sql').
- {database} : The name (in the context) of the database to use (database type target).

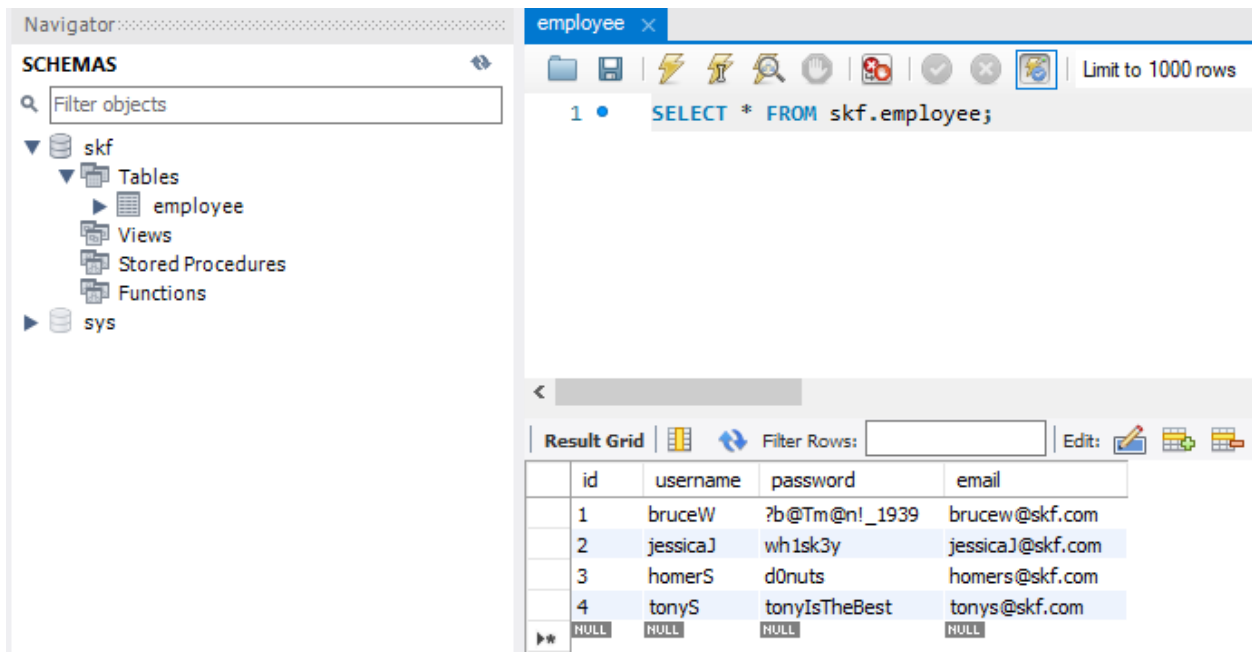
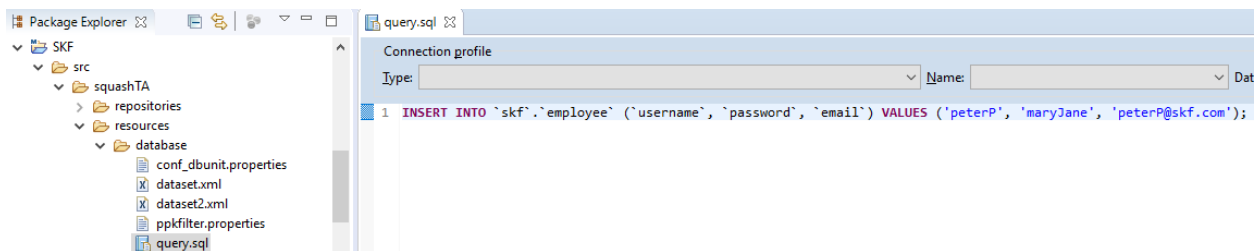
Remark : The {file} must respect the same rules than a file which would serve to create an SKF query.sql type resource via the converter (*From file to query.sql*).

> Output :

- {result} : The name of the resource which will contain the result of the SQL query(result.sql type resource).

Example :

```
# EXECUTE_SQL path/to/query1.sql ON my_database AS result
```

Example with an INSERT query :**Database overview :****.sql file for the query :****SKF script :**

The new employee has been inserted in the database :

```
S database_execute_sql.ta ⌕
1 TEST :
2
3 # EXECUTE_SQL database/query.sql ON mydatabase AS result
4
```

Navigator: SCHEMAS

Filter objects

- skf
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- sys

employee x

Limit to 1000 rows

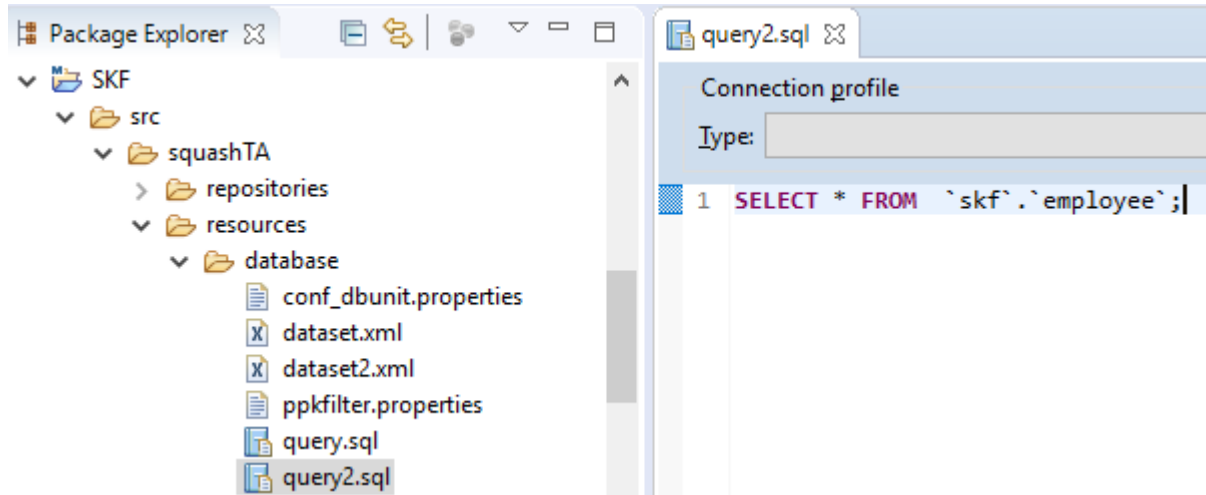
1 • SELECT * FROM skf.employee;

Result Grid

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	bruceW@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homerS	d0nuts	homerS@skf.com
4	4	tonyS	tonyIsTheBest	tonys@skf.com
5	5	peterP	maryJane	peterP@skf.com
▶*	NULL	NULL	NULL	NULL

Example with a SELECT query :

.sql file for the query :



SKF script :

```
S database_execute_sql2.ta
1 TEST :
2
3 # EXECUTE_SQL database/query2.sql ON mydatabase AS result
4 CONVERT result TO dataset.dbunit (dataset) USING $(tablename : employee) AS dataset
5 CONVERT dataset TO xml (dbu.xml) AS my_xml_file
6
```

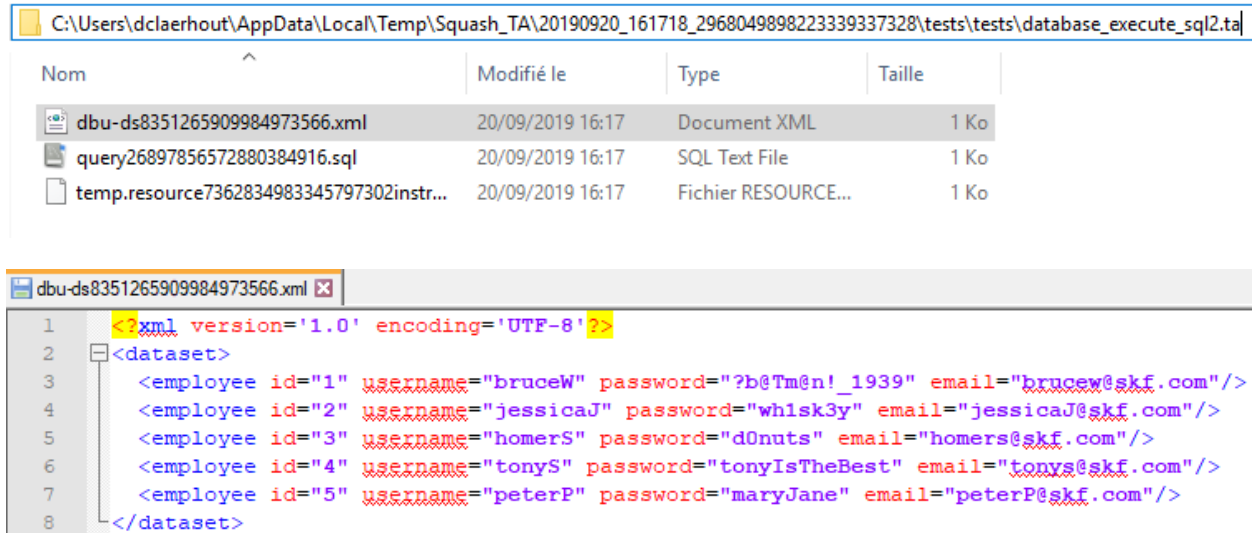
To be able to see the result output, we added in this example the following instructions :

```
CONVERT result TO dataset.dbunit (dataset) USING $(tablename : user)
↪AS dataset
CONVERT dataset TO xml (dbu.xml) AS my_xml_file
```

You can access to the result output in the following folder which contains temporary files :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test database_execute_sql2.ta
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLEAER~1\AppData\Local\Temp\Squash_TA\20190920_161718_296804989822339337328
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

Result output :



EXECUTE_SQL_SCRIPT {file} ON {database} AS {result} WITH ENCODING {encoding} AND DELIMITER {delimiter}

What ?

This macro will load and execute an SQL script against the database, then the result will be returned under the name you typed for the last parameter.

Underlying instructions :

```

LOAD {file} AS __temp{%%rand1}.file
DEFINE $(encoding:{encoding}) AS encoding{%%rand1}.opts
DEFINE $(delimiter:{delimiter}) AS delimiter{%%rand1}.opts

CONVERT __temp{%%rand1}.file TO file(param.relativedate) AS __temp{%%rand2}.file_
  ↳ USING encoding{%%rand1}.opts
CONVERT __temp{%%rand2}.file TO script.sql AS __temp_{%%rand3}.script USING encoding{
  ↳ %rand1}.opts, delimiter{%%rand1}.opts
EXECUTE execute WITH __temp_{%%rand3}.script ON {database} AS {result}

```

> Input :

- {file} : An SQL script
- {database} : The name (in the context) of the database to use (database type target).
- Optional - {encoding} : Parameter representing the query file encoding. Default value : “UTF-8”.
- Optional - {delimiter} : Parameter representing the SQL block delimiter. Default value : “@@”. It can be used in conjunction with {encoding} or by itself - in which case {encoding} will take its value by default.

Remark : The {file} must respect the same rules as a file used to create an SKF script.sql type resource via the converter (*From file to script.sql*).

> Output :

- {result} : A free identifier for the result. As the ‘execute’ command with an sql script return an empty resource, this result resource will also be empty.

Example :

```
# EXECUTE_SQL_SCRIPT path/to/script.sql ON my_database AS result WITH ENCODING UTF-16 AND DELIMITER $$$
```

script.sql file :

```
DROP TABLE IF EXISTS `skf`.`employee`;

CREATE TABLE `skf`.`employee` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(45) NULL,
  `password` VARCHAR(45) NULL,
  `email` VARCHAR(45) NULL,
  PRIMARY KEY (`id`));

INSERT INTO `skf`.`employee` (`username`, `password`, `email`) VALUES (
  ↳ 'bruceW', '?b@Tm@n!_1939', 'brucew@skf.com');
INSERT INTO `skf`.`employee` (`username`, `password`, `email`) VALUES (
  ↳ 'jessicaJ', 'whl3k3y', 'jessicaJ@skf.com');
INSERT INTO `skf`.`employee` (`username`, `password`, `email`) VALUES (
  ↳ 'homerS', 'd0nuts', 'homers@skf.com');
INSERT INTO `skf`.`employee` (`username`, `password`, `email`) VALUES ('tonyS
  ↳ ', 'tonyIsTheBest', 'tonys@skf.com');
INSERT INTO `skf`.`employee` (`username`, `password`, `email`) VALUES (
  ↳ 'çàéèééééééèèèèèè', 'ççççççéééééééé', 'test');
```

We encode the file in ISO-8859-1 and use special characters :

The screenshot shows a SQL editor window titled 'script.sql'. The code is the same as in the previous block, but the special characters in the last INSERT statement are now represented by their ISO-8859-1 byte values: 'çàéèééééééèèèèèè' and 'ççççççéééééééé'. The status bar at the bottom indicates 'Structured Query Language file', 'length: 836 lines: 14', 'Ln: 14 Col: 115 Sel: 0 | 0', 'Windows (CR LF)', and 'ISO 8859-1' (highlighted in a red box).

SKF script :

```
database_execute_script.ta
1 TEST :
2
3 # EXECUTE_SQL_SCRIPT database/script.sql ON mydatabase AS result
4 |
```

Database overview without specifying encoding in macro :

SKF script :

Database overview with encoding :

Navigator

SCHEMAS

Filter objects

- skf
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- sys

employee x

Limit to 1000 rows

1 • `SELECT * FROM skf.employee;`

Result Grid

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	brucew@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homerS	d0nuts	homers@skf.com
4	4	tonyS	tonyIsTheBest	tonys@skf.com
5	5	çàééééé...	çççççéééééé...	test
▶*	NULL	NULL	NULL	NULL

S database_execute_script.ta

```

1 TEST :
2
3 # EXECUTE_SQL_SCRIPT database/script.sql ON mydatabase AS result WITH ENCODING ISO-8859-1
4

```

Navigator

SCHEMAS

Filter objects

- skf
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- sys

employee x

Limit to 1000 rows

1 • `SELECT * FROM skf.employee;`

Result Grid

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	brucew@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homerS	d0nuts	homers@skf.com
4	4	tonyS	tonyIsTheBest	tonys@skf.com
5	5	çàééééé...	çççççéééééé...	test
▶*	NULL	NULL	NULL	NULL

#EXECUTE_SQL_SCRIPT_BY_REMOVING_SEPARATOR {file} ON {database} AS {result}**What ?**

This macro will load and execute an SQL script against the database, then the result will be returned under the name you typed for the last parameter. The separator (“;”) at the end of each SQL query of the script will be removed.

Underlying instructions:

```
LOAD {file} AS __temp{%%rand1}.file

CONVERT __temp{%%rand1}.file TO file(param.relativedate) AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO script.sql AS __temp_{%%rand3}.script
EXECUTE execute WITH __temp_{%%rand3}.script ON {database} USING $(keep.
↳separator:false) AS {result}
```

> Input :

- {file} : An SQL script.
- {database} : The name (in the context) of the database to use (database type target).

Remark : The {file} must respect the same rules than a file which would serve to create an SKF script.sql type resource via the converter (*From file to script.sql*).

> Output :

- {result} : A free identifier for the result. As the ‘execute’ command with an sql script returns an empty resource, this result resource will also be empty.

Example :

```
# EXECUTE_SQL_SCRIPT_BY_REMOVING_SEPARATOR path/to/my_script.sql ON my_database AS result
```

Database Plugin - Macros - Insert DbUnit**Contents :**

- *# INSERT_DBUNIT {dataset} INTO {database}*
- *# INSERT_DBUNIT {dataset} INTO {database} WITH CONFIG {config}*
- *# INSERT_DBUNIT {dataset} INTO {database} USING {ppkfilter}*
- *# INSERT_DBUNIT {dataset} INTO {database} WITH CONFIG {config} USING {ppkfilter}*

INSERT_DBUNIT {dataset} INTO {database}

What ?

This macro will insert all the data listed in the 'dataset file' into the 'database'.

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand1}.file

CONVERT __temp{%%rand1}.file TO file(param.relativedate) AS __temp_{%%rand2}.file
CONVERT __temp_{%%rand2}.file TO xml(structured) AS __temp_{%%rand3}.xml
CONVERT __temp_{%%rand3}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand4}.dbu

EXECUTE insert WITH __temp_{%%rand4}.dbu ON {database} USING $(operation:insert) AS __
temp_{%%rand5}.result
```

> Input :

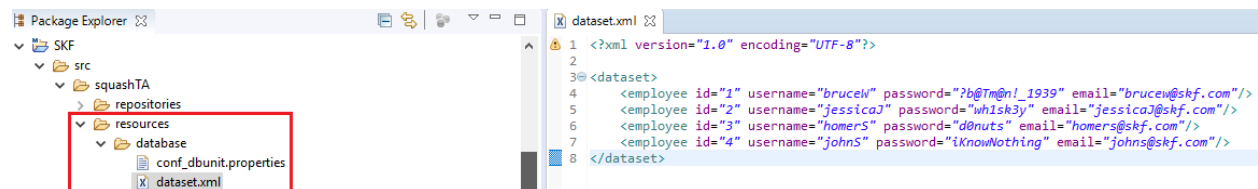
- {dataset} : A flat xml dbunit dataset file
- {database} : The name (in the context) of the database to use (database type target).

Remark : The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).

Example :

```
# INSERT_DBUNIT path/to/dataset.xml INTO my_database
```

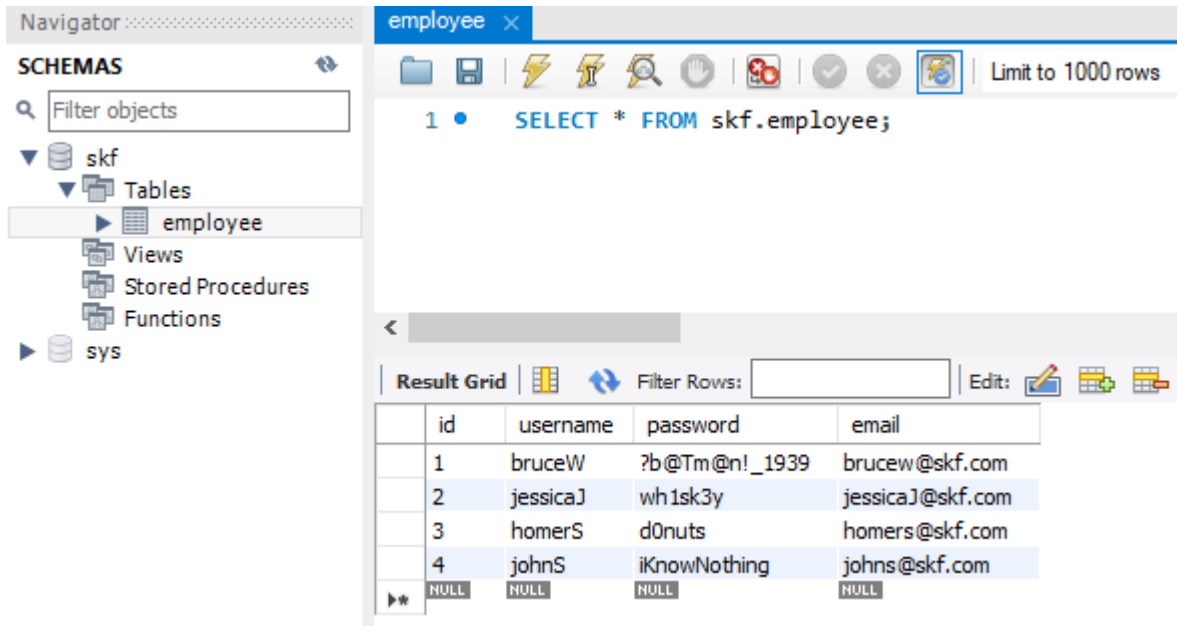
DbUnit dataset :



SKF script :

```
dataset.xml database_insert_dbunit.ta
1 TEST :
2
3 # INSERT_DBUNIT database/dataset.xml INTO mydatabase
4
```

The employees have been inserted in the database :



INSERT_DBUNIT {dataset} INTO {database} WITH CONFIG {config}

What ?

This macro will insert all the data listed in the 'dataset file' into the 'database' using a DbUnit configuration file.

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp{%%rand3}.file

CONVERT __temp{%%rand3}.file TO file(param,relativedate) AS __temp_{%%rand4}.file
CONVERT __temp_{%%rand4}.file TO xml(structured) AS __temp_{%%rand5}.xml
CONVERT __temp_{%%rand5}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand6}.dbu

EXECUTE insert WITH __temp_{%%rand6}.dbu ON {database} USING $(operation:insert),__
↳temp{config}{%%rand2}.conf AS __temp_{%%rand7}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {config} : A configuration file for DbUnit ('.properties').

Remarks :

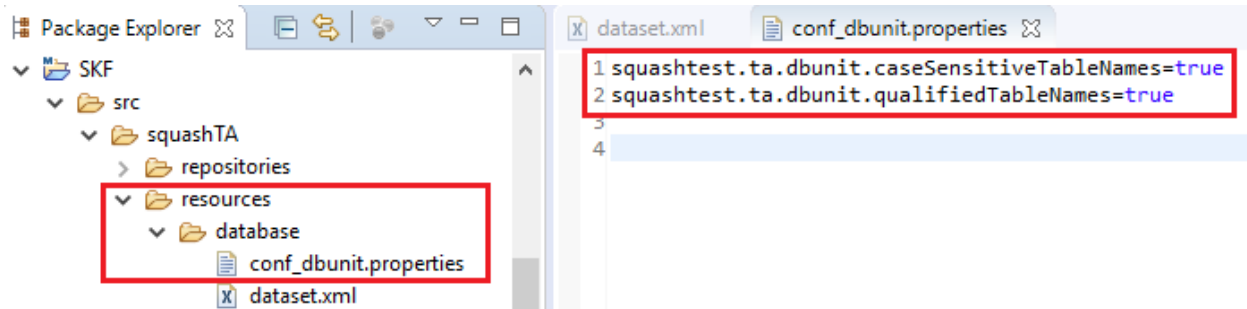
1. The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).

- The file designed by {config} must respect the same rules than a file which would serve to create an SKF `conf.dbunit` type resource via the converter (*From file to conf.dbunit*).

Example :

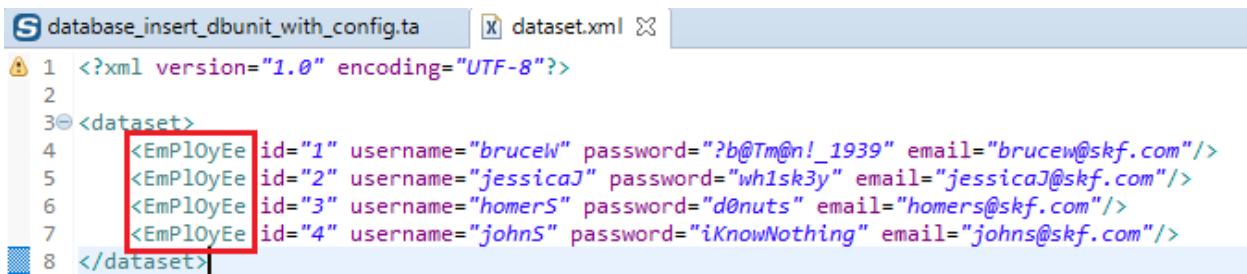
```
# INSERT_DBUNIT path/to/dataset.xml INTO my_database WITH CONFIG
path/to/my_config_dbunit.properties
```

DbUnit configuration file :

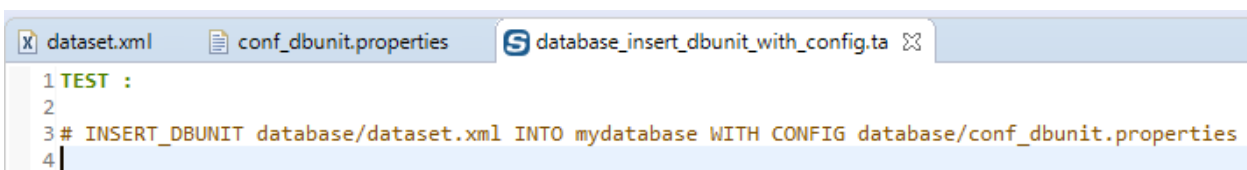


The table name is now case sensitive.

In dataset, we put capital letters in the table name :



SKF script :

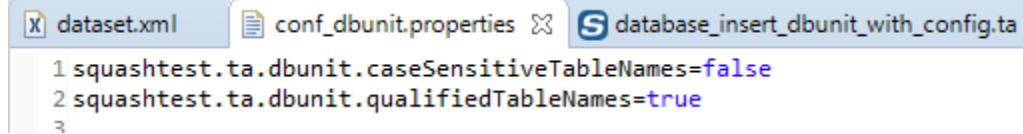


The execution raises an error :

Now we switch the property to “false” in the configuration file :

There is no error this time and users have been inserted in the database :

```
[ERROR] Table 'Emp10yEe' not found in tableMap=org.dbunit.dataset.OrderedTableNameMap[_tableNames=[employee, sys_config], _tableMap={sys_config=null, employee=null}, _caseSensitiveTableNames=true]
[ERROR] db unit insert : an error originated from the DbUnit framework occurred:
org.dbunit.dataset.NoSuchTableException: Emp10yEe
```



INSERT_DBUNIT {dataset} INTO {database} USING {ppkfilter}

What ?

This macro will insert all datas listed in the 'dataset file' into the 'database' using a DbUnit filter.

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand1}.file
CONVERT __temp{%%rand1}.file TO file(param.relativedate) AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO xml(structured) AS __temp{%%rand3}.xml
CONVERT __temp{%%rand3}.xml TO dataset.dbunit(dataset) AS __temp{%%rand4}.dbu

LOAD {ppkfilter} AS __temp{%%rand5}.file
CONVERT __temp{%%rand5}.file TO properties(structured) AS __temp{%%rand6}.props
CONVERT __temp{%%rand6}.props TO conf.dbunit.ppk(from.properties) AS __temp{%%rand7}.
↪ppk

EXECUTE insert WITH __temp{%%rand4}.dbu ON {database} USING __temp{%%rand7}.ppk,
↪$(operation:insert) AS __temp_{%%rand8}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys ('.properties').

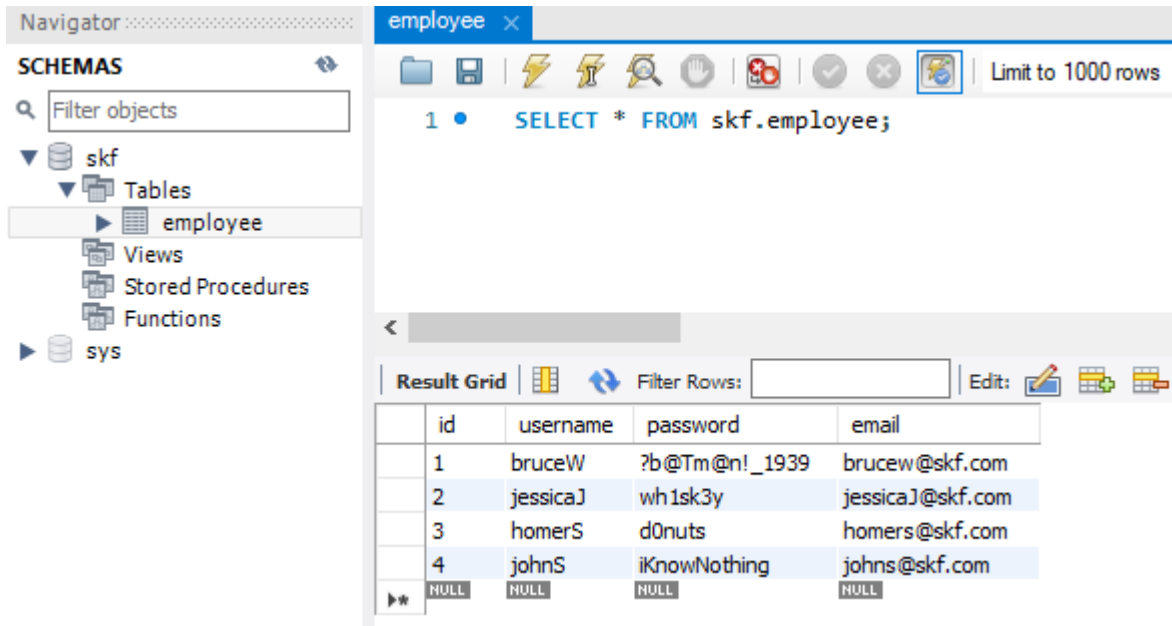
Example :

```
# INSERT_DBUNIT path/to/dataset.xml INTO my_database USING path/to/my_filter_dbunit.properties
```

INSERT_DBUNIT {dataset} INTO {database} WITH CONFIG {config} USING {ppkfilter}

What ?

This macro will insert all the data listed in the 'dataset file' into the 'database' using a DbUnit configuration file and a DbUnit filter.



Underlying instructions :

```

LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp{%%rand3}.file
CONVERT __temp{%%rand3}.file TO file(param,relativedate) AS __temp_{%%rand4}.file
CONVERT __temp_{%%rand4}.file TO xml(structured) AS __temp_{%%rand5}.xml
CONVERT __temp_{%%rand5}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand6}.dbu

LOAD {ppkfilter} AS __temp{%%rand7}.file
CONVERT __temp{%%rand7}.file TO properties(structured) AS __temp{%%rand8}.props
CONVERT __temp{%%rand8}.props TO conf.dbunit.ppk(from,properties) AS __temp{%%rand9}.
↪ppk

EXECUTE insert WITH __temp_{%%rand6}.dbu ON {database} USING __temp{%%rand9}.ppk,
↪$(operation:insert),__temp{config}{%%rand2}.conf AS __temp_{%%rand10}.result

```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {config} : A configuration file for DbUnit ('.properties').
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys ('.properties').

Example :

```

#      INSERT_DBUNIT      path/to/dataset.xml      INTO      my_database      WITH      CONFIG
path/to/my_config_dbunit.properties USING path/to/my_filter_dbunit.properties

```

Database Plugin - Macros - Delete DbUnit

Contents :

- `# DELETE_DBUNIT {dataset} FROM {database}`
- `# DELETE_DBUNIT {dataset} FROM {database} WITH CONFIG {config}`
- `# DELETE_DBUNIT {dataset} FROM {database} USING {ppkfilter}`
- `# DELETE_ALL_DBUNIT {dataset} FROM {database}`
- `# DELETE_ALL_DBUNIT {dataset} FROM {database} WITH CONFIG {config}`

DELETE_DBUNIT {dataset} FROM {database}

What ?

This macro will delete all the data listed in the 'dataset file' from the 'database'.

Underlying instructions :

```
LOAD {dataset} AS __temp_{%%rand1}.file

CONVERT __temp_{%%rand1}.file TO xml(structured) AS __temp_{%%rand2}.xml
CONVERT __temp_{%%rand2}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand3}.dbu

EXECUTE delete WITH __temp_{%%rand3}.dbu ON {database} USING $(operation : delete) AS_
↳ __temp_{%%rand4}.result
```

> Input :

- `{database}` : The name (in the context) of the database to use (database type target).
- `{dataset}` : A flat xml dbunit dataset file.

Remark : The file designed by `{dataset}` must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).

Example :

```
# DELETE_DBUNIT path/to/dataset.xml FROM my_database
```

Database overview :

Dataset .xml File :

SKF script :

There is only one employee left in the database :

Navigator

SCHEMAS

Filter objects

- skf
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- sys

employee

Limit to 1000 rows

1 • `SELECT * FROM skf.employee;`

Result Grid

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	bruceW@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homerS	d0nuts	homers@skf.com
4	4	johnS	iKnowNothing	johns@skf.com
▶▶	NULL	NULL	NULL	NULL

Package Explorer

- SKF
 - src
 - squashTA
 - repositories
 - resources
 - database
 - conf_dbunit.properties
 - dataset.xml
 - dataset2.xml

dataset2.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <employee id="1" username="bruceW" password="?b@Tm@n!_1939" email="bruceW@skf.com"/>
5   <employee id="2" username="jessicaJ" password="wh1sk3y" email="jessicaJ@skf.com"/>
6   <employee id="3" username="homerS" password="d0nuts" email="homers@skf.com"/>
7 </dataset>

```

dataset2.xml **database_delete_dbunit.ta**

```

1 TEST :
2
3 # DELETE_DBUNIT database/dataset2.xml FROM mydatabase

```

Navigator

SCHEMAS

Filter objects

- skf
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- sys

employee

Limit to 1000 rows

1 • `SELECT * FROM skf.employee;`

Result Grid

	id	username	password	email
4	4	johnS	iKnowNothing	johns@skf.com
▶▶	NULL	NULL	NULL	NULL

DELETE_DBUNIT {dataset} FROM {database} WITH CONFIG {config}

What ?

This macro will delete all the data listed in the ‘dataset file’ from the ‘database’ using a DbUnit configuration file.

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp_{%%rand3}.file

CONVERT __temp_{%%rand3}.file TO xml(structured) AS __temp_{%%rand4}.xml
CONVERT __temp_{%%rand4}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand5}.dbu

EXECUTE delete WITH __temp_{%%rand5}.dbu USING $(operation : delete),__temp{config}{%
↪%%rand2}.conf ON {database} AS __temp_{%%rand6}.result
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit (‘.properties’).

Remarks :

1. The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).
2. The file designed by {config} must respect the same rules than a file which would serve to create an SKF conf.dbunit type resource via the converter (*From file to conf.dbunit*).

Example :

```
# DELETE_DBUNIT path/to/dataset.xml FROM my_database WITH CONFIG
path/to/my_config_dbunit.properties
```

DELETE_DBUNIT {dataset} FROM {database} USING {ppkfilter}

What ?

This macro will load the specified xml dataset and delete datas listed in from the ‘database’ using a filter DbUnit.

Underlying instructions :

```
// Load and convert the dbunit dataset
LOAD {dataset} AS __{%r1}.file
CONVERT __{%r1}.file TO file(param.relativeDate) AS __{%r2}.file
CONVERT __{%r2}.file TO xml(structured) AS __{%r3}.xml
CONVERT __{%r3}.xml TO dataset.dbunit(dataset) AS __{%r4}.dbu

// Load and convert the pseudo primary key filter
LOAD {ppkfilter} AS __{%r5}.file
CONVERT __{%r5}.file TO properties(structured) AS __{%r6}.props
CONVERT __{%r6}.props TO conf.dbunit.ppk(from.properties) AS __{%r7}.ppk

// Execute delete operation using the pseudo primary key filter
EXECUTE delete WITH __{%r4}.dbu ON {database} USING __{%r7}.ppk,$(operation :_
↳delete) AS __{%r8}.result
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys (‘.properties’).

Example :

#	DELETE_DBUNIT	path/to/dataset.xml	FROM	my_database	USING
		path/to/my_filter_dbunit.properties			

For this example, we set the table employee with no primary key :

We set “username” as pseudo primary key in properties file :

Dataset .xml file :

We execute the macro without using ppk properties file :

The following error occurs :

We execute the macro with the ppk properties file :

The operation succeeds and all employees are deleted :

employee - Table

Table Name: Schema: **skf**

Charset/Collation: Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
username	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
password	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
email	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Navigator: SCHEMAS

Filter objects

- skf
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- sys

employee - Table employee

1 • **SELECT * FROM skf.employee;**

Result Grid

	id	username	password	email
▶	1	bruceW	?b@Tm@n!_1939	bruceW@skf.com
	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
	3	homerS	d0nuts	homers@skf.com
	4	johnS	iKnowNothing	johns@skf.com

ppkfilter.properties

```

1 employee=username
2

```

Package Explorer

- SKF
 - src
 - squashTA
 - repositories
 - resources
 - database
 - conf_dbunit.properties
 - dataset.xml

dataset.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <employee id="1" username="bruceW" password="?b@Tm@n!_1939" email="bruceW@skf.com"/>
5   <employee id="2" username="jessicaJ" password="wh1sk3y" email="jessicaJ@skf.com"/>
6   <employee id="3" username="homerS" password="d0nuts" email="homers@skf.com"/>
7   <employee id="4" username="johnS" password="iKnowNothing" email="johns@skf.com"/>
8 </dataset>

```

```

S database_delete_dbunit_using_ppkfilter.ta
1 TEST :
2
3 # DELETE_DBUNIT database/dataset.xml FROM mydatabase
4 |

```

[ERROR] db unit delete : an error occurred from within the DbUnit framework:
[org.dbunit.dataset.NoPrimaryKeyException](#): employee

```

S database_delete_dbunit_using_ppkfilter.ta
1 TEST :
2
3 # DELETE_DBUNIT database/dataset.xml FROM mydatabase USING database/ppkfilter.properties
4 |

```

```

[INFO] Beginning execution of test database_delete_dbunit_using_ppkfilter.ta
[WARN] Potential problem found: The configured data type factory 'class org.dbunit.dataset.datatype.DefaultDataTypeFactory' might cause proble
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190923_094136_2641677676082513936724
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

```

The screenshot shows a database management interface. On the left is a 'Navigator' pane with a tree view of 'SCHEMAS'. Under 'skf', there is a 'Tables' folder containing an 'employee' table. The main area on the right is titled 'employee - Table' and shows a query: `SELECT * FROM skf.employee;`. Below the query is a 'Result Grid' with columns: 'id', 'username', 'password', and 'email'. The grid is currently empty. There are various icons for file operations and a 'Limit to 10' option.

DELETE_ALL_DBUNIT {dataset} FROM {database}**What ?**

This macro will load the specified xml dataset and truncate every table listed in from the ‘database’.

Underlying instructions :

```
LOAD {dataset} AS __temp_{{rand1}}.file

CONVERT __temp_{{rand1}}.file TO xml(structured) AS __temp_{{rand2}}.xml
CONVERT __temp_{{rand2}}.xml TO dataset.dbunit(dataset) AS __temp_{{rand3}}.dbu

EXECUTE delete WITH __temp_{{rand3}}.dbu ON {database} AS __temp_{{rand4}}.result
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.

Remark : The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).

Example :

```
# DELETE_ALL_DBUNIT path/to/dataset.xml FROM my_database
```

DELETE_ALL_DBUNIT {dataset} FROM {database} WITH CONFIG {config}**What ?**

This macro will load the specified xml dataset and truncate every table listed in from the ‘database’ using a DbUnit configuration file.

Underlying instructions :

```
LOAD {config} AS __temp{config}{{rand1}}.file
CONVERT __temp{config}{{rand1}}.file TO conf.dbunit AS __temp{config}{{rand2}}.conf

LOAD {dataset} AS __temp_{{rand3}}.file

CONVERT __temp_{{rand3}}.file TO xml(structured) AS __temp_{{rand4}}.xml
CONVERT __temp_{{rand4}}.xml TO dataset.dbunit(dataset) AS __temp_{{rand5}}.dbu

EXECUTE delete WITH __temp_{{rand5}}.dbu USING __temp{config}{{rand2}}.conf ON
↳ {database} AS __temp_{{rand6}}.result
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit ('.properties').

Remarks :

1. The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).
2. The file designed by {config} must respect the same rules than a file which would serve to create an SKF conf.dbunit type resource via the converter (*From file to conf.dbunit*).

Example :

#	DELETE_ALL_DBUNIT	path/to/dataset.xml	FROM	my_database	WITH	CONFIG
	path/to/my_config_dbunit.properties					

Database Plugin - Macros - Refresh DbUnit

Contents :

- *# REFRESH_DBUNIT {dataset} INTO {database}*
- *# REFRESH_DBUNIT {dataset} INTO {database} WITH CONFIG {config}*
- *# REFRESH_DBUNIT {dataset} INTO {database} USING {ppkfilter}*
- *# REFRESH_DBUNIT {dataset} INTO {database} WITH CONFIG {config} USING {ppkfilter}*

REFRESH_DBUNIT {dataset} INTO {database}

What ?

This macro will refresh all the data listed in the 'dataset file' into the 'database'. Refresh operation means that data of existing rows are updated and non-existing row get inserted. Any rows which exist in the database but not in dataset stay unaffected.

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand1}.file

CONVERT __temp{%%rand1}.file TO file(param.relativedate) AS __temp_{%%rand2}.file
CONVERT __temp_{%%rand2}.file TO xml(structured) AS __temp_{%%rand3}.xml
```

(continues on next page)

(continued from previous page)

```

CONVERT __temp_{%%rand3}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand4}.dbu

EXECUTE insert WITH __temp_{%%rand4}.dbu ON {database} USING $(operation:refresh) AS _
↪ __temp_{%%rand5}.result

```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).

Remark : The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).

Example :

```
# REFRESH_DBUNIT path/to/dataset.xml INTO my_database
```

Database overview :

The screenshot shows a database overview window. On the left, a tree view displays the database structure: 'skf' (selected) contains 'Tables' (selected), which includes 'employee'. Other items in the tree are 'Views', 'Stored Procedures', 'Functions', and 'sys'. On the right, a 'Result Grid' displays data for the 'employee' table. The grid has columns for 'id', 'username', 'password', and 'email'. It shows four rows of employee data and a final row with 'NULL' values.

	id	username	password	email
▶	1	bruceW	?b@Tm@n!_1939	brucew@skf.com
	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
	3	homerS	d0nuts	homers@skf.com
	4	johnS	iKnowNothing	johns@skf.com
*	NULL	NULL	NULL	NULL

In the dataset, we update employees' information and add a new one :

The screenshot shows an IDE with a Package Explorer on the left and an XML editor on the right. The Package Explorer shows a project 'SKF' with a 'src' folder containing 'squashTA', which has subfolders 'repositories' and 'resources'. The 'resources' folder contains a 'database' subfolder with several XML files, including 'datasetRefresh.xml' (selected). The XML editor shows the content of 'datasetRefresh.xml', which is an XML document with a root element 'dataset' containing five 'employee' elements with attributes for id, username, password, and email.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <employee id="1" username="bruceW" password="r0b1n" email="bw@skf.com"/>
5   <employee id="2" username="jessicaJ" password="whatever" email="jj@skf.com"/>
6   <employee id="3" username="homerS" password="m@rge" email="hs@skf.com"/>
7   <employee id="4" username="johnS" password="iDoNothing" email="js@skf.com"/>
8   <employee id="5" username="peterP" password="@m@z1ng" email="pp@skf.com"/>
9 </dataset>

```

SKF script :

The employees are updated and the new one is inserted in the database :

The screenshot shows a database management interface. At the top, a SQL script is being edited in a file named `database_refresh_dbunit.ta`. The script contains the following lines:

```
1 TEST :
2
3 # REFRESH_DBUNIT database/datasetRefresh.xml INTO mydatabase
4
```

Below the script, a tree view on the left shows the database structure. The 'skf' database is selected, and the 'employee' table is highlighted under the 'Tables' folder. Other folders visible are 'Views', 'Stored Procedures', 'Functions', and 'sys'.

On the right, the 'Result Grid' for the 'employee' table is displayed. It shows 5 rows of data with columns: id, username, password, and email. The last row is marked with an asterisk and shows NULL values for all columns.

	id	username	password	email
▶	1	bruceW	r0b1n	bw@skf.com
	2	jessicaJ	whatever	jj@skf.com
	3	homerS	m@rge	hs@skf.com
	4	johnS	iDoNothing	js@skf.com
	5	peterP	@m@z1ng	pp@skf.com
*	NULL	NULL	NULL	NULL

REFRESH_DBUNIT {dataset} INTO {database} WITH CONFIG {config}

What ?

This macro will refresh all the data listed in the 'dataset file' into the 'database' using a DbUnit configuration file. Refresh operation means that data of existing rows are updated and non-existing row get inserted. Any rows which exist in the database but not in dataset stay unaffected.

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp{%%rand3}.file

CONVERT __temp{%%rand3}.file TO file(param.relativeDate) AS __temp_{%%rand4}.file
CONVERT __temp_{%%rand4}.file TO xml(structured) AS __temp_{%%rand5}.xml
CONVERT __temp_{%%rand5}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand6}.dbu

EXECUTE insert WITH __temp_{%%rand6}.dbu ON {database} USING $(operation:refresh),__
↳temp{config}{%%rand2}.conf AS __temp_{%%rand7}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).

- {config} : A configuration file for DbUnit ('.properties').

Remarks :

1. The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).
2. The file designed by {config} must respect the same rules than a file which would serve to create an SKF conf.dbunit type resource via the converter (*From file to conf.dbunit*).

Example :

```
# REFRESH_DBUNIT path/to/dataset.xml INTO my_database WITH CONFIG
path/to/my_config_dbunit.properties
```

REFRESH_DBUNIT {dataset} INTO {database} USING {ppkfilter}**What ?**

This macro will refresh all datas listed in the 'dataset file' into the 'database' using a DbUnit filter. Refresh operation means that data of existing rows are updated and non-existing row get inserted. Any rows which exist in the database but not in dataset stay unaffected.

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand1}.file
CONVERT __temp{%%rand1}.file TO file(param,relativedate) AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO xml(structured) AS __temp{%%rand3}.xml
CONVERT __temp{%%rand3}.xml TO dataset.dbunit(dataset) AS __temp{%%rand4}.dbu

LOAD {ppkfilter} AS __temp{%%rand5}.file
CONVERT __temp{%%rand5}.file TO properties(structured) AS __temp{%%rand6}.props
CONVERT __temp{%%rand6}.props TO conf.dbunit.ppk(from.properties) AS __temp{%%rand7}.
↪ppk

EXECUTE insert WITH __temp{%%rand4}.dbu ON {database} USING __temp{%%rand7}.ppk,
↪$(operation:refresh) AS __temp_{%%rand8}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys ('.properties').

Example :

```
# REFRESH_DBUNIT path/to/dataset.xml INTO my_database USING path/to/my_filter_dbunit.properties
```

REFRESH_DBUNIT {dataset} INTO {database} WITH CONFIG {config} USING {ppkfilter}

What ?

This macro will refresh all the data listed in the 'dataset file' into the 'database' using a DbUnit configuration file and a DbUnit filter. Refresh operation means that data of existing rows are updated and non-existing row get inserted. Any rows which exist in the database but not in dataset stay unaffected.

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp{%%rand3}.file
CONVERT __temp{%%rand3}.file TO file(param,relativedate) AS __temp_{%%rand4}.file
CONVERT __temp_{%%rand4}.file TO xml(structured) AS __temp_{%%rand5}.xml
CONVERT __temp_{%%rand5}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand6}.dbu

LOAD {ppkfilter} AS __temp{%%rand7}.file
CONVERT __temp{%%rand7}.file TO properties(structured) AS __temp{%%rand8}.props
CONVERT __temp{%%rand8}.props TO conf.dbunit.ppk(from,properties) AS __temp{%%rand9}.
↪ppk

EXECUTE insert WITH __temp_{%%rand6}.dbu ON {database} USING __temp{%%rand9}.ppk,
↪$(operation:refresh),__temp{config}{%%rand2}.conf AS __temp_{%%rand10}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {config} : A configuration file for DbUnit ('.properties').
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys ('.properties').

Example :

#	REFRESH_DBUNIT	path/to/dataset.xml	INTO	my_database	WITH	CONFIG
path/to/my_config_dbunit.properties USING path/to/my_filter_dbunit.properties						

Database Plugin - Macros - Update DbUnit

Contents :

- `# UPDATE_DBUNIT {dataset} INTO {database}`
- `# UPDATE_DBUNIT {dataset} INTO {database} WITH CONFIG {config}`
- `# UPDATE_DBUNIT {dataset} INTO {database} USING {ppkfilter}`
- `# UPDATE_DBUNIT {dataset} INTO {database} WITH CONFIG {config} USING {ppkfilter}`

UPDATE_DBUNIT {dataset} INTO {database}

What ?

This macro will update all the data listed in the ‘dataset file’ into the ‘database’. This update operation assumes that table data already exists in the target database and fails if this is not the case.

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand1}.file

CONVERT __temp{%%rand1}.file TO file(param.relativeDate) AS __temp_{%%rand2}.file
CONVERT __temp_{%%rand2}.file TO xml(structured) AS __temp_{%%rand3}.xml
CONVERT __temp_{%%rand3}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand4}.dbu

EXECUTE insert WITH __temp_{%%rand4}.dbu ON {database} USING $(operation:update) AS __
↳temp_{%%rand5}.result
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.

Remark : The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit type resource via the converter (*From xml to dataset.dbunit*).

Example :

```
# UPDATE_DBUNIT path/to/dataset.xml INTO my_database
```

Database overview :

In the dataset, we update employees’ information :

SKF script :

All employees are updated :

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	brucew@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homerS	d0nuts	homers@skf.com
4	4	johnS	iKnowNothing	johns@skf.com
»	NULL	NULL	NULL	NULL

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dataset>
4   <employee id="1" username="bruceW" password="r0b1n" email="bw@skf.com"/>
5   <employee id="2" username="jessicaJ" password="whatever" email="jj@skf.com"/>
6   <employee id="3" username="homerS" password="m@rge" email="hs@skf.com"/>
7   <employee id="4" username="johnS" password="iDoNothing" email="js@skf.com"/>
8 </dataset>

```

```

1 TEST :
2
3 # UPDATE_DBUNIT database/datasetUpdate.xml INTO mydatabase
4

```

	id	username	password	email
1	1	bruceW	r0b1n	bw@skf.com
2	2	jessicaJ	whatever	jj@skf.com
3	3	homerS	m@rge	hs@skf.com
4	4	johnS	iDoNothing	js@skf.com
»	NULL	NULL	NULL	NULL

UPDATE_DBUNIT {dataset} INTO {database} WITH CONFIG {config}**What ?**

This macro will update all the data listed in the ‘dataset file’ into the ‘database’ using a DbUnit configuration file. This update operation assumes that table data already exists in the target database and fails if this is not the case.

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp{%%rand3}.file

CONVERT __temp{%%rand3}.file TO file(param.relativeDate) AS __temp_{%%rand4}.file
CONVERT __temp_{%%rand4}.file TO xml(structured) AS __temp_{%%rand5}.xml
CONVERT __temp_{%%rand5}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand6}.dbu

EXECUTE insert WITH __temp_{%%rand6}.dbu ON {database} USING $(operation:update),__
↳temp{config}{%%rand2}.conf AS __temp_{%%rand7}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {config} : A configuration file for DbUnit (‘.properties’).

Remarks :

1. The file designed by {dataset} must respect the same rules than a file which would serve to create an SKF dataset.dbunit via the converter (*From xml to dataset.dbunit*).
2. The file designed by {config} must respect the same rules than a file which would serve to create an SKF conf.dbunit type resource via the converter (*From file to conf.dbunit*).

Example :

#	UPDATE_DBUNIT	path/to/dataset.xml	INTO	my_database	WITH	CONFIG
	path/to/my_config_dbunit.properties					

UPDATE_DBUNIT {dataset} INTO {database} USING {ppkfilter}**What ?**

This macro will update all datas listed in the ‘dataset file’ into the ‘database’ using a DbUnit filter. This update operation assumes that table data already exists in the target database and fails if this is not the case.

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand1}.file
CONVERT __temp{%%rand1}.file TO file(param.relativedate) AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO xml(structured) AS __temp{%%rand3}.xml
CONVERT __temp{%%rand3}.xml TO dataset.dbunit(dataset) AS __temp{%%rand4}.dbu

LOAD {ppkfilter} AS __temp{%%rand5}.file
CONVERT __temp{%%rand5}.file TO properties(structured) AS __temp{%%rand6}.props
CONVERT __temp{%%rand6}.props TO conf.dbunit.ppk(from.properties) AS __temp{%%rand7}.
↪ppk

EXECUTE insert WITH __temp{%%rand4}.dbu ON {database} USING __temp{%%rand7}.ppk,
↪$(operation:update) AS __temp_{%%rand8}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys ('.properties').

Example :

```
# UPDATE_DBUNIT path/to/dataset.xml INTO my_database USING path/to/my_filter_dbunit.properties
```

UPDATE_DBUNIT {dataset} INTO {database} WITH CONFIG {config} USING {ppkfilter}

What ?

This macro will update all the data listed in the 'dataset file' into the 'database' using a DbUnit configuration file and a DbUnit filter. This update operation assumes that table data already exists in the target database and fails if this is not the case.

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

LOAD {dataset} AS __temp{%%rand3}.file
CONVERT __temp{%%rand3}.file TO file(param.relativedate) AS __temp_{%%rand4}.file
CONVERT __temp_{%%rand4}.file TO xml(structured) AS __temp_{%%rand5}.xml
CONVERT __temp_{%%rand5}.xml TO dataset.dbunit(dataset) AS __temp_{%%rand6}.dbu

LOAD {ppkfilter} AS __temp{%%rand7}.file
CONVERT __temp{%%rand7}.file TO properties(structured) AS __temp{%%rand8}.props
CONVERT __temp{%%rand8}.props TO conf.dbunit.ppk(from.properties) AS __temp{%%rand9}.
↪ppk
```

(continues on next page)

(continued from previous page)

```
EXECUTE insert WITH __temp_{%%rand6}.dbu ON {database} USING __temp{%%rand9}.ppk,
→$(operation:update),__temp{config}{%%rand2}.conf AS __temp_{%%rand10}.result
```

> Input :

- {dataset} : A flat xml dbunit dataset file.
- {database} : The name (in the context) of the database to use (database type target).
- {config} : A configuration file for DbUnit ('.properties').
- {ppkfilter} : A DbUnit filter referring to pseudo primary keys ('.properties').

Example :

```
#      UPDATE_DBUNIT      path/to/dataset.xml      INTO      my_database      WITH      CONFIG
path/to/my_config_dbunit.properties USING path/to/my_filter_dbunit.properties
```

Database Plugin - Macros - Assert DbUnit**Contents :**

- *# ASSERT_DBUNIT TARGET {database} CONTAINS {dataset}*
- *# ASSERT_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config}*
- *# ASSERT_DBUNIT TARGET {database} CONTAINS {dataset} WITH FILTER {filter}*
- *# ASSERT_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config} AND FILTER {filter}*
- *# ASSERT_DBUNIT TARGET {database} EQUALS {dataset}*
- *# ASSERT_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config}*
- *# ASSERT_DBUNIT TARGET {database} EQUALS {dataset} WITH FILTER {filter}*
- *# ASSERT_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config} AND FILTER {filter}*

ASSERT_DBUNIT TARGET {database} CONTAINS {dataset}**What ?**

This macro will check that all the data listed in the 'dataset file' exist in the 'database'. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
EXECUTE get.all WITH $() ON {database} AS __actual_{{%rand1}}.dbu

LOAD {dataset} AS __temp{{%rand2}}.file
CONVERT __temp{{%rand2}}.file TO file(param,relativedate) AS __temp_{{%rand3}}.file
CONVERT __temp_{{%rand3}}.file TO xml(structured) AS __temp_{{%rand4}}.xml
CONVERT __temp_{{%rand4}}.xml TO dataset.dbunit(dataset) AS __expected_{{%rand5}}.dbu

ASSERT __actual_{{%rand1}}.dbu DOES contain THE __expected_{{%rand5}}.dbu
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.

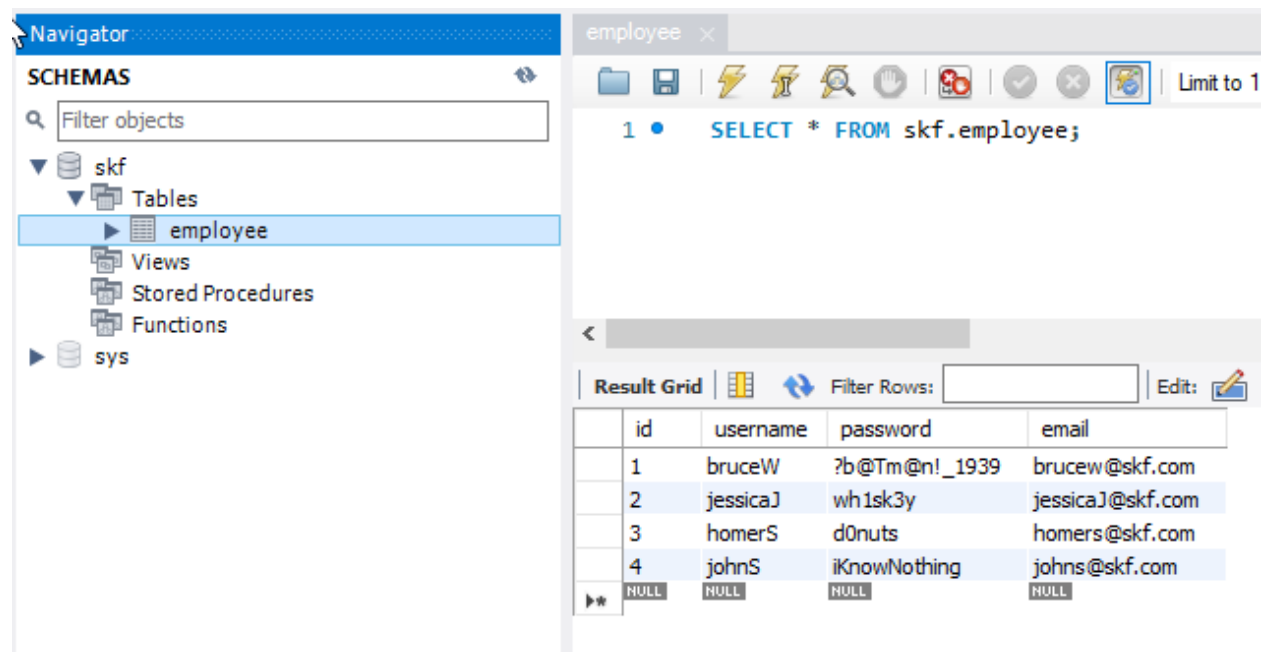
Note :

1. If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativedate*), those ones are calculated and replaced by the value.
2. No need to put all tables.
3. No need to put all lines of specified tables.
4. No need to put all columns of specified tables.

Example :

```
# ASSERT_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml
```

Database overview :

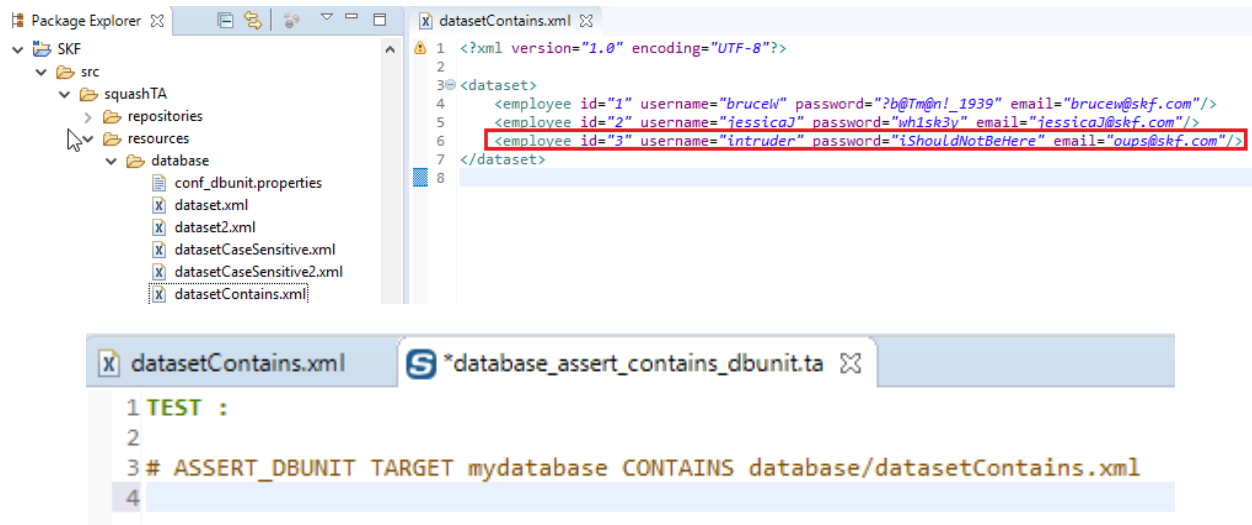


The screenshot shows a database management interface. On the left, a 'Navigator' pane displays a tree structure of schemas. The 'skf' schema is expanded, showing 'Tables' with 'employee' selected. The main area shows the 'employee' table with a toolbar and a 'Result Grid'. The SQL query 'SELECT * FROM skf.employee;' is entered. The result grid displays the following data:

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	brucew@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homersS	d0nuts	homers@skf.com
4	4	johnS	iKnowNothing	johns@skf.com
5	NULL	NULL	NULL	NULL

We add a new employee in dataset who does not exist in database :

SKF script :



We obtain the following error :

```
[ERROR] The execution failed in the TEST phase of the TA script 'database_assert_contains_dbunit.ta' with the message: 'The first dataset did not contain the second one.'
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190923_142657_6301316913687014492379
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
```

We delete the new employee from the dataset :

The assertion is true and you should obtain a build success.

ASSERT_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config}

What ?

This macro will check that all the data listed in the 'dataset file' exist in the 'database' using a DbUnit configuration file. For differences between ASSERT and VERIFY assertion mode see [this page](#).

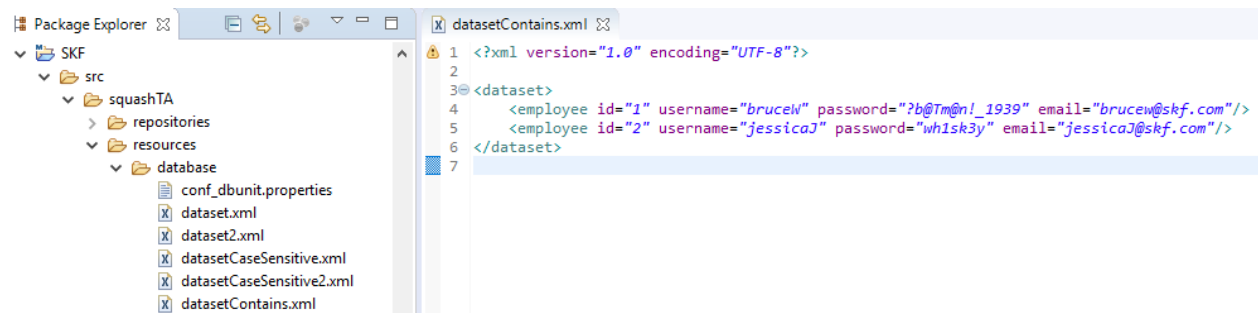
Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
  ↳ {%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativedate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
```

(continues on next page)



(continued from previous page)

```
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu
ASSERT __actual_{%%rand3}.dbu DOES contain THE __expected_{%%rand7}.dbu
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit ('.properties').

Note :

1. If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativeDate*), those ones are calculated and replaced by the value.
2. No need to put all tables.
3. No need to put all lines of specified tables.
4. No need to put all columns of specified tables.

Example :

```
# ASSERT_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml WITH CONFIG
path/to/my_config_dbunit.properties
```

ASSERT_DBUNIT TARGET {database} CONTAINS {dataset} WITH FILTER {filter}

What ?

This macro will check that all the data listed in the 'dataset file' exist in the 'database' using a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```

LOAD {dataset} AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO file(param.relativedate) AS __temp_{%%rand3}.file
CONVERT __temp_{%%rand3}.file TO xml(structured) AS __temp_{%%rand4}.xml
CONVERT __temp_{%%rand4}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand5}.dbu

LOAD {filter} AS __filter_{%%rand6}.file
CONVERT __filter_{%%rand6}.file TO filter.dbunit(filter) AS __filter_{%%rand7}.filter

ASSERT __actual_{%%rand1}.dbu DOES contain THE __expected_{%%rand5}.dbu USING __
↪filter_{%%rand7}.filter

```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {filter} : A Dbunit filter (filter.dbunit SKF resource).

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativedate*), those ones are calculated and replaced by the value.

Example :

```

# ASSERT_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml WITH FILTER
path/to/my_dbunit_filter.xml

```

ASSERT_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config} AND FILTER {filter}

What ?

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ using a DbUnit configuration file and a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```

LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↪{%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativedate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

```

(continues on next page)

(continued from previous page)

```
LOAD {filter} AS __filter_{{%rand8}}.file
CONVERT __filter_{{%rand8}}.file TO filter.dbunit(filter) AS __filter_{{%rand9}}.filter

ASSERT __actual_{{%rand3}}.dbu DOES contain THE __expected_{{%rand7}}.dbu USING __
↪filter_{{%rand9}}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset.
- {config} : A configuration file for DbUnit ('.properties'). It should be a 'conf.dbunit' SKF resource.
- {filter} : A Dbunit filter xml file. It should be a 'filter.dbunit' SKF resource.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativedate*), those ones are calculated and replaced by the value.

Example :

```
# ASSERT_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml WITH CONFIG
path/to/my_dbunit_config.properties AND FILTER path/to/my_dbunit_filter.xml
```

ASSERT_DBUNIT TARGET {database} EQUALS {dataset}

What ?

This macro will check that all the data listed in the 'dataset file' exist in the 'database' and the reverse. For differences between ASSERT and VERIFY assertion mode see *this page*.

Underlying instructions :

```
EXECUTE get.all WITH $() ON {database} AS __actual_{{%rand1}}.dbu

LOAD {dataset} AS __temp{{%rand2}}.file
CONVERT __temp{{%rand2}}.file TO file(param.relativedate) AS __temp_{{%rand3}}.file
CONVERT __temp_{{%rand3}}.file TO xml(structured) AS __temp_{{%rand4}}.xml
CONVERT __temp_{{%rand4}}.xml TO dataset.dbunit(dataset) AS __expected_{{%rand5}}.dbu

ASSERT __actual_{{%rand1}}.dbu IS equal THE __expected_{{%rand5}}.dbu
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relative date*), those ones are calculated and replaced by the value.

Example :

```
# ASSERT_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml
```

Database overview :

The screenshot shows a database management interface. On the left, a 'Navigator' pane shows the 'skf' database schema with tables, views, stored procedures, and functions. The 'employee' table is selected. On the right, a SQL editor shows the query 'SELECT * FROM skf.employee;'. Below the editor, a 'Result Grid' displays the data from the 'employee' table.

	id	username	password	email
1	1	bruceW	?b@Tm@n!_1939	brucew@skf.com
2	2	jessicaJ	wh1sk3y	jessicaJ@skf.com
3	3	homerS	d0nuts	homers@skf.com
4	4	johnS	iKnowNothing	johns@skf.com
	NULL	NULL	NULL	NULL

We use a dataset containing less employees than the database :

The screenshot shows a 'Package Explorer' on the left with a tree view of the project structure. The 'datasetEqualsKo.xml' file is selected in the 'database' folder. The right pane shows the XML content of the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <employee id="1" username="bruceW" password="?b@Tm@n!_1939" email="brucew@skf.com"/>
  <employee id="2" username="jessicaJ" password="wh1sk3y" email="jessicaJ@skf.com"/>
  <employee id="3" username="homerS" password="d0nuts" email="homers@skf.com"/>
</dataset>
```

SKF script :

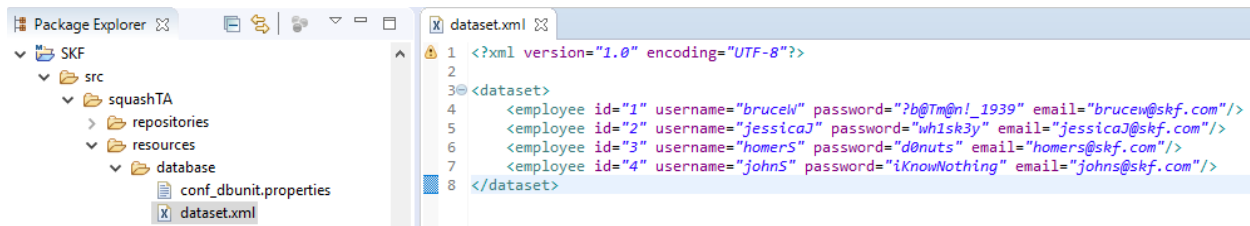
```
datasetEqualsKo.xml  S *database_assert_equals_dbunit.ta
1 TEST :
2
3 # ASSERT_DBUNIT TARGET mydatabase EQUALS database/datasetEqualsKo.xml
4
```

We obtain the following error :

```
[WARN] ERROR_MESSAGE
org.dbunit.dataset.DataSetException: Table 'employee' contains 1 more row(s) in the first dataset than in the second one.

[ERROR] The execution failed in the TEST phase of the TA script 'database_assert_equals_dbunit.ta' with the message: 'The two compared datasets are different.'.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190923_151033_4195916812012177114942
[INFO] -----
[INFO] BUILD FAILURE
```

We use a dataset containing exactly all the employees :



We execute the SKF script with the new dataset :

The assertion is true and you should obtain a build success.

ASSERT_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config}

What ?

This macro will check that all the data listed in the 'dataset file' exist in the 'database' and the reverse using a DbUnit configuration file. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

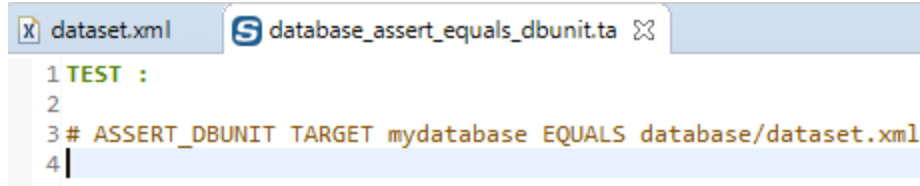
EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↳ {%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativedate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

ASSERT __actual_{%%rand3}.dbu IS equal THE __expected_{%%rand7}.dbu
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.



- {config} : A configuration file for DbUnit ('.properties').

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relative date*), those ones are calculated and replaced by the value.

Example :

```
# ASSERT_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml WITH CONFIG
path/to/my_config_dbunit.properties
```

ASSERT_DBUNIT TARGET {database} EQUALS {dataset} WITH FILTER {filter}

What ?

This macro will check that all the data listed in the 'dataset file' exist in the 'database' and the reverse using a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see *this page*.

Underlying instructions :

```
EXECUTE get.all WITH $() ON {database} AS __actual_{{%rand1}}.dbu

LOAD {dataset} AS __temp_{{%rand2}}.file
CONVERT __temp_{{%rand2}}.file TO file(param.relative date) AS __temp_{{%rand3}}.file
CONVERT __temp_{{%rand3}}.file TO xml(structured) AS __temp_{{%rand4}}.xml
CONVERT __temp_{{%rand4}}.xml TO dataset.dbunit(dataset) AS __expected_{{%rand5}}.dbu

LOAD {filter} AS __filter_{{%rand6}}.file
CONVERT __filter_{{%rand6}}.file TO filter.dbunit(filter) AS __filter_{{%rand7}}.filter

ASSERT __actual_{{%rand1}}.dbu IS equal THE __expected_{{%rand5}}.dbu USING __filter_{{%
↪ %rand7}}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {filter} : A DbUnit filter xml file. It should be a 'filter.dbunit' SKF resource.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relative date*), those ones are calculated and replaced by the value.

Example :

```
# ASSERT_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml WITH FILTER
path/to/my_dbunit_filter.xml
```

ASSERT_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config} AND FILTER {filter}

What ?

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ and the reverse using a DbUnit configuration file and a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↪{%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativedate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

LOAD {filter} AS __filter_{%%rand8}.file
CONVERT __filter_{%%rand8}.file TO filter.dbunit(filter) AS __filter_{%%rand9}.filter

ASSERT __actual_{%%rand3}.dbu IS equal THE __expected_{%%rand7}.dbu USING __filter_{%
↪%rand9}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit (‘.properties’). It should be a ‘conf.dbunit’ SKF resource.
- {filter} : A DbUnit filter xml file. It should be a ‘filter.dbunit’ SKF resource.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativedate*), those ones are calculated and replaced by the value.

Example :

```
# ASSERT_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml WITH CONFIG
path/to/my_dbunit_config.properties AND FILTER path/to/my_dbunit_filter.xml
```

Database Plugin - Macros - Verify DbUnit

Contents :

- `# VERIFY_DBUNIT TARGET {database} CONTAINS {dataset}`
- `# VERIFY_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config}`
- `# VERIFY_DBUNIT TARGET {database} CONTAINS {dataset} WITH FILTER {filter}`
- `# VERIFY_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config} AND FILTER {filter}`
- `# VERIFY_DBUNIT TARGET {database} EQUALS {dataset}`
- `# VERIFY_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config}`
- `# VERIFY_DBUNIT TARGET {database} EQUALS {dataset} WITH FILTER {filter}`
- `# VERIFY_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config} AND FILTER {filter}`

VERIFY_DBUNIT TARGET {database} CONTAINS {dataset}

What ?

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
EXECUTE get.all WITH $() ON {database} AS __actual_{%%rand1}.dbu

LOAD {dataset} AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO file(param.relativeDate) AS __temp_{%%rand3}.file
CONVERT __temp_{%%rand3}.file TO xml(structured) AS __temp_{%%rand4}.xml
CONVERT __temp_{%%rand4}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand5}.dbu

VERIFY __actual_{%%rand1}.dbu DOES contain THE __expected_{%%rand5}.dbu
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter [From file to file via param.relativeDate](#)), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml
```

This macro is very similar to the ASSERT macro. For more information, please check the following [page](#).

VERIFY_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config}

What ?

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ using a DbUnit configuration file. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↳{%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativeDate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

VERIFY __actual_{%%rand3}.dbu DOES contain THE __expected_{%%rand7}.dbu
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit (‘.properties’).

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter [From file to file via param.relativeDate](#)), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml WITH CONFIG
path/to/my_config_dbunit.properties
```

VERIFY_DBUNIT TARGET {database} CONTAINS {dataset} WITH FILTER {filter}**What ?**

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ using a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
LOAD {dataset} AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO file(param.relativeDate) AS __temp_{%%rand3}.file
CONVERT __temp_{%%rand3}.file TO xml(structured) AS __temp_{%%rand4}.xml
CONVERT __temp_{%%rand4}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand5}.dbu

LOAD {filter} AS __filter_{%%rand6}.file
CONVERT __filter_{%%rand6}.file TO filter.dbunit(filter) AS __filter_{%%rand7}.filter

VERIFY __actual_{%%rand1}.dbu DOES contain THE __expected_{%%rand5}.dbu USING __
↪filter_{%%rand7}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {filter} : A Dbunit filter (filter.dbunit TA resource).

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativeDate*), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml WITH FILTER
path/to/my_dbunit_filter.xml
```

VERIFY_DBUNIT TARGET {database} CONTAINS {dataset} WITH CONFIG {config} AND FILTER {filter}**What ?**

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ using a DbUnit configuration file and a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↪{%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativedate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

LOAD {filter} AS __filter_{%%rand8}.file
CONVERT __filter_{%%rand8}.file TO filter.dbunit(filter) AS __filter_{%%rand9}.filter

VERIFY __actual_{%%rand3}.dbu DOES contain THE __expected_{%%rand7}.dbu USING __
↪filter_{%%rand9}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit ('.properties'). It should be a 'conf.dbunit' SKF resource.
- {filter} : A Dbunit filter xml file. It should be a 'filter.dbunit' SKF resource.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativedate*), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database CONTAINS path/to/my_dataset.xml WITH CONFIG
path/to/my_dbunit_config.properties AND FILTER path/to/my_dbunit_filter.xml
```

VERIFY_DBUNIT TARGET {database} EQUALS {dataset}

What ?

This macro will check that all the data listed in the 'dataset file' exist in the 'database' and the reverse. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
EXECUTE get.all WITH $() ON {database} AS __actual_{%%rand1}.dbu

LOAD {dataset} AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO file(param.relativedate) AS __temp_{%%rand3}.file
CONVERT __temp_{%%rand3}.file TO xml(structured) AS __temp_{%%rand4}.xml
```

(continues on next page)

(continued from previous page)

```

CONVERT __temp_{%%rand4}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand5}.dbu
VERIFY __actual_{%%rand1}.dbu IS equal THE __expected_{%%rand5}.dbu

```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relative date*), those ones are calculated and replaced by the value.

Example :

```
# ASSERT_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml
```

This macro is very similar to the ASSERT macro. For more information, please check the following [page](#).

VERIFY_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config}**What ?**

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ and the reverse using a DbUnit configuration file. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```

LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↪{%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relative date) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

VERIFY __actual_{%%rand3}.dbu IS equal THE __expected_{%%rand7}.dbu

```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit (‘.properties’).

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relative date*), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml WITH CONFIG
path/to/my_config_dbunit.properties
```

VERIFY_DBUNIT TARGET {database} EQUALS {dataset} WITH FILTER {filter}

What ?

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ and the reverse using a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see *this page*.

Underlying instructions :

```
EXECUTE get.all WITH $() ON {database} AS __actual_{{%rand1}}.dbu

LOAD {dataset} AS __temp_{{%rand2}}.file
CONVERT __temp_{{%rand2}}.file TO file(param.relative date) AS __temp_{{%rand3}}.file
CONVERT __temp_{{%rand3}}.file TO xml(structured) AS __temp_{{%rand4}}.xml
CONVERT __temp_{{%rand4}}.xml TO dataset.dbunit(dataset) AS __expected_{{%rand5}}.dbu

LOAD {filter} AS __filter_{{%rand6}}.file
CONVERT __filter_{{%rand6}}.file TO filter.dbunit(filter) AS __filter_{{%rand7}}.filter

VERIFY __actual_{{%rand1}}.dbu IS equal THE __expected_{{%rand5}}.dbu USING __filter_{{%
↪ rand7}}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {filter} : A DbUnit filter xml file. It should be a ‘filter.dbunit’ SKF resource.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relative date*), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml WITH FILTER
path/to/my_dbunit_filter.xml
```

```
# VERIFY_DBUNIT TARGET {database} EQUALS {dataset} WITH CONFIG {config} AND FILTER {filter}
```

What ?

This macro will check that all the data listed in the ‘dataset file’ exist in the ‘database’ and the reverse using a DbUnit configuration file and a DbUnit filter. For differences between ASSERT and VERIFY assertion mode see [this page](#).

Underlying instructions :

```
LOAD {config} AS __temp{config}{%%rand1}.file
CONVERT __temp{config}{%%rand1}.file TO conf.dbunit AS __temp{config}{%%rand2}.conf

EXECUTE get.all WITH $() ON {database} USING __temp{config}{%%rand2}.conf AS __actual_
↳{%%rand3}.dbu

LOAD {dataset} AS __temp{%%rand4}.file
CONVERT __temp{%%rand4}.file TO file(param.relativeDate) AS __temp_{%%rand5}.file
CONVERT __temp_{%%rand5}.file TO xml(structured) AS __temp_{%%rand6}.xml
CONVERT __temp_{%%rand6}.xml TO dataset.dbunit(dataset) AS __expected_{%%rand7}.dbu

LOAD {filter} AS __filter_{%%rand8}.file
CONVERT __filter_{%%rand8}.file TO filter.dbunit(filter) AS __filter_{%%rand9}.filter

VERIFY __actual_{%%rand3}.dbu IS equal THE __expected_{%%rand7}.dbu USING __filter_{%
↳%%rand9}.filter
```

> Input :

- {database} : The name (in the context) of the database to use (database type target).
- {dataset} : A flat xml dbunit dataset file.
- {config} : A configuration file for DbUnit (‘.properties’). It should be a ‘conf.dbunit’ TA resource.
- {filter} : A DbUnit filter xml file. It should be a ‘filter.dbunit’ TA resource.

Remark : If the file designed by {dataset} contains formulas of date calculation (See the converter *From file to file via param.relativeDate*), those ones are calculated and replaced by the value.

Example :

```
# VERIFY_DBUNIT TARGET my_database EQUALS path/to/my_dataset.xml WITH CONFIG
path/to/my_dbunit_config.properties AND FILTER path/to/my_dbunit_filter.xml
```

6.2.5 Database Plugin - Advanced Users

Database Plugin - Converters

Contents :

- *From file ...*
 - ... *to conf.dbunit*
 - ... *to conf.dbunit.ppk*
 - ... *to parameter.indexed.sql*
 - ... *to parameter.named.sql*
 - ... *to query.sql*
 - ... *to script.sql*
- *From directory to dataset.dbunit*
- *From properties to conf.dbunit.ppk*
- *From result.sql to dataset.dbunit*
- *From xml ...*
 - ... *to dataset.dbunit*
 - ... *to filter.dbunit*

From file ...

... to conf.dbunit

Category-Name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a `conf.dbunit` type resource.

CONVERT {resourceToConvert<Res:file>} TO conf.dbunit (structured) AS {converted<Res:conf.dbunit>}

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a configuration file for DbUnit. This file must be a `.properties` file (list of properties key / value using '=' like separator).

> Output :

- `converted<Res:conf.dbunit>` : The name of the converted resource (`conf.dbunit` type resource).

Example :

```
LOAD path/to/dbunit-configuration.properties AS dbunit-conf.file
CONVERT dbunit-conf.file TO conf.dbunit (structured) AS conf
```

The DbUnit “features & properties” supported are : (See [DbUnit documentation](#))

Batched statements :

SKF name	<code>squashtest.ta.dbunit.batchedStatements</code>
DbUnit name	http://www.dbunit.org/features/batchedStatements
Default value	<code>false</code>
Meaning	Enable or disable the use of batch JDBC requests.

Case sensitive table names :

SKF name	<code>squashtest.ta.dbunit.caseSensitiveTableNames</code>
DbUnit name	http://www.dbunit.org/features/caseSensitiveTableNames
Default value	<code>false</code>
Meaning	Enable or disable the case sensitivity of table names. When this property is activ, table names are considered case sensitive.

Qualified table names :

SKF name	<code>squashtest.ta.dbunit.qualifiedTableNames</code>
DbUnit name	http://www.dbunit.org/features/qualifiedTableNames
De-fault value	<code>false</code>
Meaning	Enable or disable the possibility of taking in charge simultaneously several schemas. When this property is enabled tables names are preceded by the schema name to which they belong : <code>SCHEME.TABLE</code> .

Table types :

SKF name	<code>squashtest.ta.dbunit.tableType</code>
DbUnit name	http://www.dbunit.org/properties/tableType
Default value	<code>String[]{"TABLE"}</code>
Meaning	Permits to configure the type of known tables.

Datatype factory (Cf. issue 789) :

SKF name	squashtest.ta.dbunit.datatypeFactory
DbUnit name	http://www.dbunit.org/properties/datatypeFactory
Default value	org.dbunit.dataset.datatype.DefaultDataTypeFactory
Meaning	<p>Some datas types are specific to the management of relational database.</p> <p>To allow DbUnit to manage this kind of datas, it's necessary to specify the "Datas types Factory" he must use.</p> <p>The following factories are available in DbUnit :</p> <ul style="list-style-type: none"> • org.dbunit.ext.db2.Db2DataTypeFactory • org.dbunit.ext.h2.H2DataTypeFactory • org.dbunit.ext.hsqldb.HsqldbDataTypeFactory • org.dbunit.ext.mckoi.MckoiDataTypeFactory • org.dbunit.ext.mssql.MsSqlDataTypeFactory • org.dbunit.ext.mysql.MySqlDataTypeFactory • org.dbunit.ext.oracle.OracleDataTypeFactory • org.dbunit.ext.oracle.Oracle10DataTypeFactory • org.dbunit.ext.postgresql.PostgresqlDataTypeFactory • org.dbunit.ext.netezza.NetezzaDataTypeFactory

Batch size :

SKF name	squashtest.ta.dbunit.batchSize
DbUnit name	http://www.dbunit.org/properties/batchSize
Default value	100
Meaning	Integer representing the requests number in a batch requests (Only when the property <code>batchedStatements</code> is active)

Metadata handler :

SKF name	squashtest.ta.dbunit.metadataHandler
DbUnit name	http://www.dbunit.org/properties/metadataHandler
Default value	org.dbunit.database.DefaultMetadataHandler
Meaning	<p>The way of metadatas management of the base can differ according to the SGBDR.</p> <p>The following handlers are available :</p> <ul style="list-style-type: none"> • org.dbunit.ext.db2.Db2MetadataHandler • org.dbunit.ext.mysql.MySqlMetadataHandler • org.dbunit.ext.netezza.NetezzaMetadataHandler <p>For others SGBDR, default handler is enough.</p>

Escape pattern :

SKF name	squashtest.ta.dbunit.escapePattern
DbUnit name	http://www.dbunit.org/properties/escapePattern
Default value	none
Meaning	Allows schema, table and column names escaping.
Example	squashtest.tf.dbunit.escapePattern= The property above will permit to escape the table name and column names in the following query. insert into 'person' ('id', 'name', 'unique') values (1, 'Doe', true); This query will succeed even though “unique” is a SQL key word and is not normally allowed.

Remark

The DbUnit property: <http://www.dbunit.org/properties/primaryKeyFilter> exist via the category of SKF resource: `conf.dbunit.ppk`.

... to `conf.dbunit.ppk`

Category-Name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a “`conf.dbunit.ppk`” type resource.

CONVERT	{resourceToConvert<Res:file>}	TO	conf.dbunit.ppk	(structured)	AS	{converted<Res:conf.dbunit.ppk>}
---------	-------------------------------	----	-----------------	--------------	----	----------------------------------

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a configuration file to define the pseudo primary keys. This configuration file must be of type `.properties` (for each property, the key is the name of a Table, the value is the name of a column or a list of columns separated with comma and the '=' character is used like separator).

> Output :

- `converted<Res:conf.dbunit.ppk>` : The name of the converted resource (Resource of type `conf.dbunit.ppk`).

Example :

```
LOAD path/to/valid_ppk.properties AS ppk.file
CONVERT ppk.file TO properties (structured) AS ppk.properties
CONVERT ppk.properties TO conf.dbunit.ppk (from.properties) AS ppk
```

... to parameter.indexed.sql

Category-Name : *from.text*

What ?

This *from.text* converter will convert a `file` type resource to a `parameter.indexed.sql` type resource.

```
CONVERT {resourceToConvert<Res:file>} TO parameter.indexed.sql (from.text) AS {converted<Res:parameter.indexed.sql>}
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a file which each line defines the value of a sql query parameter. Each line contains two character strings separated with the character '=' :
 - The first character string corresponds to the parameter position in the SQL query.
 - The Second one corresponds to the value.
-

Remark

None of the two character strings can be empty.

> Output :

- `converted<Res:parameter.indexed.sql>` : The name of the converted resource (`parameter.indexed.sql` type resource).

Example :

```
LOAD path/to/parameter-indexed_value.properties AS value.file
CONVERT value.file TO parameter.indexed.sql (from.text) AS value.properties
```

... to parameter.named.sql**Category-Name** : *from.text***What ?**

This *from.text* converter will convert a `file` type resource to a `parameter.named.sql` type resource.

CONVERT {resourceToConvert<Res:file>} TO parameter.named.sql (from.text) AS {converted<Res:parameter.named.sql>}
--

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a file which each line defines the value of a sql query parameter. Each line contains two character strings separated with the character '=' :
 - The first character string corresponds to the parameter name in the SQL query.
 - The Second one corresponds to the value.

Remark

None of the two character strings can be empty but the name can be constituted with only space characters.

> Output :

- `converted<Res:parameter.named.sql>` : The name of the converted resource (`parameter.named.sql` type resource).

Example :

LOAD path/to/parameter-named_value.properties AS value.file CONVERT value.file TO parameter.named.sql (from.text) AS value.properties
--

... to query.sql**Category-Name** : *query***What ?**

This *query* converter will convert a `file` type resource to a `query.sql` type resource.

CONVERT {resourceToConvert<Res:file>} TO query.sql (query) AS {converted<Res:query.sql>}
--

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a file which respects the following rules :
 - The file must contain only one query.
 - The query can be written on one or several lines.
 - The query end with the character ‘;’.
 - Comments at SQL format can be inserted in the file.

> Output :

- `converted<Res:query.sql>` : The name of the converted resource (`query.sql` type resource).

Example :

```
LOAD sql/my_query.sql AS my.file
CONVERT my.file TO query.sql (query) AS my.query
```

... to script.sql

Category-Name : *script*

What ?

This *script* converter will convert a `file` type resource to a `script.sql` type resource. It is possible to add an option for the encoding as well as the SQL block delimiter.

```
CONVERT {resourceToConvert<Res:file>} TO script.sql (script) AS {converted<Res:script.sql>} [USING {en-
coding}, {delimiter}]
```

> Input :

- `resourceToConvert<Res:file>` : The name (in the context) of the resource which references a file whose content is an SQL script.
- `Optional - encoding` : Parameter representing the query file encoding. Default value : “UTF-8”.
- `Optional - delimiter` : Parameter representing the SQL block delimiter. Default value : “@@”. It can be used in conjunction with encoding or by itself - in which case encoding will take its value by default.

> Output :

- `converted<Res:script.sql>` : The name of the converted resource (`script.sql` type resource).

Example :

```
LOAD path/to/my_script.sql AS my_script.file
CONVERT my_script.file TO script.sql (script) AS script [USING str_encoding, str_delimiter]
```

Remarks

- In your SQL script the delimiter should enclose any block of code that should be stored and passed in it's entirety to the database server.

Example :

- Creating a MySQL procedure :

```
DROP PROCEDURE IF EXISTS `some_procedure`;
@@
CREATE PROCEDURE `some_procedure` (...)
BEGIN
    ...
END;
@@
```

- Creating a PL/pgSQL function :

```
@@
CREATE OR REPLACE FUNCTION somefunc() RETURNS ... AS $$
DECLARE
    ...
BEGIN
    ...
    DECLARE
        ...
        BEGIN
            ...
            END;

    RETURN ...;

END;
$$ LANGUAGE plpgsql;@@
```

- Calling a stored PL/SQL procedure with delimiter set to "<DELIMITER>" :

```
<DELIMITER>
BEGIN
    some_procedure;
END;
<DELIMITER>
```

- In case of nested SQL blocks you only need to englobe the top level block with the delimiter.
 - **Comments** : refrain from using comments at the end of a line of code because it might induce a malfunction if it contains certain characters.
-

From directory to dataset.dbunit

Category-Name : *dataset*

What ?

This *dataset* converter will convert a *directory* type resource to a *dataset.dbunit* type resource.

CONVERT	{resourceToConvert<Res:directory>}	TO	dataset.dbunit	(dataset)	AS	{converted<Res:dataset.dbunit>}
---------	------------------------------------	----	----------------	-----------	----	---------------------------------

> Input :

`resourceToConvert<Res:directory>` : The name (in the context) of the resource which references a directory (*directory* type resource). This directory must contain at the root a file named `table-ordering.txt` which contains an ordered list of tables to add to the dataset. Each line of the file is a relative path to the root directory towards the csv file containing the table.

> Output :

- `converted<Res:dataset.dbunit>` : The name of the converted resource (*dataset.dbunit* type resource).

Example :

LOAD csv/csv1 AS csv1.file
CONVERT csv1.file TO directory (filesystem) AS csv1.dir
CONVERT csv1.dir TO dataset.dbunit (dataset) AS csv1.dataset

From properties to conf.dbunit.ppk

Category-Name : *from.properties*

What ?

This *from.properties* converter will convert a *properties* type resource to a *conf.dbunit.ppk* type resource.

CONVERT	{resourceToConvert<Res:properties>}	TO	conf.dbunit.ppk	(from.properties)	AS	{converted<Res:conf.dbunit.ppk>}
---------	-------------------------------------	----	-----------------	-------------------	----	----------------------------------

> Input :

- `resourceToConvert<Res:properties>` : The name (in the context) of the resource which references a `.properties` file (`properties` type resource). For each property, the key is a Table name, the value is a column or columns list separated with comma.

> Output :

- `converted<Res:conf.dbunit.ppk>` : The name of the converted resource (`conf.dbunit.ppk` type resource).

Example :

```
LOAD path/to/valid-ppk.properties AS ppk.file
CONVERT ppk.file TO properties (structured) AS ppk.properties
CONVERT ppk.properties TO conf.dbunit.ppk (from.properties) AS ppk
```

From result.sql to dataset.dbunit**Category-Name :** *dataset***What ?**

This *dataset* converter will convert a `result.sql` type resource to a `dataset.dbunit` type resource.

```
CONVERT {resourceToConvert<Res:result.sql>} TO dataset.dbunit (dataset) AS {converted<Res:dataset.dbunit>} USING {config<Res:file>}
```

> Input :

- `resourceToConvert<Res:result.sql>` : The name (in the context) of the resource which references a `result.sql` resource. It corresponds to the result of a SQL query `SELECT`.
- `config<Res:file>` : The name of the complementary resource which references a configuration file which contains only one key / value : `tablename` separated of the value with the character `'.'`. It's mandatory and can be define with an inline instruction. A `result.sql` resource does not have Table name and to transform a `result.sql` in `dataset.dbunit` we need a Table name.

> Output :

- `converted<Res:dataset.dbunit>` : The name of the converted resource (Resource of type `dataset.dbunit`).

Example :

```
CONVERT insertion_query.resultset TO dataset.dbunit (dataset) USING $(tablename : <name_Table>) AS dataset
```

From xml ...

... to dataset.dbunit

Category-Name : *dataset*

What ?

This *dataset* converter will convert an `xml` type resource to a `dataset.dbunit` type resource.

`CONVERT {resourceToConvert<Res:xml>} TO dataset.dbunit (dataset) AS {converted<Res:dataset.dbunit>}`

> Input :

- `resourceToConvert<Res:xml>` : The name (in the context) of the resource which references an `xml` file. The content of the file must be at the format of [FlatXMLDataset of DbUnit](#). Each line of a table is represented by an XML element :
 - The tag name corresponds of the name table.
 - Each column of the table is represented by an attribut :
 - * The attribute name corresponds to the column name.
 - * The attribute value corresponds to the column value in the represented line.

> Output :

- `converted<Res:dataset.dbunit>` : The name of the converted resource (`dataset.dbunit` type resource).

Example of XML file :

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <table1 colonne0="row 0 col 0" colonne1="row 0 col 1" />
  <table1 colonne0="row 1 col 0" colonne1="row 1 col 0" />
  <table1 colonne0="row 2 col 0" colonne1="row 2 col 0" />
  <table2 colonne0="row 0 col 0" />
  <table3 />
</dataset>
```

Remarks

- If in the initial resource the attribute value is the string character `[NULL]`, the corresponding column will have the value `null` in the converted resource (`dataset.dbunit` type resource).
- During the conversion, table columns are determined from the attributes of the first element corresponding to this table in the initial resource (`xml` type resource). For instance, if an `xml` resource contains 'T' elements :

- **Case 1** : Attribute of the first ‘T’ element not present but present after :

If :

- The first element ‘T’ doesn’t contain ‘C’ attribute and
- A ‘T’ element following contains a ‘C’ attribute

Then :

- The ‘C’ attribute will be ignored during the conversion. That means no ‘C’ column in the converted resource.

- **Case 2** :

If :

- The first ‘T’ element contains a ‘C’ attribute and
- A ‘T’ element following doesn’t contain a ‘C’ attribute

Then :

- There will be a ‘C’ column in the converted resource. In lines corresponding to the elements which doesn’t contain the ‘C’ attribute, the ‘C’ column will have the value ‘null’.

Example :

```
LOAD path/to/dataset.xml AS dataset.file
CONVERT dataset.file TO xml (structured) AS my_xml_file
CONVERT my_xml_file TO dataset.dbunit (dataset) AS dataset.dbu
```

... to filter.dbunit

Category-Name : *filter*

What ?

This *filter* converter will convert an `xml` type resource to a `filter.dbunit` type resource.

```
CONVERT {resourceToConvert<Res:xml>} TO filter.dbunit (filter) AS {converted<Res:filter.dbunit>}
```

> Input :

- `resourceToConvert<Res:xml>` : The name (in the context) of the resource which references a `xml` file. This `xml` file looks like :
 - For a Table exclusion :

```
<?xml version="1.0" encoding="UTF-8"?>
<filter>
  <tableExclude tableRegex="table_name"/>
</filter>
```

- For a column exclusion :

```
<?xml version="1.0" encoding="UTF-8"?>
<filter>
<tableInclude tableRegex="table_name">
  <columnExclude>column_name</columnExclude>
</tableInclude>
</filter>
```

> Output :

- converted<Res:dataset.dbunit> : The name of the converted resource (dataset.dbunit type resource).

Example :

```
LOAD path/to/column_exclude.xml AS filter_dbunit.file
CONVERT filter_dbunit.file TO filter.dbunit (filter) AS filter_dbunit
```

Database Plugin - Commands

Contents :

- *'execute' 'query.sql' on 'database'*
- *'execute' 'query.sql' on 'database' via 'parameter.indexed.sql'*
- *'execute' 'query.sql' on 'database' via 'parameter.named.sql'*
- *'execute' 'script.sql' on 'database'*
- *'get.all' on 'database'*
- *'insert' 'dataset.dbunit' on 'database'*
- *'delete' 'dataset.dbunit' on 'database'*

'execute' 'query.sql' on 'database'

What ?

This command executes a SQL query represented by a `query.sql` resource on the specified database target.

```
EXECUTE execute WITH {query<Res:query.sql>} ON {<Tar:database>} AS {result<Res:result.sql>}
```

> Input :

- `query<Res:query.sql>` : The name (in the context) of the resource which references a SQL query (`query.sql` type resource).
- `<Tar:database>` : The name (in the context) of the database to use (`database` type target).

> Output :

- `result<Res:result.sql>` : The name of the resource which contains the result of the SQL query (`result.sql` type resource).

Example :

```
LOAD path/to/my_query.sql AS query.file
CONVERT query.file TO query.sql (query) AS query1

EXECUTE execute WITH query1 ON mydatabase-db AS my_query_result
```

'execute' 'query.sql' on 'database' via 'parameter.indexed.sql'

What ?

This command executes a SQL query represented by a `query.sql` resource on the specified database target via indexed parameters.

```
EXECUTE execute WITH {query<Res:query.sql>} ON {<Tar:database>} AS {result<Res:result.sql>} USING
{config<Res:parameter.indexed.sql>}
```

> Input :

- `query<Res:query.sql>` : The name (in the context) of the resource which references a SQL query (`query.sql` type resource).
- `<Tar:database>` : The name (in the context) of the database to use (`database` type target).
- `config<Res:parameter.indexed.sql>` : The name of the resource which contains indexed parameters for the SQL query (`parameter.indexed.sql` type resource).

> Output :

- `result<Res:result.sql>` : The name of the resource which contains the result of the SQL query (`result.sql` type resource).

Example :

```
LOAD path/to/my_query.sql AS query.file
CONVERT query.file TO query.sql (query) AS query1

LOAD path/to/parameter-indexed_value.properties AS value.file
CONVERT value.file TO parameter.indexed.sql (from.text) AS value.properties

EXECUTE execute WITH query1 ON mydatabase-db AS my_query_result USING value.properties
```

'execute' 'query.sql' on 'database' via 'parameter.named.sql'

What ?

This command executes a SQL query represented by a `query.sql` resource on the specified database target via named parameters.

```
EXECUTE execute WITH {query<Res:query.sql>} ON {<Tar:database>} AS {result<Res:result.sql>} USING
{config<Res:parameter.named.sql>}
```

> Input :

- `query<Res:query.sql>` : The name (in the context) of the resource which references a SQL query (`query.sql` type resource).
- `<Tar:database>` : The name (in the context) of the database to use (`database` type target).
- `config<Res:parameter.named.sql>` : The name of the resource which contains named parameters for the SQL query (`parameter.named.sql` type resource).

> Output :

- `result<Res:result.sql>` : The name of the resource which contains the result of the SQL query (`result.sql` type resource).

Example :

```
LOAD path/to/my_query.sql AS query.file
CONVERT query.file TO query.sql (query) AS query1
LOAD path/to/parameter-named_value.properties AS value.file
CONVERT value.file TO parameter.named.sql (from.text) AS value.properties
EXECUTE execute WITH query1 ON mydatabase-db AS my_query_result USING value.properties
```

‘execute’ ‘script.sql’ on ‘database’

What ?

This command executes a SQL script represented by a `script.sql` resource on the specified database target.

```
EXECUTE execute WITH {script<Res:script.sql>} ON {<Tar:database>} AS $() [ USING $(keep.separator:
<keepSeparator>)]
```

> Input :

- `script<Res:script.sql>` : The name (in the context) of the resource which references a SQL script (`script.sql` type resource).
- `<Tar:database>` : The name (in the context) of the database on which the SQL script should be used (database type target).
- `<keepSeparator>` : Indicate to the command to keep or remove the separator (“;”) at the end of each SQL query of the script. This parameter can take one of two values : “true” or “false”. By default this parameter is set to “true”.

Example 1 :

```
LOAD path/to/my_script.sql AS script.file
CONVERT script.file TO script.sql (script) AS script1

EXECUTE execute WITH script1 ON mydatabase-db AS $()
```

Example 2 :

```
EXECUTE execute WITH script1 ON mydatabase-db AS $() USING $(keep.separator:false)
```

‘get.all’ on ‘database’

What ?

This command allows to create a DbUnit dataset from a specific database.

```
EXECUTE get.all WITH $() ON {<Tar:database>} AS {result<Res:dataset.dbunit>} [USING
[{{<Res:conf.dbunit>}},[{{<Res:conf.dbunit.ppk>}}] ]
```

> Input :

- <Tar:database> : The name (in the context) of the database to use (database type target).
- <Res:conf.dbunit> : This resource contains DbUnit configuration properties.
- <Res:conf.dbunit.ppk> : The name of the resource which references a configuration file to define the pseudo primary keys.

Remarks

1. If for a table a primary key and a pseudo primary key are defined, the pseudo primary key override the primary key.
 2. If for a table a pseudo primary key is defined with one or more non existent columns, the command fails.
-

> Output :

- result<Res:dataset.dbunit> : The name of the resource which contains the DbUnit dataset of all the database.

Example 1 :

```
EXECUTE get.all WITH $() ON myDatabase-db AS myDataset
```

Example 2 :

```
LOAD path/to/dbunit-conf.properties AS conf.file
CONVERT conf.file TO conf.dbunit (structured) AS conf.dbu
```

```
EXECUTE get.all WITH $() ON my_Database-db USING conf.dbu AS my_dataset
```


'insert' 'dataset.dbunit' on 'database'**What ?**

This command insert a DbUnit dataset on the specified database target.

```
EXECUTE insert WITH {dataset<Res:dataset.dbunit>} ON {<Tar:database>} AS $() [USING [$(operation : <type>)],[{<Res:conf.dbunit>}],[{<Res:conf.dbunit.ppk>}]]
```

> Input :

- `dataset<Res:dataset.dbunit>` : The name of the resource which references a DbUnit dataset (Resource of type `dataset.dbunit`).
- `<Tar:database>` : The name (in the context) of the database to use (database type target).
- `<type>` : 4 values are possible for this parameter :
 - `INSERT` : for a simple operation of insert. This operation assumes that table data does not exist in the target database and fails if this is not the case.
 - `CLEAN_INSERT` : a 'delete all' is realised before the 'insert' operation.
 - `UPDATE` : this operation assumes that table data already exists in the target database and fails if this is not the case.
 - `REFRESH` : data of existing rows are updated and non-existing row get inserted. Any rows which exist in the database but not in dataset stay unaffected.

Remark

If “\$(operation : <type>)” is not defined, property is by default `CLEAN_INSERT`.

- `<Res:conf.dbunit>` : This resource contains DbUnit configuration properties.
- `<Res:conf.dbunit.ppk>` : The name of the resource which references a configuration file to define the pseudo primary keys.

Remark

1. If for a table a primary key and a pseudo primary key are defined, the pseudo primary key override the primary key.
2. If for a table a pseudo primary key is defined with one or more non existents columns, the command fails.

Example :

```
LOAD path/to/dataset_to_insert.xml AS dataset_file
CONVERT dataset_file TO xml (structured) AS dataset_xml
CONVERT dataset_xml TO dataset.dbunit (dataset) AS dataset_dbu
```

```
LOAD path/to/dbunit-conf.properties AS conf_file
CONVERT conf_file TO conf.dbunit (structured) AS conf_dbu
```

```
EXECUTE insert WITH dataset_dbu ON my_database-db USING conf_dbu,$( operation : INSERT ) AS $()
```

‘delete’ ‘dataset.dbunit’ on ‘database’

What ?

This command delete a DbUnit Dataset on the specified database target.

```
EXECUTE delete WITH {dataset<Res:dataset.dbunit>} ON {<Tar:database>} AS $() [USING [$(operation : <type>)],[{<Res:conf.dbunit>}],[{<Res:conf.dbunit.ppk>}]]
```

> Input :

- dataset<Res:dataset.dbunit> : The name of the resource which references a DbUnit dataset (dataset.dbunit type resource).
- <Tar:database> : The name (in the context) of the database to use (database type target).
- <type> : 2 values are possible for this parameter :
 - DELETE : This operation deletes only the dataset contents from the database. This operation does not delete the entire table contents but only data that are present in the dataset.
 - DELETE_ALL : Deletes all rows of tables present in the specified dataset. If the dataset does not contains a particular table, but that table exists in the database, the database table is not affected. Table are truncated in reverse sequence.

Remark

If “\$(operation : <type>)” is not defined, property is by default DELETE_ALL.

- <Res:conf.dbunit> : This resource contains DbUnit configuration properties.
- <Res:conf.dbunit.ppk> : The name of the resource which references a configuration file to define the pseudo primary keys.

Remarks

1. If for a table a primary key and a pseudo primary key are defined, the pseudo primary key override the primary key.
2. If for a table a pseudo primary key is defined with one or more non existents columns, the command fails.

Example :

```
LOAD path/to/dataset_to_insert.xml AS dataset_file
CONVERT dataset_file TO xml (structured) AS dataset_xml
CONVERT dataset_xml TO dataset.dbunit (dataset) AS dataset_dbu

LOAD path/to/dbunit-conf.properties AS conf_file
CONVERT conf_file TO conf.dbunit (structured) AS conf_dbu

EXECUTE delete WITH dataset_dbu ON my_database-db USING conf_dbu,$( operation : DELETE ) AS $()
```

Database Plugin - Asserts**Contents :**

- *'dataset.dbunit' does 'contain' 'dataset.dbunit'*
- *'dataset.dbunit' is 'equal' 'dataset.dbunit'*

'dataset.dbunit' does 'contain' 'dataset.dbunit'**What ?**

Asserts that the first dataset contains the second one.

```
ASSERT {dataset1<Res:dataset.dbunit>} DOES contain THE {dataset2<Res:dataset.dbunit>} [USING
[<Res:filter.dbunit>]], [<Res:conf.dbunit.ppk>]]
VERIFY {dataset1<Res:dataset.dbunit>} DOES contain THE {dataset2<Res:dataset.dbunit>} [USING
[<Res:filter.dbunit>]], [<Res:conf.dbunit.ppk>]]
```

> Input :

- `dataset1<Res:dataset.dbunit>` : The name of the resource which references the first DbUnit dataset (`dataset.dbunit` type resource).
- `dataset2<Res:dataset.dbunit>` : The name of the resource which references the second DbUnit dataset (`dataset.dbunit` type resource).
- `<Res:filter.dbunit>` (Optional) : This resource contains a filter DbUnit (`filter.dbunit` type resource).
- `<Res:conf.dbunit.ppk>` (Optional) : The name of the resource which references a configuration file to define the pseudo primary keys (`conf.dbunit.ppk` type resource).

Example :

```
LOAD path/to/dataset1.xml AS dataset1_file
CONVERT dataset1_file TO xml (structured) AS dataset1_xml
CONVERT dataset1_xml TO dataset.dbunit (dataset) AS dataset1_dbu

LOAD path/to/dataset1.xml AS dataset2_file
CONVERT dataset2_file TO xml (structured) AS dataset2_xml
CONVERT dataset2_xml TO dataset.dbunit (dataset) AS dataset2_dbu

// Get the pseudo primary keys
LOAD path/to/my_ppk.properties AS ppk_file
CONVERT ppk_file TO properties (structured) AS ppk_properties
CONVERT ppk_properties TO conf.dbunit.ppk (from.properties) AS ppk_dbu

// Load the DbUnit filter
LOAD path/to/filter-name.xml AS filter_file
CONVERT filter_file TO filter.dbunit AS filter_dbu

// Compare the two datasets
ASSERT dataset1_dbu DOES contain THE dataset2_dbu USING ppk_dbu,filter_dbu
```

Remarks

- During the assertion, if the first or the second dataset contains primary keys they are used for the assertion.
 - If for a given table, a primary key and a pseudo primary key are defined, pseudo primary key overrides the primary key.
 - If for a given table, a pseudo primary key has one or several columns excluded from the assertion by a DbUnit filter, the command fails.
-

'dataset.dbunit' is 'equal' 'dataset.dbunit'

What ?

Asserts that the first dataset is equal to the second one (same number of tables, for each table same number of lines / columns and same data).

ASSERT	{dataset1<Res:dataset.dbunit>}	IS	equal	THE	{dataset2<Res:dataset.dbunit>}	[USING
	[[<Res:filter.dbunit>]],[[<Res:conf.dbunit.ppk>]]]					
VERIFY	{dataset1<Res:dataset.dbunit>}	IS	equal	THE	{dataset2<Res:dataset.dbunit>}	[USING
	[[<Res:filter.dbunit>]],[[<Res:conf.dbunit.ppk>]]]					

> Input :

- dataset1<Res:dataset.dbunit> : The name of the resource which references the first DbUnit dataset.(dataset.dbunit type resource).
- dataset2<Res:dataset.dbunit> : The name of the resource which references the second DbUnit dataset.(dataset.dbunit type resource).
- <Res:filter.dbunit> (Optional) : This resource contains a filter DbUnit (filter.dbunit type resource).
- <Res:conf.dbunit.ppk> (Optional) : The name of the resource which references a configuration file to define the pseudo primary keys.(conf.dbunit.ppk type resource).

Example :

```
LOAD path/to/dataset1.xml AS dataset1_file
CONVERT dataset1_file TO xml (structured) AS dataset1_xml
CONVERT dataset1_xml TO dataset.dbunit (dataset) AS dataset1_dbu

LOAD path/to/dataset1.xml AS dataset2_file
CONVERT dataset2_file TO xml (structured) AS dataset2_xml
CONVERT dataset2_xml TO dataset.dbunit (dataset) AS dataset2_dbu

// Get the pseudo primary keys
LOAD path/to/my_ppk.properties AS ppk_file
CONVERT ppk_file TO properties (structured) AS ppk_properties
CONVERT ppk_properties TO conf.dbunit.ppk (from.properties) AS ppk_dbu

// Load the DbUnit filter
LOAD path/to/filter-name.xml AS filter_file
CONVERT filter_file TO filter.dbunit AS filter_dbu

// Compare the two datasets
ASSERT dataset1_dbu IS equal THE dataset2_dbu USING ppk_dbu,filter_dbu
```

Remarks

- During the assertion, if the first or the second dataset contains primary keys they are used for the assertion.
 - If for a given table, a primary key and a pseudo primary key are defined, pseudo primary key overrides the primary key.
 - If for a given table, a pseudo primary key has one or several columns excluded from the assertion by a DbUnit filter, the command fails.
-

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin provides all the elements needed to interact with a database.

6.3 Filechecker Plugin

6.3.1 Filechecker Plugin - Resources

Contents :

- *fixed.field.file*
- *fixed.field.file.descriptor*
- *fixed.field.file.queries*

fixed.field.file

Category-name : *fixed.field.file*

What ?

fixed.field.file is a resource type that represents a fixed field file (aka FFF).

fixed.field.file.descriptor

Category-name : *fixed.field.file.descriptor*

What ?

fixed.field.file.descriptor is a resource type that represents the descriptor of fixed field file. This descriptor defines the structure of the fixed field file.

fixed.field.file.queries

Category-name : *fixed.field.file.queries*

What ?

fixed.field.file.queries is a resource type that represents a file which contains a list of queries. Each query is an assertion against a fixed field file.

6.3.2 Filechecker Plugin - Macros

Contents :

- `# LOAD_FFF {fixed_field_file_path} USING {fixed_field_file_descriptor_path} AS {fixed_field_file}`
- `# ASSERT_FFF {fff} HAS EXPECTED CONTENT USING {fff_queries_path}`

LOAD_FFF {fixed_field_file_path} USING {fixed_field_file_descriptor_path} AS {fixed_field_file}

What ?

This macro will load the fixed field file descriptor and the fixed field file. It will then verify that the fixed field file is valid by using the fixed field file descriptor. Finally it will check that the fixed field file has the expected autonumbers. The macro will also convert the fixed field file into a resource of type `fixed.field.file`.

Underlying instructions :

```
LOAD {fixed_field_file_descriptor_path} AS __temp{%%rand1}.file
CONVERT __temp{%%rand1}.file TO fixed.field.file.descriptor(descriptor) AS __temp_{%
↪%rand2}.fff.descriptor

LOAD {fixed_field_file_path} AS __temp{%%rand3}.file
CONVERT __temp{%%rand3}.file TO fixed.field.file(structured) USING __temp_{%%rand2}.
↪fff.descriptor AS {fixed_field_file}

ASSERT {fixed_field_file} IS valid
ASSERT {fixed_field_file} HAS expected.autonumbers
```

> Input :

- `{fixed_field_file_path}` : The path to the fixed field file (relative to the root of the repository).
- `{fixed_field_file_descriptor_path}` : The path to the fixed field file descriptor (relative to the root of the repository).

> Output :

- `{fixed_field_file}` : The name (in the context) of the fixed field file.

Example :

```
# LOAD_FFF repo/fff/data.txt USING repo/descriptor/my_descriptor.xml AS fixed_field_file.fff
```

Fixed Field File to process :

```
1 00001100v1.007/09/2012
2 00002101POPI ERIC 69 RUE DE WOODSTOCK 12584LARZACVILLE 110/05/2013 003999
3 00003102INTERNATIONAL PREMIUM 001000
4 00004105PERMANENTE
5 00005104HORS FORFAIT 001000
6 000061070601010101
7 000071070601010101
8 000081070602020202
9 000091070603030303
10 00010103005999
11 00011101HANS REY 8 BOULEVARD FLEURI 44000NANTES 210/05/2013 004999
12 00012102INTERNATIONAL PREMIUM 001000
13 000131051MOIS
14 00014106MINUTES REPORTEES 09
15 000151070601010101
16 000161070601010101
17 000171070602020202
18 000181070603030303
19 00019103005999
20 00020101BLABLA MONIQUE 10 RUE DE VAUGIRARD 75000PARIS 201/08/2014 004999
21 00021102INTERNET 4G0 001000
22 00022105PERMANENTE
23 00023106MINUTES REPORTEES 09
24 000241070601010101
25 000251070601010101
26 000261070602020202
27 000271070603030303
28 00028103005999
29 00029101TOTO CHRISTIANE 56 RUE BROSSOLETTE 92170VANVES 331/11/2013 004999
30 00030102INTERNATIONAL PREMIUM 001000
31 00031105PERMANENTE
32 00032104HORS FORFAIT 000500
33 000331070601010101
34 000341070601010101
35 000351070602020202
36 000361070603030303
37 00037103006499
38 0003810900003800004
```

Descriptor xml file to process (partial) :

The project's doctree showing the resources directory (containing the fff and descriptor xml file) to process :

SKF script :

Result output on success :

To get a clearer view of the fonctionnality offered by the LOAD_FFF macro we can create a deliberate error in the fixed field file.

We add an **empty line** between the end of the first **leaf record** and the start of the second :

Result output on failure :

```
# ASSERT_FFF {fff} HAS EXPECTED CONTENT USING {fff_queries_path}
```

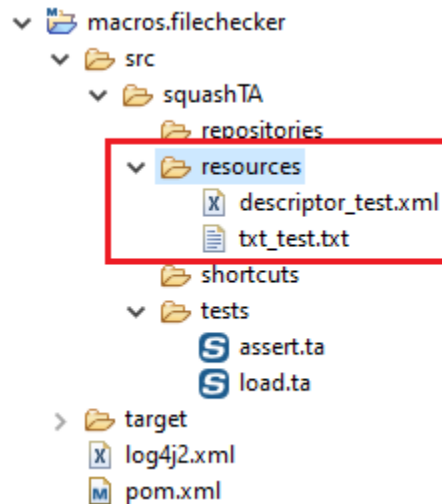
What ?

This macro allows to valid the content of a fixed field file using a fixed field file query file.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <fff:root name="test_file" multiRecords="true" flat="false"
3   xmlns:fff="http://www.squashtest.org/SquashTA/FFFDescriptorSchema"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://www.squashtest.org/SquashTA/FFFDescriptorSchema http://www.squashtest.org/docta/schemas/FFFDescriptorSchema_1.1.xsd">
6   <fff:sequences>
7     <fff:sequence id="Ligne" start="1" increment="1" />
8   </fff:sequences>
9   <fff:records>
10    <fff:leaves>
11      <fff:leafRecord name="101">
12        <fff:label>101</fff:label>
13        <fff:fields>
14          <fff:field type="autonumber" start="1" length="5" sequence="Ligne">
15            <fff:label>N° ligne</fff:label>
16            <fff:description>N° ligne</fff:description>
17          </fff:field>
18          <fff:field type="id" start="6" length="3" format="101">
19            <fff:label>Infos</fff:label>
20            <fff:description>Infos client</fff:description>
21          </fff:field>
22          <fff:field type="text" start="9" length="20" nullable="false" format="{\w|-| }{20}">
23            <fff:label>Nom client</fff:label>
24            <fff:description>Nom client</fff:description>
25          </fff:field>
26          <fff:field type="text" start="29" length="20" nullable="false" format="{\w|-| }{20}">
27            <fff:label>Prénom client</fff:label>
28            <fff:description>Prénom client</fff:description>
29          </fff:field>
30          <fff:field type="text" start="49" length="50" nullable="false" format="{\w|-| }{50}">
31            <fff:label>Adresse client</fff:label>
32            <fff:description>Adresse client</fff:description>
33          </fff:field>
34          <fff:field type="text" start="99" length="5" nullable="false" format="{\w|-| }{5}">
35            <fff:label>CP client</fff:label>
36            <fff:description>Code Postal client</fff:description>
37          </fff:field>
38          <fff:field type="text" start="104" length="20" nullable="false" format="{\w|-| }{20}">
39            <fff:label>Ville client</fff:label>
40            <fff:description>Ville client</fff:description>
41          </fff:field>
42          <fff:field type="text" start="124" length="1" nullable="false" format="{\w|-| }{1}">
43            <fff:label>Code forfait</fff:label>
44            <fff:description>Code forfait</fff:description>
45          </fff:field>
46          <fff:field type="date" start="125" length="10" nullable="false" format="dd/mm/yyyy">
47            <fff:label>Date fin</fff:label>
48            <fff:description>Date fin forfait</fff:description>

```



```

1 test :
2
3 # LOAD_FFF txt_test.txt USING descriptor_test.xml AS test
4

```

```
[INFO] Squash TA : testing...
[INFO] 2019-09-24 11:44:40.343 [main] EcosystemRunnerImpl - Beginning execution of ecosystem tests
[INFO] 2019-09-24 11:44:40.350 [main] TestRunnerImpl - Beginning execution of test assert.ta
[INFO] 2019-09-24 11:44:40.453 [main] DescriptorChecker - Validation structurelle et syntaxique du fichier de description "file://C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108905/tests/tests/assert.ta/descriptor_test213460142231.xml"
[INFO] 2019-09-24 11:44:40.484 [main] DescriptorChecker - Validation fonctionnelle du fichier de description "file://C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108905/tests/tests/assert.ta/descriptor_test213460142233211877.xml"
[INFO] 2019-09-24 11:44:40.937 [main] FFDescriptorParser - Traitement du fichier de description file://C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108905/tests/tests/assert.ta/descriptor_test213460142233211877.xml
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TA : build complete.
[INFO] 2019-09-24 11:44:41.200 [main] TempFileUtils - You can access to the temporary files created in C:\Users\lpoma\AppData\Local\Temp\Squash_TA\20190924_114438_6399061409104313108905
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.326 s
[INFO] Finished at: 2019-09-24T11:44:41+02:00
[INFO] Final Memory: 219/205M
[INFO] -----
```

1	00001108v1.007/09/2012					
2	00002101POPI	ERIC	69 RUE DE WOODSTOCK	12584LARZACVILLE	110/05/2013	003999
3	00003102INTERNATIONAL PREMIUM		001000			
4	00004105PERMANENTE					
5	00005104HORS FORFAIT			001000		
6	000061070601010101					
7	000071070601010101					
8	000081070602020202					
9	000091070603030303					
10	00010103005999					
11						
12	000110101NANTES	REY	0 BOULEVARD FLEURI	44000NANTES	210/05/2013	004999
13	00012102INTERNATIONAL PREMIUM		001000			
14	000131051MOIS					
15	00014106MINUTES REPORTEES		09			
16	000151070601010101					
17	000161070601010101					
18	000171070602020202					
19	000181070603030303					
20	00019103005999					
21	00020101BLABLA	MONIQUE	10 RUE DE VAUGIRARD	75000PARIS	201/08/2014	004999
22	00021102INTERNET 4G0		001000			
23	00022105PERMANENTE					
24	00023106MINUTES REPORTEES		09			
25	000241070601010101					
26	000251070601010101					
27	000261070602020202					
28	000271070603030303					
29	00028103005999					
30	00029101TOTO	CHRISTIANE	56 RUE BROSSOLETTE	92170VANVES	331/11/2013	004999
31	00030102INTERNATIONAL PREMIUM		001000			
32	00031105PERMANENTE					
33	00032104HORS FORFAIT			000500		
34	000331070601010101					
35	000341070601010101					
36	000351070602020202					
37	000361070603030303					
38	00037103006499					
39	0003810900003000004					

```
[ERROR] 2019-09-24 12:02:59.402 [main] TestRunnerImpl - The execution raised an error in the TEST phase of the TA script 'load.ta' with the message: 'Test Automation Engine error (non SUT)':
Detaills: La ligne n°11 ne correspond à aucun des enregistrements possibles.(longueur et/ou champ(s) de type "id" incorrects)'.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TA : build complete.
[INFO] 2019-09-24 12:02:59.606 [main] TempFileUtils - You can access to the temporary files created in C:\Users\lpoma\AppData\Local\Temp\Squash_TA\20190924_120257_1752935063697395272850
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 3.709 s
[INFO] Finished at: 2019-09-24T12:02:59+02:00
[INFO] Final Memory: 22M/300M
[INFO] -----
[ERROR] Failed to execute goal org.squashtest.ta:squash-ta-maven-plugin:1.12.0-RELEASE:run (default-cli) on project macros.filechecker: Build failure : there are tests failures
[ERROR] Test statistics : 1 test runs, 0 passed, 1 tests didn't pass
[ERROR] Tests failed / crashed :
[ERROR] -----
[ERROR] tests:
[ERROR] ==> load.ta
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/HAVEN/MojoFailureException
```

Underlying instructions :

```
LOAD {fff_queries_path} AS __temp{%%rand1}.file
CONVERT __temp{%%rand1}.file TO fixed.field.file.queries(query) AS __temp_{%%rand2}.
↪fff.queries
ASSERT {fff} HAS expected.content USING __temp_{%%rand2}.fff.queries
```

> Input :

- {fff} : The path to the fixed field file (relative to the root of the repository).
- {fff_queries_path} : The path to the query file (relative to the root of the repository).

Example :

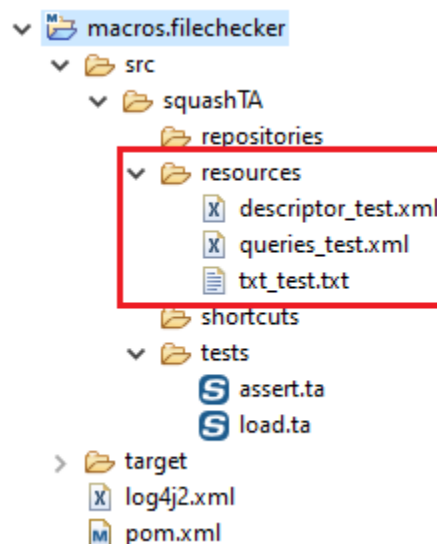
```
# ASSERT_FFF repo/fff/data.txt HAS EXPECTED CONTENT USING repo/queries/my_queries.xml
```

This example is based on the previous one. For more details, please check [here](#).

Queries file to process :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <sv:root xmlns:sv="http://www.squashtest.org/SquashTA/FFFQueriesSchema"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.squashtest.org/SquashTA/FFFQueriesSchema http://www.squashtest.org/docta/schemas/FFFQueriesSchema_1.1.xsd">
5   <sv:assertion>
6     <sv:queryName>Vérif des options du forfait de Jean POPI</sv:queryName>
7     <sv:query>
8       count (.[name="Fichier"]/
9         children[name="Client"]/
10        children[name="Options"]/
11        openingRecord[name="102"]/
12        fields[label="Numero ligne" and value="00003"]/../../
13        fields[label="Option" and value="102"]/../../
14        fields[label="Nom" and value="INTERNATIONAL PREMIUM" and value="001000"]/../../
15        fields[label="Montant Option" and value="001000"]/../../
16      )
17    </sv:query>
18    <sv:result>1</sv:result>
19  </sv:assertion>
20 </sv:root>
```

The project's doctree showing the resources directory (containing the fff, descriptor xml file and now queries file) to process :



```

1 test :
2
3 # LOAD_FFF txt_test.txt USING descriptor_test.xml AS fixed_field_file.fff
4
5 # ASSERT_FFF fixed_field_file.fff HAS EXPECTED CONTENT USING queries_test.xml
6 |

```

SKF script :

Result output on success :

```

[INFO] Squash TA : testing...
[INFO] 2019-09-24 11:44:40.343 [main] EcosystemRunnerImpl - Beginning execution of ecosystem tests
[INFO] 2019-09-24 11:44:40.359 [main] TestRunnerImpl - Beginning execution of test assert.ta
[INFO] 2019-09-24 11:44:40.453 [main] DescriptorChecker - Validation structurelle et syntaxique du fichier de description "file:/C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108095/tests/ta/assert.ta/descriptor_test213460142233211877.xml"
[INFO] 2019-09-24 11:44:40.484 [main] DescriptorChecker - Validation fonctionnelle du fichier de description "file:/C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108095/tests/ta/assert.ta/descriptor_test213460142233211877.xml"
[INFO] 2019-09-24 11:44:40.937 [main] FFFDescriptorParser - Traitement du fichier de description file:/C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108095/tests/ta/assert.ta/descriptor_test213460142233211877.xml
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TA : build complete.
[INFO] 2019-09-24 11:44:41.200 [main] TempFileUtils - You can access to the temporary files created in C:/Users/lpoma/AppData/Local/Temp/Squash_TA/20190924_114438_6399061409104313108095
[INFO] BUILD SUCCESS
[INFO] Total time: 0.326 s
[INFO] Finished at: 2019-09-24T11:44:41+02:00
[INFO] Final Memory: 235/205M
[INFO]

```

To get a clearer view of the fonctionnality offered by the ASSERT_FFF macro we can create a deliberate error in the queries file :

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <sv:root xmlns:sv="http://www.squashtest.org/SquashTA/FFFQueriesSchema"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.squashtest.org/SquashTA/FFFQueriesSchema http://www.squashtest.org/docta/schemas/FFFQueriesSchema_1.1.xsd">
5   <sv:assertion>
6     <sv:queryName>Vérif des options du forfait de Jean POPI</sv:queryName>
7     <sv:query>
8       count (.[name="Fichier"]/
9         children[name="Client"]/
10        children[name="Options"]/
11        openingRecord[name="102"]/
12        fields[label="Numero ligne" and value="12345"]/../../
13        fields[label="Option" and value="102"]/../../
14        fields[label="Nom" and value="INTERNATIONAL PREMIUM" and value="001000"]/../../
15        fields[label="Montant Option" and value="001000"]/..
16      )
17    </sv:query>
18    <sv:result>1</sv:result>
19  </sv:assertion>
20 </sv:root>

```

Result output on failure :

6.3.3 Filechecker Plugin - Specifications for the Fixed Field Files

Contents :

- *Functionalities*
- *Terminology about the FFF*
 - *Leafrecords*
 - *Composite records*
- *FFF Descriptor*
 - *Structure of the FFF descriptor*

```
[ERROR] 2019-09-24 12:34:31.963 [main] TestRunnerImpl - The execution failed in the TEST phase of the TA script 'assert.ta' with the message: 'Query 'Vérif des options du forfait de Jean POPI' returned 0 record(s) instead of 1.'
```

```
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TA : build complete.
[INFO ] 2019-09-24 12:34:32.166 [main] TempFileUtils - You can access to the temporary files created in C:\Users\lpoma\AppData\Local\Temp\Squash_TA\20190924_123429_7735753760327739377389
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 3.661 s
[INFO] Finished at: 2019-09-24T12:34:32+02:00
[INFO] Final Memory: 23M/383M
```

- * *<root> element*
- * *<sequences> and <sequence> elements*
- * *LeafRecord*
- * *Composite Record*
- * *Succession of the composite Record's children*
 - *'AND' Clause*
 - *'or' Clause*
 - *'repeat' Clause*
 - *Combination of the 'and', 'or' and 'repeat' clauses*
- *Validation of the FFF descriptor*

Functionalities

The Filechecker plugin allows to :

- Read a FFF (Fixed Field File) in the two kinds of format : binary or text.
- Identify leafrecords / composite records of a FFF.
- Validate fields syntax and validate the structure of a FFF.
- Verify the value of the fields.

To accomplish the first 3 points, an XML file named 'FFF descriptor' is needed. In addition, an Xpath queries file is used to verify the value of the fields.

Terminology about the FFF

Leafrecords

A FFF file is built of leafrecords and each leafrecord is built of fields. For instance :

Textual FFF (a FFF of type text) have one record per line (records are separated with a word wrap character) so the records access is sequential. Within a record, the position and the number of characters for each field is known. In our example, the civility field is built of 3 characters whereas the name and the first name are built of 10 characters.

Remarks :

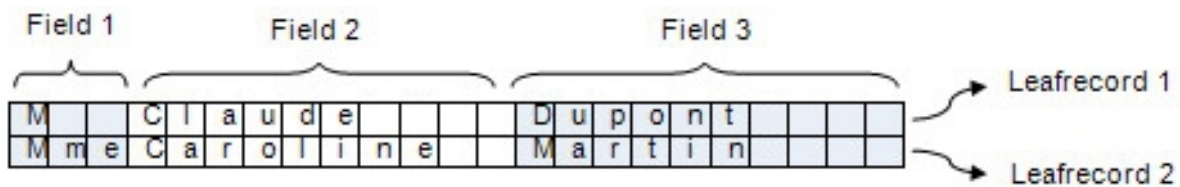


Fig. 1: This file is built of 2 leafrecord and each leafrecord constituted of 3 fields.

- It's the same for a binary file. All records have the same bytes number which allows distinguishing them from each other. And inside a record, the position and the number of bytes for each field is known.
- In a FFF file, if all records are built of the same type (as our example, we speak about mono-recording file). If the file is built of several kinds of records, we speak about multi-recording file and each record has one or more identifier fields. For instance :

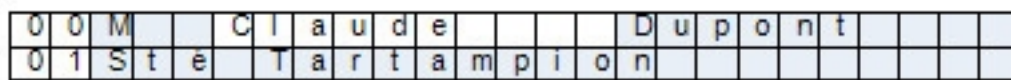
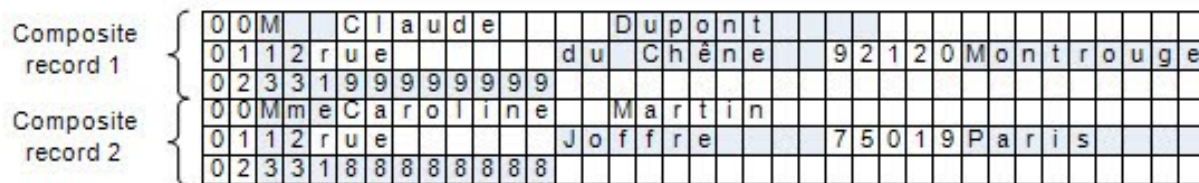


Fig. 2: In this example of a multi-recording file, the field ‘00’ allows identifying a physical person and the field ‘01’ allows identifying a moral person.

- If a file is built of only leafrecords we speak about ‘Flat File’.

Composite records

When a suite of leafrecords form an unity, we speak about composite records. For instance :



In this example, each composite record is built of 3 leafrecords :

- The leafrecord '00' for the civility.
- The leafrecord '01' for the adress.
- The leafrecord '02' for the phone number.

Among the leafrecords composing a composite record, we can distinguish 3 kinds of leafrecords :

- The opening record which indicates the first leafrecord of a composite record.
- The closing record which indicates the last leafrecord of a composite record. It allows to detect the end of a composite record but they're not mandatory (in the previous example, there is no closing records).

- Others leafrecords are named children leafrecords.

Remark : Generally the suite of leafrecords of a composite record are subject to management rules. For instance, a person must have a civil status AND a phone number.

FFF Descriptor

Structure of the FFF descriptor

A FFF descriptor is a XML file which has the following structure :

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <sequences>...</sequences>
  <records>
    </leaves>...</leaves>
    <composites>...</composites>
  </records>
</root>
```

The <ROOT> tag is the root element of the FFF descriptor. It allows to :

- State the schema to use to validate the FFF descriptor.
- Describe the general characteristics of the FFF to verify.

It contains 2 tags :

- The <sequences> tag (Optional) : It contains sequences definition used for the auto-incremental fields.
- The <records> tag : It contains the records description of the file and is built with :
 - A <leaves> tag which contains n <leafRecord> tags (they describe the leafrecords type of the FFF to verify).
 - A <composites> tag which contains n <compositeRecord> tags (they describe the composite records type of the FFF to verify).

<root> element

The <root> tag must have the following attributes :

```
<fff:root name="LISTING" binary="true" encoding="Cp1047" bytesPerLine="62"
xmlns = "http://www.squashtest.org/SquashTA/FFFDescriptorSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.squashtest.org/SquashTA/FFFDescriptorSchema http://www.squashtest.org/
↪docta/schemas/FFFDescriptorSchema_1.1.xsd">
```

We're going to explain the different attributes of the <root> tag :

```
xmlns= "http://www.squashtest.org/SquashTA/FFFDescriptorSchema"
```


The 'xmlns' attribute allows to declare the URL of the default namespace. It means that the XML elements used in the FFF descriptor must have been defined in this namespace.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

This namespace has several attributes which allows to declare the schema to use to validate the file.

```
xsi:schemaLocation=
"http://www.squashtest.org/SquashTA/FFFDestructorSchema
http://www.squashtest.org/docta/schemas/FFFDestructorSchema_1.1.xsd"
```

The attribute 'schema location' of the namespace "**http://www.w3.org/2001/XMLSchema-instance**" allows to declare the XSD schema to use for the validation and to associate it with the previous namespace.

The others attributes of the <root> tag are used to describe the general characteristics of the FFF to verify :

- The 'name' attribute : It indicates the name of the file to use.
- The 'binary' attribute : A boolean specifying if the FFF is a binary or not.
- The 'encoding' attribute : It allows to specify the encoding of the file. Using names to design the encoding are those of the java class 'java.nio.charset.Charset'. For a binary file, this attribute is mandatory whereas for a text file it's optional because if not specified it's the encoding of the Java Virtual Machine who's used.
- The 'bytesPerLine' attribute : It allows for a binary file to specify the amount of bytes per record.

<sequences> and <sequence> elements

The <sequences> tag contains a list of <sequence> tags. Sequences are counters. They are used to incremente fields of 'autonumber' type.

```
<sequences>
  <sequence id="No" start="1" increment="1" />
</sequences>
```

With :

- id : The attribute identifying the sequence.
- start : The number from which the sequence begin.
- increment : The incrementation step of the sequence.

LeafRecord

Each <leafRecord> tag describes a leafRecord and inside each <leafRecord> tag we have <fields> and <field> tags which describe the fields of each leafrecord.

For instance :



<leafRecord> elements :

- A 'name' attribute : It designs the record name of the leaf.
- a <label> tag (Optional) : It contains the wording of the leafRecord.
- a <fields> tag which contains n <field> tags (One for each field of the leafRecord).

<field> elements :

- a <label> tag (Optional) : It contains the wording of the field.
- a <description> tag (Optional) : It contains a description of the field.
- a 'type' attribute : It indicates the type of the field.
- a 'start' attribute : It describes the position of the first character / byte inside the record (It begins at 1).
- a 'length' attribute which describes the characters / bytes number of the field.
- Depending of the field type, other attributes are available (See the fields types description section).

Composite Record

The <compositeRecord> tag contains 3 tags :

- The <openingRecord> tag which defines the opening leafRecord. The text of this tag must be linked to the value of the attribute name of the <leafRecord> tag corresponding.
- The <closingRecord> tag which defines the closing leafRecord. The text of this tag must be linked to the value of the attribute name of the <leafRecord> tag corresponding.
- The <children> tag which contains the list of the children records.

Example 1 : Composite record with closingRecord

```

<?xml version="1.0" encoding="UTF-8"?>
<root...>
  <records>
    <leaves>
      ...
      <leafRecord name="leafRecord00">
        ...
      </leafRecord>
      <leafRecord name="leafRecord99">
        ...

```

(continues on next page)

(continued from previous page)

```

    </leafRecord>
    ...
  </leaves>
  <composites>
    <compositeRecord name="personne">
      <label>Coordonnées d'une personne</label>
      <openingRecord>leafRecord00</openingRecord>
      <closingRecord> leafRecord99</closingRecord>
      <children>
        ...
      </children>
    </compositeRecord>
  </composites>
  ...

```

Example 2 : Composite Record without closingRecord

```

<?xml version="1.0" encoding="UTF-8"?>
<root...>
  <records>
    <leaves>
      <leafRecord name="leafRecord00">
        ...
      </leafRecord>
      ...
    </leaves>
    <composites>
      <compositeRecord name="personne">
        <label>Coordonnées d'une personne</label>
        <openingRecord>leafRecord00</openingRecord>
        <children>
          ...
        </children>
      </compositeRecord>
    </composites>
  </records>
</root>

```

Succession of the composite Record's children

The succession of the composite Record's children is defined with the help of a pattern built by combining 'and', 'or' and 'repeat' clauses.

'AND' Clause

The 'and' clause is used to indicate that a A-type leafRecord **AND** a B-type leafRecord (**AND** a C-type leafRecord...) must be present. The number of records included in an 'and' clause must be higher or equal to 2.

Example :

```

...
<compositeRecord name="personne">
  <label>Détail d'une personne</label>
  <openingRecord>leafRecord00</openingRecord>
  <closingRecord>leafRecord 99</closingRecord>

```

(continues on next page)

(continued from previous page)

```

    <children>
      <and>
        <record>leafRecord01<record>
        <record>leafRecord02<record>
        <record>leafRecord03<record>
      <and>
    <children>
  <compositeRecord>
  ...

```

‘or’ Clause

The ‘or’ clause is used to indicate that a A-type leafRecord **OR** a B-type leafRecord (**OR** a C-type leafRecord...) must be present. The number of records included in an ‘or’ clause must be higher or equal to 2.

Example :

```

...
<compositeRecord name="personne">
  <label>Détail d'une personne</label>
  <openingRecord>leafRecord00</openingRecord>
  <closingRecord>leafRecord 99</closingRecord>
  <children>
    <or>
      <record>leafRecord01</record>
      <record>leafRecord02</record>
      <record>leafRecord03</record>
    </or>
  <children>
</compositeRecord>
...

```

‘repeat’ Clause

The ‘repeat’ clause is used to indicate that leafRecord must be present a number of times, between a minimal and a maximal defined value (min>=0, min<max<unbounded).

Example :

```

...
<compositeRecord name="personne">
  <label>Détail d'une personne</label>
  <openingRecord>leafRecord00</openingRecord>
  <closingRecord>leafRecord99</closingRecord>
  <children>
    <repeat min="1" max="unbounded">
      <record>leafRecord01</record>
    </repeat>
  </children>
</compositeRecord>
...

```

Combination of the ‘and’, ‘or’ and ‘repeat’ clauses

The ‘and’, ‘or’ and ‘repeat’ clauses can be recursively combined.

Example :

```
...
<compositeRecord name="evenement">
  <label>Evenement</label>
  <openingRecord>100</openingRecord>
  <children>
    <and>
      <record>101</record>
      <record>102</record>
      <repeat min="1" max="8">
        <and>
          <or>
            <record>103A</record>
            <record>103D</record>
            <record>103S</record>
          </or>
          <record>105</record>
        </and>
      </repeat>
    </and>
  </children>
</compositeRecord>
...
```

Validation of the FFF descriptor

A FFF descriptor loaded by the FileChecker must be validated. Its structure is validated by the declared XSD schema and other additional validations used to ensure the functional coherence of the file.

6.3.4 Filechecker Plugin - Advanced Users

Filechecker Plugin - Converters

Contents :

- *From file To fixed.field.file.descriptor*
- *From file to fixed.field.file*
- *From file to fixed.field.file.queries*

From file To fixed.field.file.descriptor

Category-name : *descriptor*

What ?

This *descriptor* converter will convert a `file` type resource to a `fixed.field.file.descriptor` type resource. This converter verifies that the resource is a well formed descriptor (structure + content).

CONVERT {resourceToConvert<Res:file>} TO fixed.field.file.descriptor (descriptor) AS {converted<Res:fixed.field.file.descriptor>}

> Input :

- {resourceToConvert<Res:file>} : The name of the resource to convert (`file` type resource). This resource should reference a fixed field file descriptor.

> Output :

- {converted<Res:fixed.field.file.descriptor>} : The name of the converted resource (`fixed.field.file.descriptor` type resource).

Example :

LOAD path/to/fixed_field_file_descriptor.txt AS fff_descriptor.file CONVERT fff_descriptor.file TO fixed.field.file.descriptor (descriptor) AS fff_descriptor.ffff

From file to fixed.field.file

Category-name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a `fixed.field.file` type resource by using `fixed.field.file.descriptor`.

CONVERT {fffToConvert<Res:file>} TO fixed.field.file (structured) USING {fffDescriptor<Res:fixed.field.file.descriptor>} AS {converted<Res:fixed.field.file>}

> Input :

- {fffToConvert<Res:file>} : The name (in the context) of the resource to convert (`file` type resource). This resource should reference a fixed field file (e.g. by using a `LOAD` instruction on the fixed field file path).
- {fffDescriptor<Res:fixed.field.file.descriptor>} : The name (in the context) of the descriptor resource (`fixed.field.file.descriptor` type resource).

> Output :

- {converted<Res:fixed.field.file>} : The name of the converted resource (fixed.field.file type resource).

Example :

```
LOAD path/to/fixed_field_file_descriptor.txt :keywords:'AS fff_descriptor.file
CONVERT fff_descriptor.file TO fixed.field.file.descriptor (descriptor) AS fff_descriptor.ffffd
LOAD path/to/fixed_field_file.txt AS fixed_field_file.file
CONVERT fixed_field_file.file TO fixed.field.file (structured) USING fff_descriptor.ffffd AS fixed_field_file.fff
```

From file to fixed.field.file.queries

Category-name : *query*

What ?

This *query* converter will convert a *file* type resource to a *fixed.field.file.queries* type resource.

```
CONVERT    {queriesToConvert<Res:file>}    TO    fixed.field.file.queries    (query)    AS    {con-
verted<Res:fixed.field.file.queries>}
```

> Input :

- {{queriesToConvert<Res:file>}} : The name (in the context) of the resource to convert (file type resource). This resource should reference a fixed field file queries (e.g. by using a LOAD instruction on the fixed field file queries path).

> Output :

- {converted<Res:fixed.field.file.queries>} : The name of the converted resource (fixed.field.file.queries type resource).

Example :

```
LOAD path/to/fixed_field_file_queries.txt AS fff_queries.file
CONVERT fff_queries.file TO fixed.field.file.queries (query) AS fff_query.ffffq
```

Filechecker Plugin - Asserts

Contents :

- *'file' does 'contain' {regex}*
- *'file' does 'not.contain' {regex}*
- *'fixed.field.file' is 'valid'*
- *'fixed.field.file' has 'expected.autonumbers'*
- *'fixed.field.file' has 'expected.content'*

'file' does 'contain' {regex}

What ?

This assertion searches a pattern inside a `file` resource. If the pattern is not found, the assertion fails.

ASSERT {resource<file>} DOES contain THE \$(<pattern>)

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#) [this page](#).

> Input :

- {resource<Res:file>} : The name (in the context) of the file resource (`file` type resource).
- {<pattern>} : The regular expression searched in the file.

Note : If you want to check for special characters used in the regular expression formalism, you will have to escape them with a backslash ("\"").

Example :

LOAD filechecker/FFF_txt_test.txt AS fixed_field_file.file
ASSERT fixed_field_file.file DOES contain THE \$(Hello)

'file' does 'not.contain' {regex}

What ?

This assertion verifies that a pattern is not present inside a `file` resource. If the pattern is found, the assertion fails.

ASSERT {resource<Res:file>} DOES not.contain THE \$(<pattern>)

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {resource<Res:file>} : The name (in the context) of the file resource (file type resource).
- {<pattern>} : The regular expression searched in the file.

Note : If you want to check for special characters used in the regular expression formalism, you will have to escape them with a backslash ("\"").

Example :

LOAD filechecker/FFF_txt_test.txt AS fixed_field_file.file
ASSERT fixed_field_file.file DOES not.contain THE \$(Hello)

‘fixed.field.file’ is ‘valid’

What ?

This assertion verifies that a `fixed.field.file` resource is valid (structure and syntax).

ASSERT {fffResource<Res:fixed.field.file>} IS valid

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {fffResource<Res:fixed.field.file>} : The name (in the context) of the fixed field file resource to validate (fixed.field.file type resource).

Example :

LOAD filechecker/descriptor_txt_test.xml AS fixed_field_file_descriptor.file
CONVERT fixed_field_file_descriptor.file TO fixed.field.file.descriptor (descriptor) AS
fixed_field_file_descriptor.descriptor
LOAD filechecker/FFF_txt_test.txt AS fixed_field_file.file
CONVERT fixed_field_file.file TO fixed.field.file (structured) USING fixed_field_file_descriptor.descriptor AS
fixed_field_file.fff
ASSERT fixed_field_file.fff IS valid

‘fixed.field.file’ has ‘expected.autonumbers’

What ?

This assertion verifies that a `fixed.field.file` resource has the expected auto numbers.

ASSERT {fffResource<Res:fixed.field.file>} HAS expected.autonumbers

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {fffResource<Res:fixed.field.file>} : The name of the fixed field file resource to verify (fixed.field.file type resource).

Example :

LOAD filechecker/descriptor_txt_test.xml AS fixed_field_file_descriptor.file CONVERT fixed_field_file_descriptor.file TO fixed.field.file.descriptor (descriptor) AS fixed_field_file_descriptor.descriptor LOAD filechecker/FFF_txt_test.txt AS fixed_field_file.file CONVERT fixed_field_file.file TO fixed.field.file (structured) USING fixed_field_file_descriptor.descriptor AS fixed_field_file.fff ASSERT fixed_field_file.fff HAS expected.autonumbers

‘fixed.field.file’ has ‘expected.content’

What ?

This assertion verifies a `fixed.field.file` resource has the expected content. The expected content is defined in the fixed field file queries resource provided in the USING clause.

ASSERT {fffResource<Res:fixed.field.file>} HAS expected.content USING {fff- Queries<Res:fixed.field.file.queries>}

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {fffResource<Res:fixed.field.file>} : The name (in the context) of the fixed field file resource to verify (fixed.field.file type resource).
- {fffQueries<Res:fixed.field.file.queries>} : The name (in the context) of the fixed field file queries which contains the expected contents (fixed.field.file.queries type resource).

Example :

```
LOAD filechecker/descriptor_txt_test.xml AS fixed_field_file_descriptor.file
CONVERT    fixed_field_file_descriptor.file    TO    fixed.field.file.descriptor    (descriptor)    AS
fixed_field_file_descriptor.descriptor
LOAD filechecker/FFF_txt_test.txt AS fixed_field_file.file
CONVERT fixed_field_file.file TO fixed.field.file (structured) USING fixed_field_file_descriptor.descriptor AS
fixed_field_file.fff
LOAD filechecker/FFF_queries_txt_test.xml AS fixed_field_file_queries.file
CONVERT fixed_field_file_queries.file TO fixed.field.file.queries (query) AS fixed_field_file_queries.query
ASSERT fixed_field_file.fff HAS expected.content USING fixed_field_file_queries.query
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin provides all the elements needed to use the Filechecker tool (only for fixed field file currently) in SKF. Fixed Field Files are files whose position and length of different fields are known.

To use Filechecker in your Squash-TF project, you should provide the following resources :

- The file to process and its descriptor file.
- A query file if you want to do content test on the file to process.

6.4 FTP Plugin

6.4.1 FTP Plugin - Repository

What ?

Will download your resources from a FTP.

Category-name : *ftp.repository*

Configuration : A simple .properties file dropped in the 'repositories' directory of your test project. It must contain AT LEAST : 'squashtest.ta.ftp.host'.

Available properties :

- squashtest.ta.ftp.host : Supply the host name (mandatory).
- squashtest.ta.ftp.username : The username to log to.
- squashtest.ta.ftp.password : The corresponding password.
- squashtest.ta.ftp.port : An alternate command port.

- `squashtest.ta.ftp.filetype` : The default files type. Currently supported : `ascii` or `binary` (either uppercase or lowercase).
- `squashtest.ta.ftp.system` : The host system type. Currently supported : `unix`, `vms`, `windows`, `os/2`, `os/400`, `as/400`, `mvs`, `l8`, `netware`, `macos` (either uppercase or lowercase).
- `squashtest.ta.ftp.useCache` : Tells if the repository must cache its resource to increase performances. Default is `false`.

Example : valid configuration file

```
squashtest.ta.ftp.host = myhost
squashtest.ta.ftp.username = tester
squashtest.ta.ftp.password = _tester
squashtest.ta.ftp.port = 50000
squashtest.ta.ftp.filetype = ascii
squashtest.ta.ftp.system = os/400
squashtest.ta.ftp.useCache = true
```

6.4.2 FTP Plugin - Target

What ?

A ftp target is exactly what you think it is.

Category-name : *ftp.target*

Configuration : A simple `.properties` file dropped in the 'targets' directory of your test project. The file must include the shebang on the very first line : `#!/ftp`. It must also contain AT LEAST : `'squashtest.ta.ftp.host'`.

Available properties :

- `squashtest.ta.ftp.host` : Supply the host name (mandatory).
- `squashtest.ta.ftp.username` : The username to log to.
- `squashtest.ta.ftp.password` : The corresponding password.
- `squashtest.ta.ftp.port` : An alternate command port.
- `squashtest.ta.ftp.filetype` : The default files type. Currently supported : `ascii` or `binary` (either uppercase or lowercase).
- `squashtest.ta.ftp.system` : The host system type. Currently supported : `unix`, `vms`, `windows`, `os/2`, `os/400`, `as/400`, `mvs`, `l8`, `netware`, `macos` (either uppercase or lowercase).

Example of valid configuration file :

```
#!/ftp
squashtest.ta.ftp.host = myhost
squashtest.ta.ftp.username = tester
squashtest.ta.ftp.password = _tester
squashtest.ta.ftp.port = 50000
squashtest.ta.ftp.filetype = ascii
squashtest.ta.ftp.system = os/400
```

Remark

During the download from / upload on the FTP server :

- If the property 'squashtest.ta.ftp.filetype' has the value 'binary', the file is identically transfered.
- If the property has the 'ascii' value, the file is converted during the transfer (encoding, end-lines and end-file characters). The transfer mode 'ascii' allows to transfer files between heterogeneous systems. The server converts the file from its original format to a standard '8-bit NVT-ASCII' format. The client then converts the '8-bit NVT-ASCII' format file to the output format. Consequently when a binary file is transferred in 'ascii' mode, generally it's corrupted during the transfer.

6.4.3 FTP Plugin - Macros**Contents:**

- *# FTP_DOWNLOAD {remotePath} FROM {FTPserverTarget} AS {downloadedResourceName}*
- *# FTP_DOWNLOAD ASCII FILE {remotePath} FROM {FTPserverTarget} AS {downloadedResourceName}*
- *# FTP_DOWNLOAD BINARY FILE {remotePath} FROM {FTPserverTarget} AS {downloadedResourceName}*
- *# FTP_UPLOAD {localResourcePath} ONTO {FTPserverTarget} USING REMOTE PATH {remotePath}*
- *# FTP_UPLOAD ASCII FILE {localResourcePath} ONTO {FTPserverTarget} USING REMOTE PATH {remotePath}*
- *# FTP_UPLOAD BINARY FILE {localResourcePath} ONTO {FTPserverTarget} USING REMOTE PATH {remotePath}*
- *# FTP_DELETE {remotePathOfFileToDelete} FROM {FTPserverTarget}*
- *# FTP_DELETE_IF_EXISTS {remotePathOfFileToDelete} FROM {FTPserverTarget}*

FTP_DOWNLOAD {remotePath} FROM {FTPserverTarget} AS {downloadedResourceName}

What ?

This macro will download a resource from a FTP server.

Underlying instruction :

```
EXECUTE get WITH $( ) ON {FTPserverTarget} USING $(remotepath : {remotePath}) AS
↳ {downloadedResourceName}
```

> Input :

- {remotePath} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to get.
- {FTPserverTarget} : The name (in the context) of the FTP server to use (ftp.target type target).
- {downloadResourceName} : The name of the resource which references the file you download on the FTP server (file type resource).

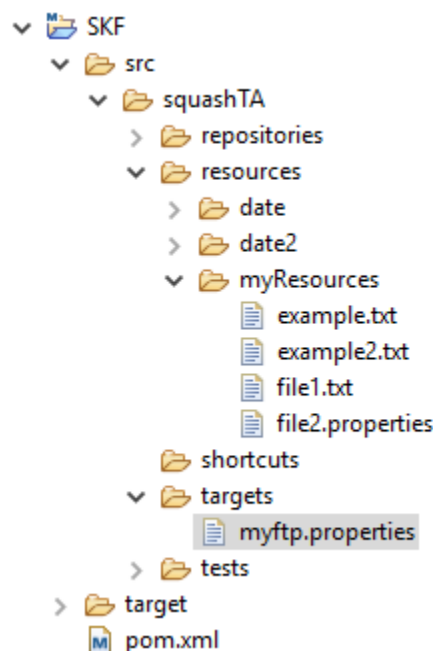
Example :

```
# FTP_DOWNLOAD path/to/example.txt FROM my-ftp-server AS example.file
```

.properties file which contains FTP information :

```
*myftp.properties 1
2 squashtest.ta.ftp.host=*yourFTPAddress*
3 squashtest.ta.ftp.username=*yourUsername*
4 squashtest.ta.ftp.password=*yourPassword*
```

.properties must be in “targets” folder of your project :



SKF script :

```
myftp.properties ftp_download.ta 1 TEST :
2
3 # FTP_DOWNLOAD example.txt FROM myftp AS example.file
```

FTP confirms that the example.txt has been downloaded :

```

(000002)11/09/2019 10:35:06 - (not logged in) (127.0.0.1)> USER dclae rhout
(000002)11/09/2019 10:35:06 - (not logged in) (127.0.0.1)> 331 Password required for dclae rhout
(000002)11/09/2019 10:35:06 - (not logged in) (127.0.0.1)> PASS *****
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 230 Logged on
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> TYPE A
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 200 Type set to A
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> PWD
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 257 "/" is current directory.
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> NOOP
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 200 OK
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> CWD /
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 250 CWD successful. "/" is current directory.
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> PASV
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 227 Entering Passive Mode (127,0,0,1,235,20)
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> RETR example.txt
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 150 Opening data channel for file download from server of "/example.txt"
(000002)11/09/2019 10:35:06 - dclae rhout (127.0.0.1)> 226 Successfully transferred "/example.txt"
(000002)11/09/2019 10:35:07 - dclae rhout (127.0.0.1)> disconnected.

```

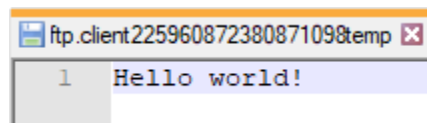
```

[INFO] You can access to the temporary files created in C:\Users\DCLEAER~1\AppData\Local\Temp\Squash_TA\20190911_103504_2333571537673605953619
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 00:00:01

```

You can see the downloaded file in SQUASH_TA temporary files folder :

Here is the downloaded file :



FTP_DOWNLOAD ASCII FILE {remotePath} FROM {FTPserverTarget} AS {downloadedResource-Name}

What ?

This macro will download an ASCII type resource from a FTP server.

Underlying instruction :

```

EXECUTE get WITH $( ) ON {FTPserverTarget} USING $(remotepath : {remotePath}, filetype_
↳ : ascii) AS {downloadedResourceName}

```

> Input :

- {remotePath} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to get.
- {FTPserverTarget} : The name (in the context) of the FTP server to use (ftp.target type target).

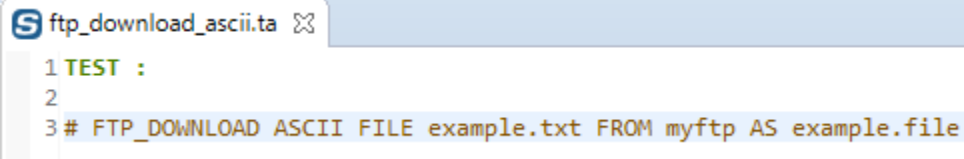
- {downloadResourceName} : The name of the resource which references the file you download on the FTP server (file type resource).

Example :

```
# FTP_DOWNLOAD ASCII FILE path/to/example.txt FROM my-ftp-server AS example.file
```

This example is based on the first one. For more details, please check [here](#).

SKF script :



```
1 TEST :
2
3 # FTP_DOWNLOAD ASCII FILE example.txt FROM myftp AS example.file
```

FTP transfer mode is set to ASCII :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test ftp_download_ascii.ta
[WARN] DEPRECATED : Using legacy 'Command get' with ignored FileResource as input (WITH clause). Please consider using the version using the built-in '{void}' VoidResource as input.
[WARN] The FTP transfer mode is set to "ascii". If you try to upload or download a binary file, it might be corrupted in the process.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCIAER-1\AppData\Local\Temp\Squash_TA\20190911_110456_1324367927290408083956
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.381 s
[INFO] Finished at: 2019-09-11T11:04:58+02:00
[INFO] Final Memory: 21M/260M
[INFO] -----
```

FTP_DOWNLOAD BINARY FILE {remotePath} FROM {FTPserverTarget} AS {downloadedResourceName}

What ?

This macro will download a binary type resource from a FTP server.

Underlying instruction :

```
EXECUTE get WITH $() ON {FTPserverTarget} USING $(remotepath : {remotePath}, filetype_
↪: binary) AS {downloadedResourceName}
```

> Input :

- {remotePath} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to get.
- {FTPserverTarget} : The name (in the context) of the FTP server to use (ftp.target type target).
- {downloadResourceName} : The name of the resource which references the file you download on the FTP server (file type resource).

Example :


```
# FTP_DOWNLOAD BINARY FILE path/to/example.bin FROM my-ftp-server AS example.file
```

This example is based on the first one. For more details, please check [here](#).

SKF script :

```
ftp_download_binary.ta
1 TEST :
2
3 # FTP_DOWNLOAD BINARY FILE example.txt FROM myftp AS example.file
```

```
# FTP_UPLOAD {localResourcePath} ONTO {FTPserverTarget} USING REMOTE PATH {remotePath}
```

What ?

This macro will upload a resource on a FTP server.

Underlying instructions :

```
LOAD {localResourcePath} AS __temp_{{randl}}.file
EXECUTE put WITH __temp_{{randl}}.file ON {FTPserverTarget} USING $(remotepath :
↳{remotePath}) AS {{whocares}}
```

> Input :

- {localResourcePath} : The path of the file you want to upload on the FTP server.
- {FTPserverTarget} : The name (in the context) of the FTP server to use (ftp.target type target)
- {remotePath} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to put.

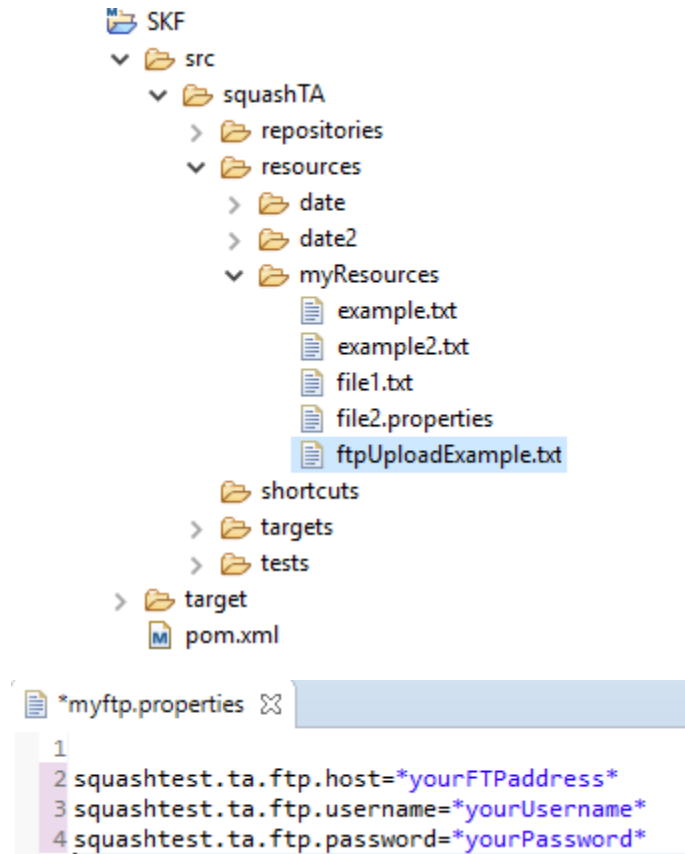
Example :

```
# FTP_UPLOAD path/to/example.txt ONTO my-ftp-server USING REMOTE PATH
abc/name.txt
```

File to upload :

```
ftpUploadExample.txt
1 I want to upload this file to my FTP.
```

File location :



.properties file which contains FTP information :

.properties must be in “targets” folder of your project :

SKF script :

FTP confirms that the uploadExample.txt has been uploaded :

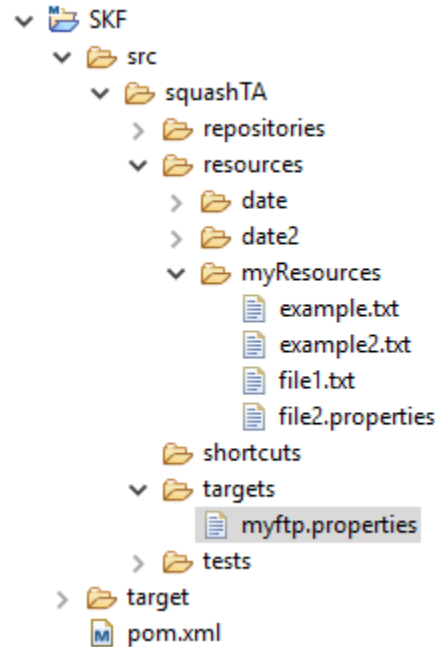
FTP_UPLOAD ASCII FILE {localResourcePath} ONTO {FTPserverTarget} USING REMOTE PATH {remotePath}

What ?

This macro will upload an ASCII type resource on a FTP server.

Underlying instruction :

```
LOAD {localResourcePath} AS __temp_{%%randl}.file  
EXECUTE put WITH __temp_{%%randl}.file ON {FTPserverTarget} USING $(remotepath :  
→{remotePath}, filetype : ascii) AS {{whocares}}
```



```

S ftp_upload.ta
1 TEST :
2
3 # FTP_UPLOAD myResources/ftpUploadExample.txt ONTO myftp USING REMOTE PATH uploadExample.txt

```

```

(000006)11/09/2019 12:01:20 - (not logged in) (127.0.0.1)> USER dclahout
(000006)11/09/2019 12:01:20 - (not logged in) (127.0.0.1)> 331 Password required for dclahout
(000006)11/09/2019 12:01:20 - (not logged in) (127.0.0.1)> PASS *****
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 230 Logged on
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> TYPE A
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 200 Type set to A
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> PWD
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 257 "/" is current directory.
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> NOOP
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 200 OK
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> CWD /
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 250 CWD successful. "/" is current directory.
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> CWD /
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 250 CWD successful. "/" is current directory.
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> PASV
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 227 Entering Passive Mode (127,0,0,1,229,157)
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> STOR uploadExample.txt
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 150 Opening data channel for file upload to server of "/uploadExample.txt"
(000006)11/09/2019 12:01:20 - dclahout (127.0.0.1)> 226 Successfully transferred "/uploadExample.txt"
(000006)11/09/2019 12:01:21 - dclahout (127.0.0.1)> disconnected.

```

SKF	Dossier de ...	09/09/2019 10:01:14
skf-usersoap5	Dossier de ...	13/06/2019 14:54:05
squash	Dossier de ...	08/08/2019 14:44:29
example.txt	12 Document ...	07/08/2019 14:56:37
hello_local_macro.txt	72 Document ...	27/08/2019 14:27:13
uploadExample.txt	37 Document ...	11/09/2019 12:01:20

> Input :

- {localResourcePath} : The path of the file you want to upload on the FTP server.
- {FTPserverTarget} : The name (in the context) of the FTP server to use (ftp.target type target)
- {remotePath} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to put.

Example :

```
# FTP_UPLOAD ASCII FILE path/to/example.txt ONTO my-ftp-server USING REMOTE PATH
abc/name.txt
```

This example is based on the previous one. For more details, please check [here](#).

SKF script :

```
S ftp_upload_ascii.ta ⓘ
1 TEST :
2
3 # FTP_UPLOAD ASCII FILE myResources/ftpUploadExample.txt ONTO myftp USING REMOTE PATH uploadExample.txt
```

FTP transfer mode is set to ASCII :

```
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test ftp upload ascii.ta
[WARN] The FTP transfer mode is set to "ascii". If you try to upload or download a binary file, it might be corrupted in the process.
[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] You can access to the temporary files created in C:\Users\DCLAER~1\AppData\Local\Temp\Squash_TA\20190911_142959_5566098133740571565297
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.235 s
[INFO] Finished at: 2019-09-11T14:30:01+02:00
[INFO] Final Memory: 20M/258M
[INFO] -----
```

```
# FTP_UPLOAD BINARY FILE {localResourcePath} ONTO {FTPserverTarget} USING REMOTE PATH
{remotePath}
```

What ?

This macro will upload a binary type resource on a FTP server.

Underlying instruction :

```
LOAD {localResourcePath} AS __temp_{%%randl}.file
EXECUTE put WITH __temp_{%%randl}.file ON {FTPserverTarget} USING $(remotepath :
↪{remotePath}, filetype : binary) AS {{whocares}}
```

> Input :

- `{localResourcePath}` : The path of the file you want to upload on the FTP server.
- `{FTPserverTarget}` : The name (in the context) of the FTP server to use (`ftp.target` type target)
- `{remotePath}` : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to put.

Example :

```
# FTP_UPLOAD BINARY FILE path/to/example.bin ONTO my-ftp-server USING REMOTE PATH
abc/name.bin
```

This example is based on the previous one. For more details, please check [here](#).

SKF script :

```
S ftp_upload_binary.ta
1 TEST :
2
3 # FTP_UPLOAD BINARY FILE myResources/ftpUploadExample.txt ONTO myftp USING REMOTE PATH uploadExample.txt
```

FTP_DELETE {remotePathOfFileToDelete} FROM {FTPserverTarget}

What ?

This macro will delete a file on a FTP server.

Underlying instruction :

```
EXECUTE delete WITH $() ON {FTPserverTarget} USING $(remotepath :
↪{remotePathOfFileToDelete}) AS {{whocares}}
```

> Input :

- `{remotePathOfFileToDelete}` : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to delete.
- `{FTPserverTarget}` : The name (in the context) of the FTP server to use (`ftp.target` type target).

Example :

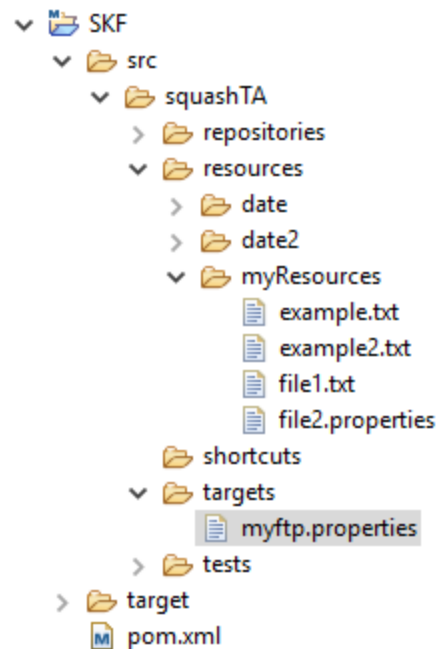
```
# FTP_DELETE distant/path/example.txt FROM my-ftp-server
```

.properties file which contains FTP information :

.properties must be in “targets” folder of your project :

SKF script :

```
*myftp.properties ⌕  
1  
2 squashtest.ta.ftp.host=*yourFTPAddress*  
3 squashtest.ta.ftp.username=*yourUsername*  
4 squashtest.ta.ftp.password=*yourPassword*
```



```
S ftp_delete.ta ⌕  
1 TEST :  
2  
3 # FTP_DELETE uploadExample.txt FROM myftp
```

FTP confirms that the file has been deleted :

```
(000011)11/09/2019 14:59:59 - (not logged in) (127.0.0.1)> USER dclerhout
(000011)11/09/2019 14:59:59 - (not logged in) (127.0.0.1)> 331 Password required for dclerhout
(000011)11/09/2019 14:59:59 - (not logged in) (127.0.0.1)> PASS *****
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> 230 Logged on
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> TYPE A
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> 200 Type set to A
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> PWD
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> 257 "/" is current directory.
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> NOOP
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> 200 OK
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> DELE uploadExample.txt
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> 250 File deleted successfully
(000011)11/09/2019 14:59:59 - dclerhout (127.0.0.1)> disconnected.
```

FTP_DELETE_IF_EXISTS {remotePathOfFileToDelete} FROM {FTPserverTarget}

What ?

This macro will delete a resource on a FTP server. If the file doesn't exist, the macro doesn't fail.

Underlying instruction :

```
EXECUTE delete WITH $() ON {FTPserverTarget} USING $(remotepath :
→{remotePathOfFileToDelete},failIfDoesNotExist:false) AS {{whocares}}
```

> Input :

- {remotePathOfFileToDelete} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to delete.
- {FTPserverTarget} : The name (in the context) of the FTP server to use (ftp.target type target).

Example :

```
# FTP_DELETE_IF_EXISTS distant/path/example.txt FROM my-ftp-server
```

This example is similar to the previous one. For more details, please check [here](#).

6.4.4 FTP Plugin - Advanced Users

FTP Plugin - Commands

Contents :

- *'put' 'file' on 'ftp'*
- *'put' 'folder' on 'ftp'*
- *'get' 'file' on 'ftp'*
- *'get' 'folder' on 'ftp'*
- *'delete' 'file' on 'ftp'*

'put' 'file' on 'ftp'

What ?

This command allows to put a file on a FTP server.

```
EXECUTE put WITH {<Res:file>} ON {<Tar:ftp.target>} AS $() USING $(remotepath : <distantPath> [,filetype : <FileType>] )
```

> Input :

- {<Res:file>} : The name of the resource (in the context) which references the file to put on the FTP server (file type resource).
- {<Tar:ftp.target>} : The name (in the context) of the FTP server to use (ftp.target type target).
- {<distantPath>} : It corresponds to the file path on the FTP server, relatively to the home directory.
- {<fileType>} : It allows to describe the type of your file. 2 values are possible : 'ascii' or 'binary'.

Remarks :

- If in the path {<distantPath>} some directories don't exist on the server so they are created.
- If the property {<filetype>} is indicated in the configuration file of the FTP target and via the instruction, the value defined in the instruction prime.

Example :

```
LOAD path/to/my_file_to_put.txt AS my_file_to_put.file  
EXECUTE put WITH my_file_to_put.file ON myFTP-server USING $( remotepath : path/to/put/distant_name.txt)  
AS $()
```

'put' 'folder' on 'ftp'

What ?

This command allows to put a folder on a FTP server.

```
EXECUTE putFolder WITH {<Res:file>} ON {<Tar:ftp.target>} AS $() USING $(remotepath : <distantPath>
[,filetype : <FileType>])
```

> Input :

- {<Res:file>} : The name of the resource (in the context) which references the folder to put on the FTP server (file type resource).
- {<Tar:ftp.target>} : The name (in the context) of the FTP server to use (ftp.target type target).
- {<distantPath>} : It corresponds to the folder path on the FTP server, relatively to the home directory.
- {<fileType>} : It allows to describe the type of your file. 2 values are possible : 'ascii' or 'binary'.

Remarks :

- If in the path {<distantPath>} some directories don't exist on the server so they are created.
- If the property {<filetype>} is indicated in the configuration file of the FTP target and via the instruction, the value defined in the instruction prime.

Example :

```
LOAD path/my_folder_to_put AS my_folder_to_put.file
EXECUTE putFolder WITH my_folder_to_put.file ON myFTP-server USING $( remotepath :
path/to/put/distant_folder_name) AS $()
```

'get' 'file' on 'ftp'

What ?

This command allows to get a file from a FTP server.

```
EXECUTE get WITH $() ON {<Tar:ftp.target>} AS {result<Res:file>} USING $(remotepath : <distantPath>
[,filetype : <FileType>])
```

> Input :

- {<Tar:ftp.target>} : The name (in the context) of the FTP server to use. (ftp.target type target)
- {<distantPath>} : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to get.
- {<fileType>} : It allows to describe the type of your file. 2 values are possible : 'ascii' or 'binary'.
- {result<Res:file>} : The name of the resource which references the file you get from the FTP server (file type resource).

Remark : If the property {<fileType>} is indicated in the configuration file of the FTP target and via the instruction, the value defined in the instruction predominates.

Example :

```
EXECUTE get WITH $() ON myFTP-server USING $( remotepath : path/to/get/FileToGet) AS getFile.file
```

‘get’ ‘folder’ on ‘ftp’

What ?

This command allows to get a folder (with all its content) from a FTP server.

```
EXECUTE getFolder WITH $() ON {<Tar:ftp.target>} AS {result<Res:file>} USING $(remotepath : <distant-Path> [,filetype : <FileType>])
```

> Input :

- {<Tar:ftp.target>} : The name (in the context) of the FTP server to use. (ftp.target type target)
- {<distantPath>} : It corresponds to the folder path on the FTP server, relatively to the home directory of the folder you want to get.
- {<fileType>} : It allows to describe the type of your file. 2 values are possible : ‘ascii’ or ‘binary’.
- {result<Res:file>} : The name of the resource which references the folder you get from the FTP server (file type resource).

Remark : If the property {<fileType>} is indicated in the configuration file of the FTP target and via the instruction, the value defined in the instruction predominates.

Example :

```
EXECUTE getFolder WITH $() ON myFTP-server USING $( remotepath : path/to/get/FileToGet) AS getFile.file
```

‘delete’ ‘file’ on ‘ftp’

What ?

This command allows to delete a file located on a FTP server.

```
EXECUTE delete WITH $() ON {<Tar:ftp.target>} AS $() USING $(remotepath : <distantPath> [,failIfDoesNotExist : false])
```

> Input :

- `{<Tar:ftp.target>}` : The name (in the context) of the FTP server to use. (`ftp.target` type `target`)
- `{<distantPath>}` : It corresponds to the file path on the FTP server, relatively to the home directory of the file you want to delete.
- `'failIfDoesNotExist: false'` : It allows to specify to **Squash TF** that the test must not fail if the file we're trying to delete doesn't exist.

Remark : `{<distantPath>}` can indicate a file OR a directory. To represent a directory, the path should end with the character `'/'`. The deletion of a directory is recursive : deletion of all sub-directories and files.

Example :

```
EXECUTE delete WITH $() ON myFTP-server USING $( remotepath : path/to/myfile.txt, failIfDoesNotExist: false) AS $()
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin provides all the elements needed to interact with a FTP server.

6.5 JUnit Plugin

6.5.1 JUnit Plugin - Resources

Contents :

- *script.junit5*
- *result.junit5*

script.junit5

Category-name : *script.junit5*

What ?

script.junit5 is a resource type used by JUnit related components. It references a java code bundle, including resources and compiled java classes.

result.junit5

Category-name : *result.junit5*

What ?

result.junit5 is a resource holding the result of any JUnit command.

6.5.2 JUnit Plugin - Macros

Contents :

- *# EXECUTE_JUNIT_TEST {displayName} FROM {qualifiedClass} IN {bundlePath}*
- *# EXECUTE_JUNIT_TEST {displayName} FROM {qualifiedClass} IN {bundlePath} WITH COMPILE OPTIONS {options}*

EXECUTE_JUNIT_TEST {displayName} FROM {qualifiedClass} IN {bundlePath}

What ?

This macro will execute a JUnit test and verify that the result is a success.

Underlying instructions :

```
LOAD {bundlePath} AS junit5{%%r1}.file
CONVERT junit5{%%r1}.file TO script.java(compile) AS junit5{%%r1}.bundle
CONVERT junit5{%%r1}.bundle TO script.junit5(structured) AS junit5{%%r1}.script
EXECUTE execute WITH junit5{%%r1}.script AS junit5{%%1}.result USING
→$(qualifiedClassName:{qualifiedClass},displayName:{displayName})
ASSERT junit5{%%1}.result IS success
```

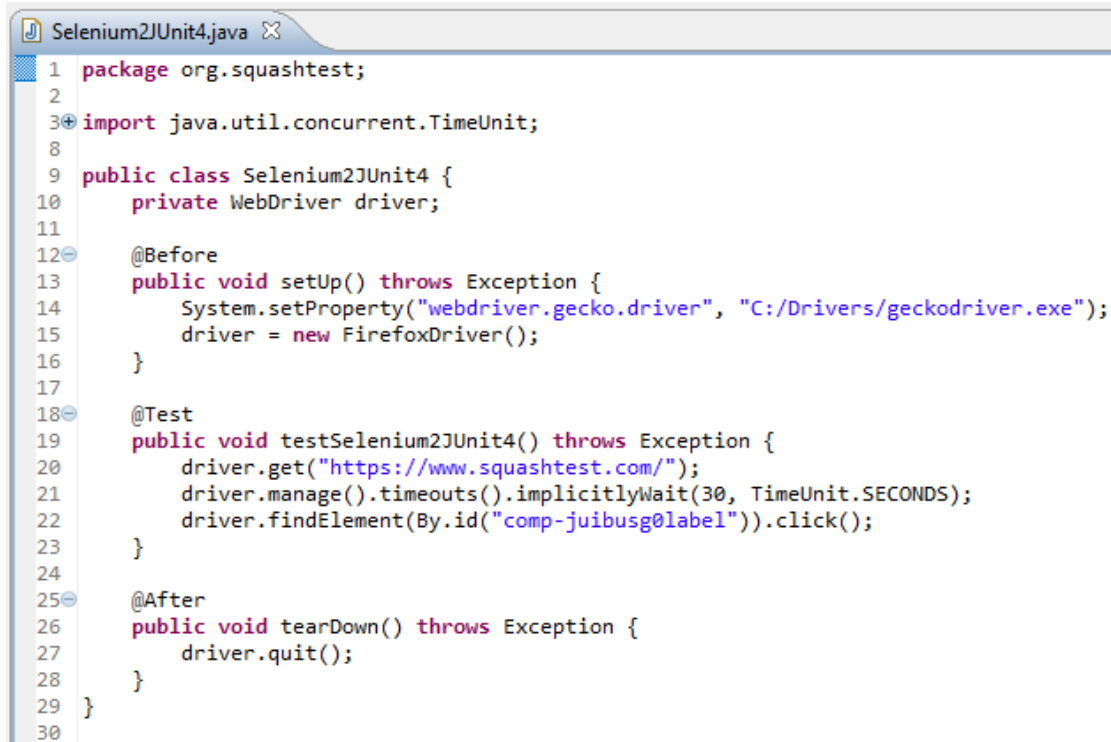
> Input :

- {qualifiedClass} : The qualified name of the class containing the test to execute.
- {displayName} : The name of the test to execute. Make sure the display name of the test is unique in the class being tested.
- {bundlePath} : The path to the java code bundle, including resources and compiled java classes.

Example :

```
# EXECUTE_JUNIT_TEST testSelenium2JUnit4 FROM org.squashtest.Selenium2JUnit4 IN selenium
```

File to process :

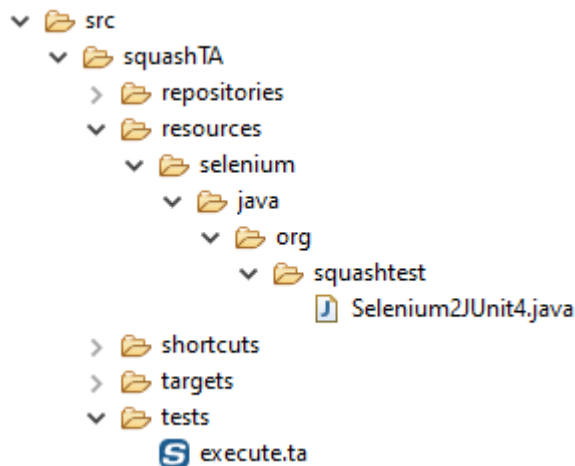


```

1 package org.squashtest;
2
3 import java.util.concurrent.TimeUnit;
4
5
6
7
8
9 public class Selenium2JUnit4 {
10     private WebDriver driver;
11
12     @Before
13     public void setUp() throws Exception {
14         System.setProperty("webdriver.gecko.driver", "C:/Drivers/geckodriver.exe");
15         driver = new FirefoxDriver();
16     }
17
18     @Test
19     public void testSelenium2JUnit4() throws Exception {
20         driver.get("https://www.squashtest.com/");
21         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
22         driver.findElement(By.id("comp-juibusg0label")).click();
23     }
24
25     @After
26     public void tearDown() throws Exception {
27         driver.quit();
28     }
29 }
30

```

The folder containing the resources to process :

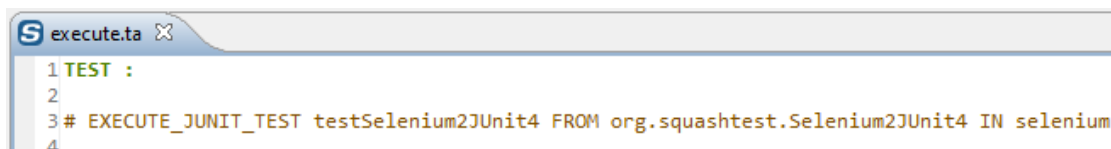


```

v src
  v squashTA
    > repositories
    v resources
      v selenium
        v java
          v org
            v squashtest
              Selenium2JUnit4.java
    > shortcuts
    > targets
  v tests
    S execute.ta

```

SKF script :



```

1 TEST :
2
3 # EXECUTE_JUNIT_TEST testSelenium2JUnit4 FROM org.squashtest.Selenium2JUnit4 IN selenium
4

```

```
# EXECUTE_JUNIT_TEST {displayName} FROM {qualifiedClass} IN {bundlePath} WITH COMPILE
OPTIONS {options}
```

What ?

This macro will execute a JUnit test, with compile options, and verify that the result is a success.

Underlying instructions :

```
LOAD {bundlePath} AS junit5{%%r1}.file
CONVERT junit5{%%r1}.file TO script.java(compile) USING {options} AS junit5{%%r1}.
↳bundle
CONVERT junit5{%%r1}.bundle TO script.junit5(structured) AS junit5{%%r1}.script
EXECUTE execute WITH junit5{%%r1}.script AS junit5{%%1}.result USING
↳$(qualifiedClassName:{qualifiedClass},displayName:{displayName})
ASSERT junit5{%%1}.result IS success
```

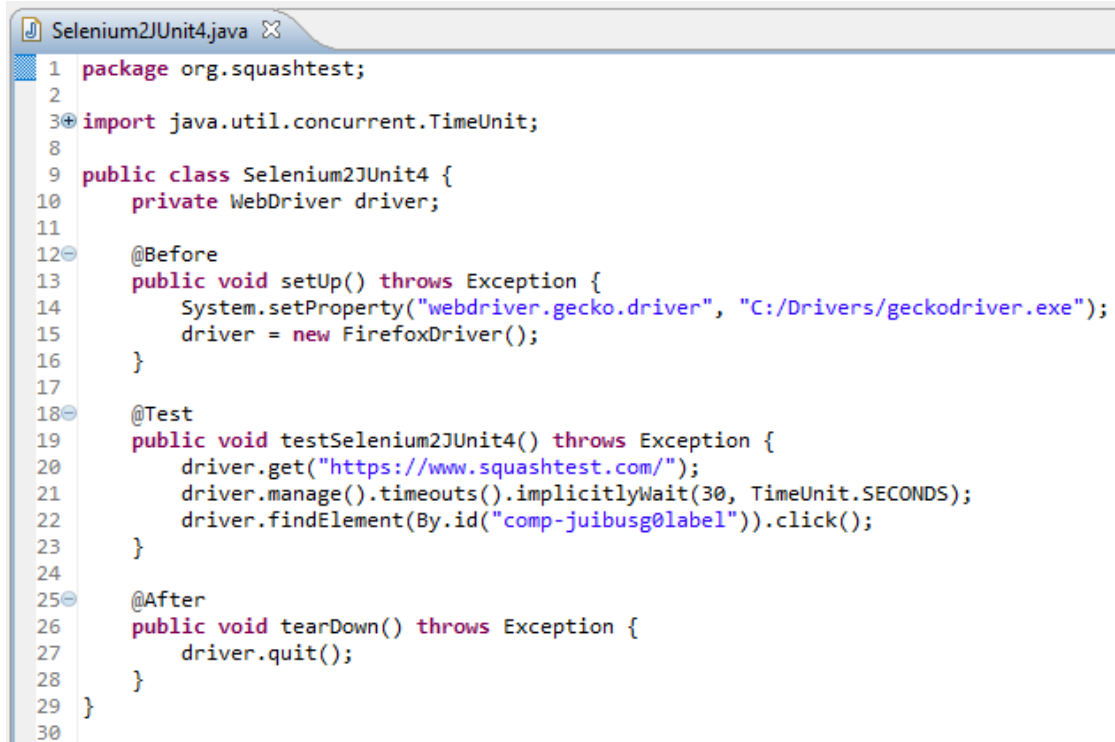
> Input :

- {qualifiedClass} : The qualified name of the class containing the test to execute.
- {displayName} : The name of the test to execute. Make sure the display name of the test is unique in the class being tested.
- {bundlePath} : The path to the java code bundle, including resources and compiled java classes.
- {options} : The name of the configuration resource. It represents a configuration file containing java compilation options (possible options are those of the Java compiler present on the machine). In this file options can be written :
 - In line separated with a space character
 - One option per line
 - A mix of both

Example :

```
# EXECUTE_JUNIT_TEST testSelenium2JUnit4 FROM org.squashtest.Selenium2JUnit4 IN sele-
nium WITH COMPILE OPTIONS compile.options.file
```

First file to process :

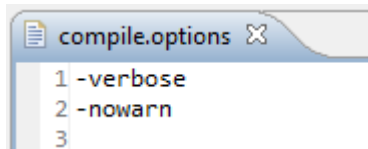


```

1 package org.squashtest;
2
3 import java.util.concurrent.TimeUnit;
4
5
6
7
8
9 public class Selenium2JUnit4 {
10     private WebDriver driver;
11
12     @Before
13     public void setUp() throws Exception {
14         System.setProperty("webdriver.gecko.driver", "C:/Drivers/geckodriver.exe");
15         driver = new FirefoxDriver();
16     }
17
18     @Test
19     public void testSelenium2JUnit4() throws Exception {
20         driver.get("https://www.squashtest.com/");
21         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
22         driver.findElement(By.id("comp-juibusg0label")).click();
23     }
24
25     @After
26     public void tearDown() throws Exception {
27         driver.quit();
28     }
29 }
30

```

Second file to process :

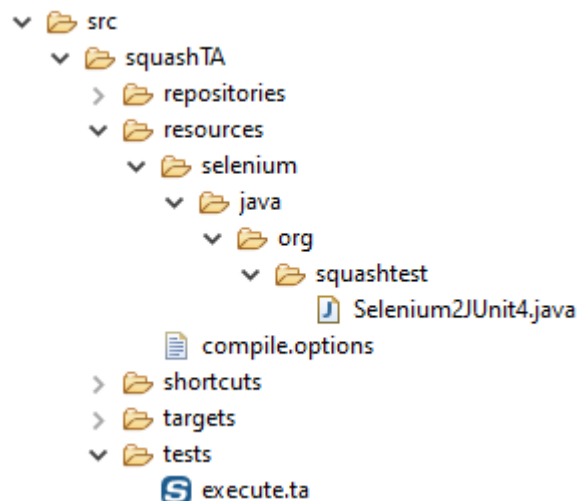


```

1 -verbose
2 -nowarn
3

```

The folder containing the resources to process :

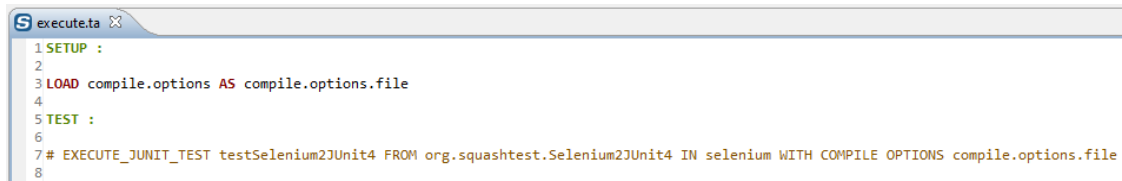


```

src
├── squashTA
│   ├── repositories
│   ├── resources
│   │   ├── selenium
│   │   │   ├── java
│   │   │   │   ├── org
│   │   │   │   │   ├── squashtest
│   │   │   │   │   │   ├── Selenium2JUnit4.java
│   │   │   │   │   │   └── compile.options
│   │   ├── shortcuts
│   │   ├── targets
│   │   └── tests
│   │       └── execute.ta

```

SKF script :



```
1 SETUP :
2
3 LOAD compile.options AS compile.options.file
4
5 TEST :
6
7 # EXECUTE_JUNIT_TEST testSelenium2JUnit4 FROM org.squashtest.Selenium2JUnit4 IN selenium WITH COMPILE_OPTIONS compile.options.file
8
```

6.5.3 JUnit Plugin - Advanced Users

JUnit Plugin - Converters

From script.java to script.junit5

Category-Name : *structured*

What ?

This *structured* converter will convert a `script.java` type resource to a `script.junit5` type resource.

CONVERT	{resourceToConvert<Res:script.java>}	TO	script.junit5	(structured)	AS	{converted<Res:script.junit5>}
---------	--------------------------------------	----	---------------	--------------	----	--------------------------------

> Input :

- `resourceToConvert<Res:script.java>` : The name of the resource which references a java code bundle, including resources and compiled java classes (`script.java` type resource).

> Output :

- `converted<Res:script.junit5>` : The name of the converted resource (`script.junit5` type resource).

Example :

LOAD path/to/java-bundle AS java-bundle.file
CONVERT java-bundle.file TO script.java (compile) AS java-bundle.script.java
CONVERT java-bundle.script.java TO script.junit5 (structured) AS java-bundle.script.junit5

JUnit Plugin - Commands

execute

What ?

Command to execute any JUnit test using the JUnit5 framework.

`EXECUTE execute WITH {<Res:script.junit5>} AS {<Res:result.junit5>} USING $({configuration})`

> Input :

- `{<Res:script.junit5>}` : The name (in the context) of the resource which references the java code bundle, including resources and compiled java classes (`script.junit5` type resource).
- `{configuration}` : The qualified display name of the test to execute, using “qualifiedClassName:<Specify name>,displayName:<Specify name>”, or its unique Id, provided by the JUnit engine, using “uniqueId:<specify Id>”.

Note 1 : These two possible configuration contents are mutually exclusive. Meaning that one has to choose how one wants to select the test(s) to execute. Either using its qualified display name or using its unique Id, but not both.

Note 2 : If you use the qualified display name, test(s) bears a conditional “s”. Indeed in this scenario one cannot guarantee the uniqueness of a display name (even qualified).

> Output :

- `{<Res:result.junit5>}` : The name (in the context) of the resource holding the result of the command (`script.junit5` type resource).

JUnit Plugin - Asserts

‘result.junit5’ is ‘success’

What ?

This assertion verifies that the result of the execution of a JUnit test is a success. If the assertion is verified the test continues, else the test fails.

`ASSERT {resourceToAssert<Res:result.junit5>} IS success`

> Input :

- `{resourceToAssert<Res:result.junit5>}` : The name of the resource to assert (`result.junit5` type resource).

Example :

```
LOAD selenium AS bundle.file
CONVERT bundle.file TO script.java (compile) AS bundle.script.java
CONVERT bundle.script.java TO script.junit5 (structured) AS bundle.script.junit5
EXECUTE execute WITH bundle.script.junit5 AS bundle.result.junit5 USING $(qualifiedClass-
Name:org.squashtest.Selenium2JUnit4,displayName:testSelenium2JUnit4)
ASSERT bundle.result.junit5 IS success
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin provides all the elements needed to execute JUnit (4 & 5) tests in SKF.

6.6 Local process Plugin

6.6.1 Local process Plugin - Resources

Contents:

- *process*
- *query.shell*
- *result.shell*

process

Category-name : *process*

What ?

The *process* resource category encapsulates a process handle to allow operations on processes. Currently only one command operates on this kind of resource: the cleanup (process) command that is designed to kill the process (this is mainly associated with environment management).

query.shell

Category-name : *query.shell*

What ?

query.shell represents a resource which contains one or several shell command lines.

Attributes : The command line supports an optional timeout attribute that specifies how long to wait before timing out during its execution. The time is measured in milliseconds (range : strictly positive up to $2^{31}-1$ (around 2 billions)).

result.shell

Category-name : *result.shell*

What ?

result.shell is a resource type that represents the result of a shell command execution (exit code, output streams...).

6.6.2 Local process Plugin - Macros

Local process Plugin - Execute Macros

Contents :

- *# EXECUTE \${command_content} LOCALLY AS {result}*
- *# EXECUTE \${command_content} LOCALLY AS {result} WITHIN {timeout} ms*
- *# EXECUTE SCRIPT {command_content} LOCALLY AS {result}*
- *# EXECUTE SCRIPT {command_content} LOCALLY AS {result} WITHIN {timeout_in_seconds} s*
- *# EXECUTE SCRIPT {command_content} LOCALLY AS {result} WITH STREAMLENGTH {length}*
- *# EXECUTE SCRIPT {command_content} LOCALLY AS {result} WITH STREAMLENGTH {length} WITHIN {timeout_in_seconds} s*

EXECUTE \$({command_content}) LOCALLY AS {result}

What ?

This macro will execute an inline command on the local system and check if the result is a success.

Underlying instructions :

```
DEFINE $({command_content}) AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell AS __commandLine{%%rand2}
EXECUTE local WITH __commandLine{%%rand2} AS {result}
ASSERT {result} IS success
```

> Input :

- {command_content} : The shell command to execute, preceded by a call to the shell (“cmd.exe /C” for Windows, “/bin/sh -c” for Linux).

> Output :

- {result} : The name of the resource which references the result of the command (result.shell type resource).

Example :

```
# EXECUTE $(cmd.exe /C echo “hello world”) LOCALLY AS result
```

EXECUTE \$({command_content}) LOCALLY AS {result} WITHIN {timeout} ms

What ?

This macro will execute an inline command on the local system, within a timeframe, and check if the result is a success.

Underlying instructions :

```
DEFINE $({command_content}) AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell AS __commandLine{%%rand2} USING $(timeout:
↪{timeout})
EXECUTE local WITH __commandLine{%%rand2} AS {result}
ASSERT {result} IS success
```

> Input :

- {command_content} : The shell command to execute, preceded by a call to the shell (“cmd.exe /C” for Windows, “/bin/sh -c” for Linux).
- {timeout} : Maximal time authorized for the command execution (in milliseconds).

> Output :

- {result} : The name of the resource which references the result of the command (result.shell type resource).

Example :

```
# EXECUTE $(cmd.exe /C echo "hello world") LOCALLY AS result WITHIN 15000 ms
```

EXECUTE SCRIPT {command_content} LOCALLY AS {result}**What ?**

This macro will execute a script on the local system and check if the result is a success.

Underlying instructions :

```
LOAD {command_content} AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell AS __commandLine{%%rand2}
EXECUTE local WITH __commandLine{%%rand2} AS {result}
ASSERT {result} IS success
```

> Input :

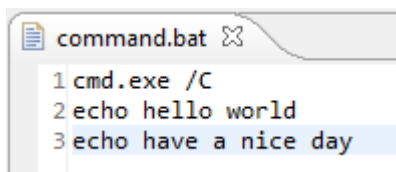
- {command_content} : The script file containing the shell commands to execute, preceded by a call to the shell ("cmd.exe /C" for Windows, "/bin/sh -c" for Linux).

> Output :

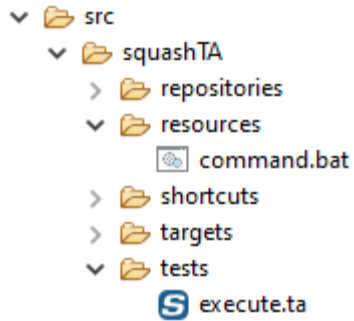
- {result} : The name of the resource which references the result of the command (result.shell type resource).

Example :

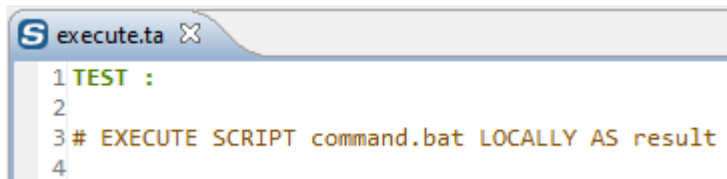
```
# EXECUTE SCRIPT command.bat LOCALLY AS result
```

File to process (Windows) :

The folder containing the resources to process :



SKF script :



EXECUTE SCRIPT {command_content} LOCALLY AS {result} WITHIN {timeout_in_seconds} s

What ?

This macro will execute a script on the local system, within a timeframe, and check if the result is a success.

Underlying instructions :

```
LOAD {command_content} AS __command{%%rand1}  
CONVERT __command{%%rand1} TO query.shell USING ${timeout:{timeout_in_seconds}000} AS_  
→__commandLine{%%rand2}  
EXECUTE local WITH __commandLine{%%rand2} AS {result}  
ASSERT {result} IS success
```

> Input :

- {command_content} : The script file containing the shell commands to execute, preceded by a call to the shell (“cmd.exe /C” for Windows, “/bin/sh -c” for Linux).
- {timeout_in_seconds} : Maximal time authorized for the command execution (in seconds).

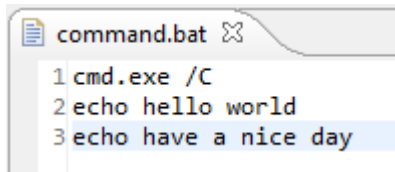
> Output :

- {result} : The name of the resource which references the result of the command (result.shell type resource).

Example :

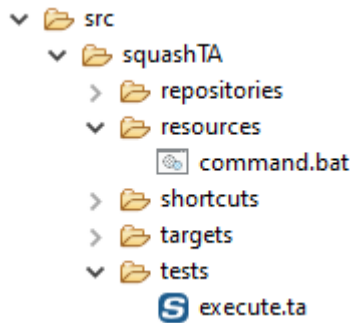
```
# EXECUTE SCRIPT command.bat LOCALLY AS result WITHIN 5 s
```

File to process (Windows) :

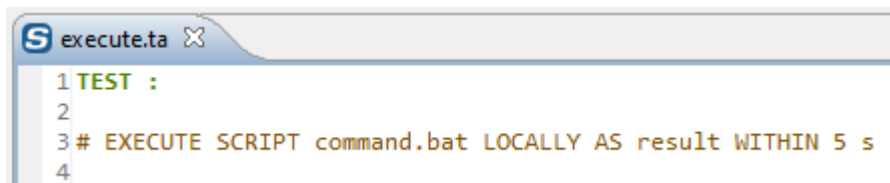


```
command.bat
1 cmd.exe /C
2 echo hello world
3 echo have a nice day
```

The folder containing the resources to process :



SKF script :



```
execute.ta
1 TEST :
2
3 # EXECUTE SCRIPT command.bat LOCALLY AS result WITHIN 5 s
4
```

EXECUTE SCRIPT {command_content} LOCALLY AS {result} WITH STREAMLENGTH {length}

What ?

This macro will execute a script on the local system, with the length of the stream specified, and check if the result is a success.

Underlying instructions :

```
LOAD {command_content} AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell USING $(streamlength:{length}) AS __
↪commandLine{%%rand2}
EXECUTE local WITH __commandLine{%%rand2} AS {result}
ASSERT {result} IS success
```

> Input :

- {command_content} : The script file containing the shell commands to execute, preceded by a call to the shell (“cmd.exe /C” for Windows, “/bin/sh -c” for Linux).

- `{length}` : An integer that represents stream length (number of characters). Specifying “full” allows to have the entire stream.

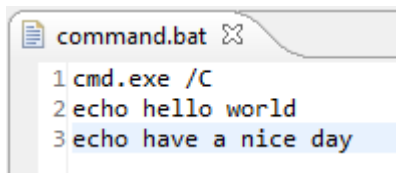
> Output :

- `{result}` : The name of the resource which references the result of the command (`result.shell` type resource).

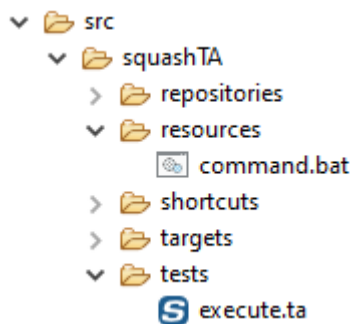
Example :

```
# EXECUTE SCRIPT command.bat LOCALLY AS result WITH STREAMLENGTH 200
```

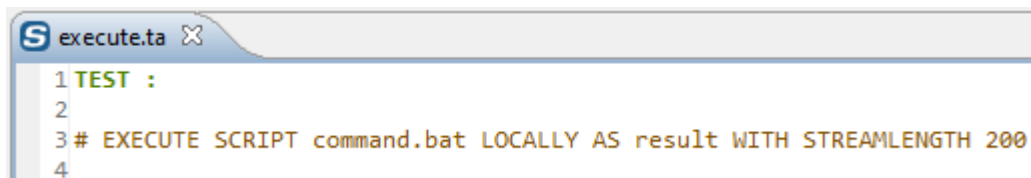
File to process (Windows) :



The folder containing the resources to process :



SKF script :



```
# EXECUTE SCRIPT {command_content} LOCALLY AS {result} WITH STREAMLENGTH {length}  
WITHIN {timeout_in_seconds} s
```

What ?

This macro will execute a script on the local system, with the length of the stream specified and within a timeframe, and check if the result is a success.

Underlying instructions :

```
LOAD {command_content} AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell USING $(timeout:{timeout_in_seconds}000,
↪streamlength:{length}) AS __commandLine{%%rand2}
EXECUTE local WITH __commandLine{%%rand2} AS {result}
ASSERT {result} IS success
```

> Input :

- {command_content} : The script file containing the shell commands to execute, preceded by a call to the shell (“cmd.exe /C” for Windows, “/bin/sh -c” for Linux).
- {length} : An integer that represents stream length (number of characters). Specifying “full” allows to have the entire stream.
- {timeout_in_seconds} : Maximal time authorized for the command execution (in seconds).

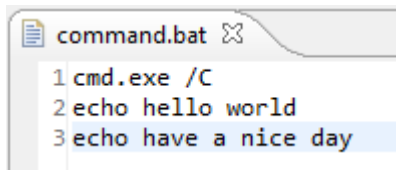
> Output :

- {result} : The name of the resource which references the result of the command (result.shell type resource).

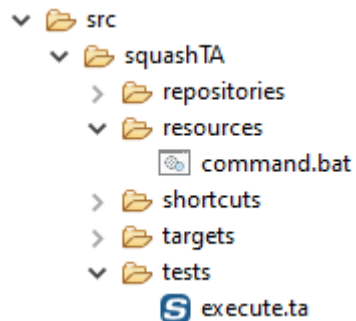
Example :

```
# EXECUTE SCRIPT command.bat LOCALLY AS result WITH STREAMLENGTH 200 WITHIN
5 s
```

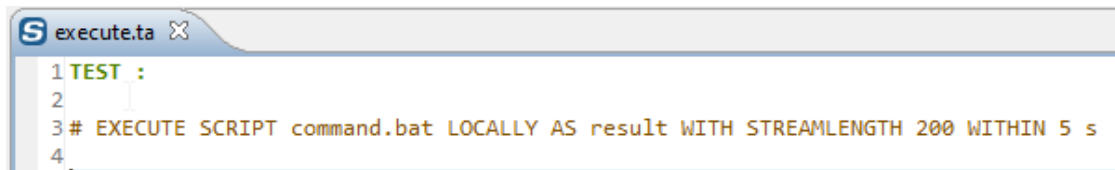
File to process (Windows) :



The folder containing the resources to process :



SKF script :



Local process Plugin - Assert Macros

Contents :

- *# ASSERT {result} IS FAILURE WITH EXIT CODE {exitCode}*
- *# ASSERT {result} STDOUT CONTAINS {regex}*
- *# ASSERT {result} STDOUT DOES NOT CONTAIN {regex}*
- *# ASSERT {result} STDERR CONTAINS {regex}*
- *# ASSERT {result} STDERR DOES NOT CONTAIN {regex}*

ASSERT {result} IS FAILURE WITH EXIT CODE {exitCode}

What ?

This macro will verify that the result of a failed execution command contains the expected exit code.

Underlying instruction :

```
ASSERT {result} IS failure WITH ${exitCode}
```

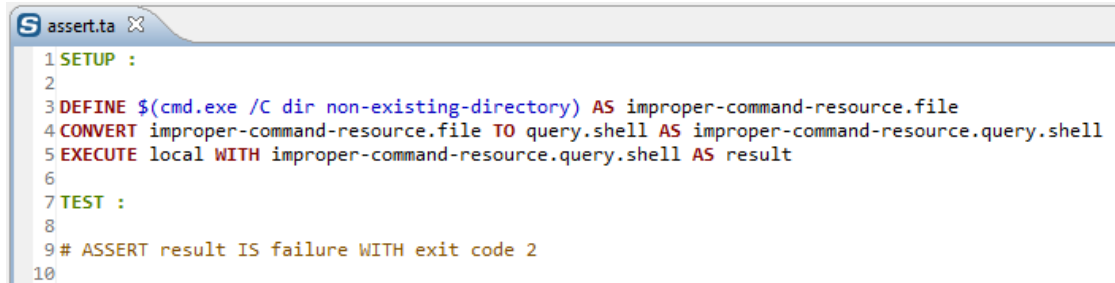
> Input :

- {result} : The resource file that contains the result of a shell execution command (result.shell type resource).
- {exitCode} : The expected return code of the command execution.

Example :

```
# ASSERT result IS failure WITH exit code 2
```

SKF script :



```

1 SETUP :
2
3 DEFINE $(cmd.exe /C dir non-existing-directory) AS improper-command-resource.file
4 CONVERT improper-command-resource.file TO query.shell AS improper-command-resource.query.shell
5 EXECUTE local WITH improper-command-resource.query.shell AS result
6
7 TEST :
8
9 # ASSERT result IS failure WITH exit code 2
10

```

Console output :

```
[ERROR] The execution failed in the TEST phase of the TA script 'execute.ta' with the message: 'Shell command should have failed with code 2 but actual code was 1'
```

ASSERT {result} STDOUT CONTAINS {regex}**What ?**

This macro will verify that the standard outflow resulting of a shell command execution contains a specific character string.

Underlying instruction :

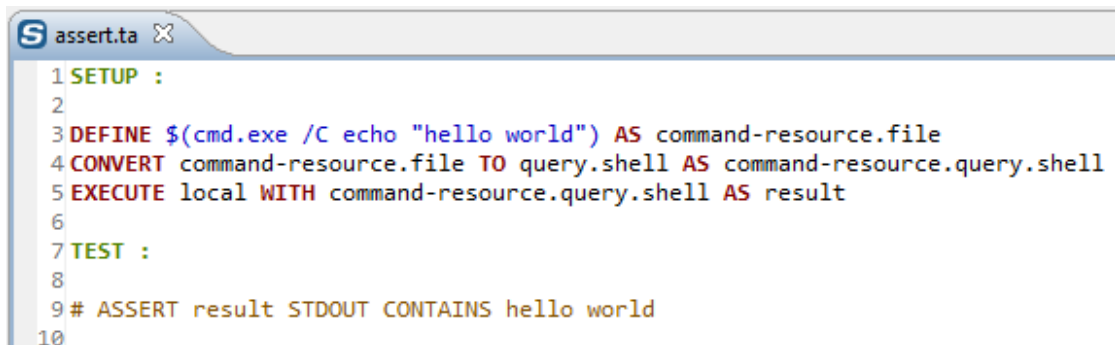
```
ASSERT {result} DOES contain WITH ${regex} USING $(out)
```

> Input :

- {result} : The resource file that contains the result of a shell execution command (result.shell type resource).
- {regex} : The searched character string.

Example :

```
# ASSERT result STDOUT CONTAINS hello world
```

SKF script :


```

1 SETUP :
2
3 DEFINE $(cmd.exe /C echo "hello world") AS command-resource.file
4 CONVERT command-resource.file TO query.shell AS command-resource.query.shell
5 EXECUTE local WITH command-resource.query.shell AS result
6
7 TEST :
8
9 # ASSERT result STDOUT CONTAINS hello world
10

```

ASSERT {result} STDOUT DOES NOT CONTAIN {regex}

What ?

This macro will verify that the standard outflow resulting of a shell command execution does not contain a specific character string.

Underlying instruction :

```
ASSERT {result} DOES not.contain WITH $({regex}) USING $(out)
```

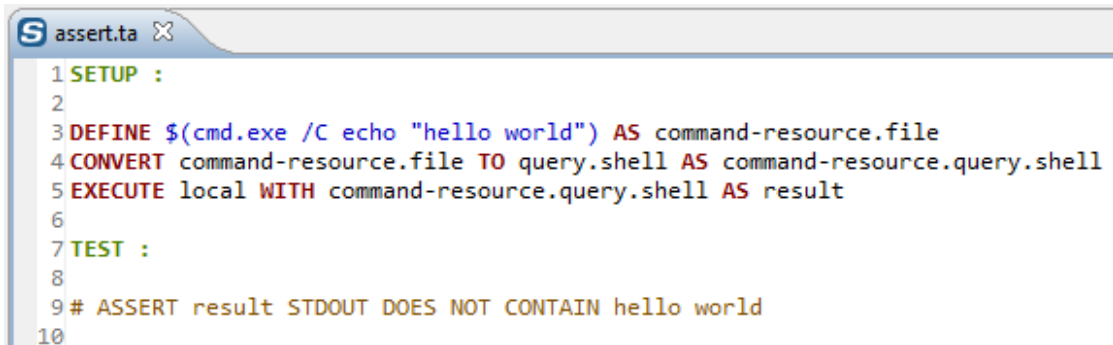
> Input :

- {result} : The resource file that contains the result of a shell execution command (result.shell type resource).
- {regex} : The searched character string.

Example :

```
# ASSERT result STDOUT DOES NOT CONTAIN hello world
```

SKF script :



```
1 SETUP :
2
3 DEFINE $(cmd.exe /C echo "hello world") AS command-resource.file
4 CONVERT command-resource.file TO query.shell AS command-resource.query.shell
5 EXECUTE local WITH command-resource.query.shell AS result
6
7 TEST :
8
9 # ASSERT result STDOUT DOES NOT CONTAIN hello world
10
```

Console output :

```
[ERROR] The execution failed in the TEST phase of the TA script 'assert.ta' with the message: 'Failure pattern 'hello world' was found.'
```

ASSERT {result} STDERR CONTAINS {regex}

What ?

This macro will verify that the error outflow resulting of a shell command execution contains a specific character string.

Underlying instruction :

```
ASSERT {result} DOES contain WITH $({regex}) USING $(err)
```

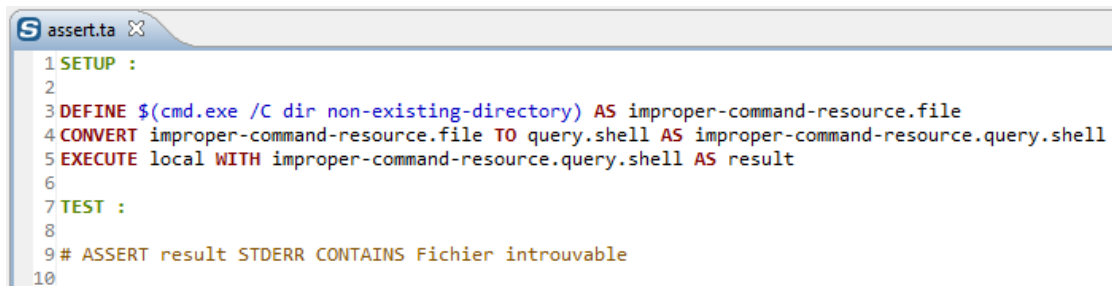
> Input :

- {result} : The resource file that contains the result of a shell execution command (result.shell type resource).
- {regex} : The searched character string.

Example :

```
# ASSERT result STDERR CONTAINS Fichier introuvable
```

SKF script :



```

1 SETUP :
2
3 DEFINE $(cmd.exe /C dir non-existing-directory) AS improper-command-resource.file
4 CONVERT improper-command-resource.file TO query.shell AS improper-command-resource.query.shell
5 EXECUTE local WITH improper-command-resource.query.shell AS result
6
7 TEST :
8
9 # ASSERT result STDERR CONTAINS Fichier introuvable
10

```

ASSERT {result} STDERR DOES NOT CONTAIN {regex}

What ?

This macro will verify that the error outflow resulting of a shell command execution does not contain a specific character string.

Underlying instruction :

```
ASSERT {result} DOES not.contain WITH $({regex}) USING $(err)
```

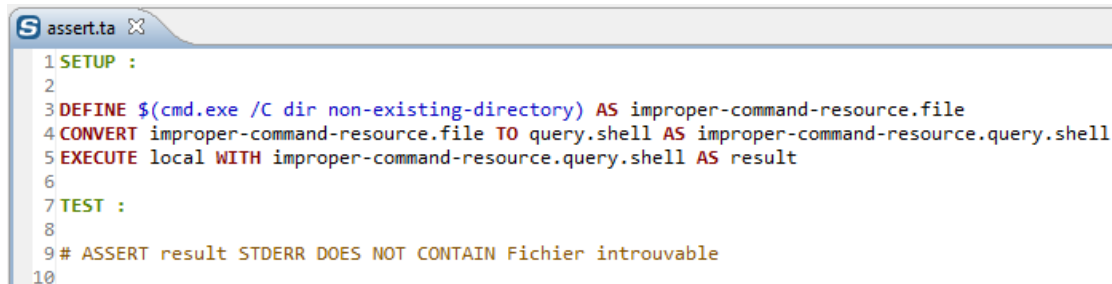
> Input :

- `{result}` : The resource file that contains the result of a shell execution command (`result.shell` type resource).
- `{regex}` : The searched character string.

Example :

`# ASSERT result STDERR DOES NOT CONTAIN Fichier introuvable`

SKF script :



```
1 SETUP :
2
3 DEFINE $(cmd.exe /C dir non-existing-directory) AS improper-command-resource.file
4 CONVERT improper-command-resource.file TO query.shell AS improper-command-resource.query.shell
5 EXECUTE local WITH improper-command-resource.query.shell AS result
6
7 TEST :
8
9 # ASSERT result STDERR DOES NOT CONTAIN Fichier introuvable
10
```

Console output :

```
[ERROR] The execution failed in the TEST phase of the TA script 'assert.ta' with the message: 'Failure pattern 'Fichier introuvable' was found.'.
```

6.6.3 Local process Plugin - Advanced Users

Local process Plugin - Converters

From file to query.shell

Category-name : *query*

What ?

This *query* converter will convert a *file* type resource to a *query.shell* type resource.

`CONVERT {resourceToConvert<Res:file>} TO query.shell (query) AS {converted<Res:query.shell>} [USING {config<Res:file>}]`

> Input :

- `{resourceToConvert<Res:file>}` : The name of the file which includes one or several shell command lines. Commands can be on one line separated by the character ‘;’ or on several lines (in this case the ‘;’ is optional) and comments beginning with ‘#’ are authorized.
- `{config<Res:file> (Optional)}` : The name of the resource which references a configuration file which contains only one key / value :

- ‘timeout’ : An integer that represents time in milliseconds. It’s the time to wait before the command execution times out. It can also be defined via an inline instruction : \$(timeout : ...).
- ‘streamlength’ : An integer that represents the stream length (number of characters). An option “full” allows to have the entire stream. It can also be defined via an inline instruction : \$(streamlength : ...). Streamlength property is available since **1.8.0** version.

> Output :

- {converted<Res:query.shell>} : The name of the converted resource (query.shell type resource).

Example :

```
LOAD shell/shell_command_03.txt AS command.file
CONVERT command.file TO query.shell USING $(timeout:15000, streamlength:600) AS commandLine
LOAD shell/shell_command_03.txt AS command.file
CONVERT command.file TO query.shell USING $(streamlength:full) AS commandLine
```

Local process Plugin - Commands**Contents :**

- *cleanup*
- ‘local’ ‘query.shell’

cleanup**What ?**

It allows killing a processus, mainly for ecosystem environment management.

```
EXECUTE cleanup WITH <Res:process> AS $()
```

> Input :

- <Res:process> : The name of the resource which references the processus to kill (process type resource).

‘local’ ‘query.shell’

What ?

Command to execute a command line on the local system.

```
EXECUTE local WITH {query<Res:query.shell>} USING $(timeout:<n>,streamlength:<n’>) AS {result<Res:Result.shell>}
```

> Input :

- {query<Res:query.shell>} : The name of the resource referencing a file which includes one (and one only) command line (query.shell type resource).
- <n> : An integer that represents time in milliseconds. It is the time the command execution will wait before crashing.
- <n’> : An integer that represents the stream length (number of characters). An option “full” allows to have the entire stream. It can be define via an inline instruction : \$(streamlength : ...). Streamlength property is available since **1.8.0** version.

Note 1 : If the timeout property is not defined here, we use the timeout property of query.shell resource (set to 5s by default).

Note 2 : Be careful : Local process is not equivalent to a console, it only executes programs. So if you want to use it as a console, you should specify which shell you want to use in your command line. Most of the time :

- For windows, start your command line with : “cmd.exe /C” (first line).
- For linux, start your command with : /bin/sh -c (according to the distribution this may be useless).

Note 3 : As local process use the underlying OS, the TA scripts which use it are platform dependent.

Note 4 : if the streamlength property is not defined here, we use the streamlength property by default (set to 300 characters).

> Output :

- {result<Res:result.shell>} : The name of the resource which contains the shell command result (result.shell type resource).

Example (Linux) :

```
DEFINE $(echo hello world) AS command.file
CONVERT command.file TO query.shell USING $(timeout:15000, streamlength:600) AS commandLine
EXECUTE local WITH commandLine AS result
ASSERT result DOES contain WITH $(hello world) USING $(out)
```

Note : To execute several command lines, you will need to execute a batch. You must then either give the absolute path of your batch or its relative path from your project’s root in the DEFINE instruction.

Example (Windows) :

```
LOAD command.bat AS command.file
CONVERT command.file TO query.shell AS commandLine
EXECUTE local WITH commandLine USING $(timeout:15000, streamlength:full) AS result
ASSERT result DOES contain WITH $(hello world) USING $(out)
ASSERT result DOES contain WITH $(nice day) USING $(out)
```


command.bat :

```
cmd.exe /C
echo hello world
echo have a nice day
```

Local process Plugin - Asserts

Contents :

- *'result.shell' is 'success'*
- *'result.shell' is 'failure' with {expected return code}*
- *'result.shell' does 'contain' {regex}*
- *'result.shell' does 'not.contain' {regex}*

'result.shell' is 'success'

What ?

Asserts that the result of the command execution is success.

```
ASSERT {result<Res:result.shell>} IS success
VERIFY {result<Res:result.shell>} IS success
```

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- `{result<Res:result.shell>}` : The name of the resource which references a resource that contains the result of a shell execution command (`result.shell` type resource).

Example :

```
LOAD shell/shell_command.txt AS command.file
CONVERT command.file TO query.shell USING $(timeout:15000) AS commandLine
EXECUTE execute WITH commandLine ON ssh_server AS result
ASSERT result IS success
```

'result.shell' is 'failure' with {expected return code}

What ?

Asserts that the return code of a command execution who failed is the expected code.

ASSERT {result<Res:result.shell>} IS failure WITH \$(<expectedCode>) VERIFY {result<Res:result.shell>} IS failure WITH \$(<expectedCode>)
--

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {result<Res:result.shell>} : The name of the resource which references a resource that contains the result of a shell execution command (result.shell type resource).
- <expectedCode> : The expected return code of the command execution.

Example :

ASSERT result IS failure WITH \$(1)

'result.shell' does 'contain' {regex}

What ?

Asserts that a stream (standard exit stream or error stream) resulting of an execution command contains a character string.

ASSERT {result<Res:result.shell>} DOES contain WITH \$(<searchPattern>) USING \$(<streamType>) VERIFY {result<Res:result.shell>} DOES contain WITH \$(<searchPattern>) USING \$(<streamType>)
--

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {result<Res:result.shell>} : The resource that contains the result of an execution command (result.shell type resource).
- <searchPattern> : The regular expression searched in the stream.
- <streamType> : The kind of stream in which we are searching the character string. 2 values are possible :

- **out** : To search inside a standard output stream.
- **err** : To search inside the error stream.

Note : If you want to check for special characters used in the regular expression formalism, you will have to escape them with a backslash (“\”).

Example :

EXECUTE execute WITH commandLine ON ssh-server AS result ASSERT result DOES contain WITH \$(hello world) USING \$(err)

‘result.shell’ does ‘not.contain’ {regex}

What ?

Asserts that a stream (standard exit stream or error stream) resulting of an execution command does not contain a specific character string.

ASSERT {result<Res:result.shell>} DOES not.contain WITH \$(<searchPattern>) USING \$(<streamType>) VERIFY {result<Res:result.shell>} DOES not.contain WITH \$(<searchPattern>) USING \$(<streamType>)
--

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {result<Res:result.shell>} : The resource that contains the result of an execution command (result.shell type resource).
- <searchPattern> : The regular expression searched in the stream.
- <streamType> : The kind of stream in which we are searching the character string. 2 values are possible :
 - **out** : To search inside a standard output stream.
 - **err** : To search inside the error stream.

Note : If you want to check for special characters used in the regular expression formalism, you will have to escape them with a backslash (“\”).

Example :

EXECUTE execute WITH commandLine ON ssh-server AS result ASSERT result DOES not.contain WITH \$(hello world) USING \$(err)

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin enables the possibility to execute processes on the local system.

Note : Be careful, as local process use the underlying OS, the TA scripts which use it are platform dependent.

6.7 MEN XML Checker Plugin

6.7.1 MEN XML Checker Plugin - Resources

xsd

Category-name : *xsd*

What ?

xsd is a resource representing a xml schema type file.

6.7.2 MEN XML Checker Plugin - Macros

Contents :

- *# ASSERT_XML {xml_path} IS VALID USING XSD {xsd_path}*
- *# ASSERT_XML {actual_file} SIMILAIRE TO {expected_file_path} USING {config}*

ASSERT_XML {xml_path} IS VALID USING XSD {xsd_path}

What ?

This macro apply will verify if an xml file is valid according to a schema (xsd type file).

Underlying instructions :

```

LOAD {xsd_path} AS __{%%r1}.xsdfile

LOAD {xml_path} AS __xml_{%%r2}.file
CONVERT __xml_{%%r2}.file TO file(param.relativeDate) AS __temp_{%%r3}.file
CONVERT __temp_{%%r3}.file TO xml (structured) AS __temp_{%%r4}.xmlfile

ASSERT __temp_{%%r4}.xmlfile IS valid USING __{%%r1}.xsdfile

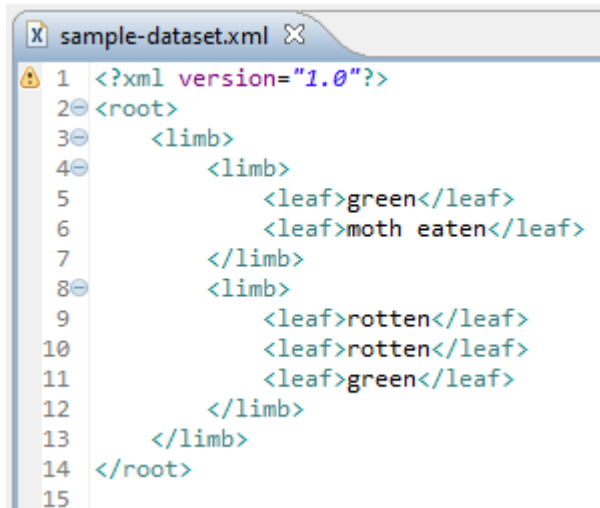
```

> Input :

- {xsd_path} : The name of the schema (xsd type file).
- {xml_path} : The name of the xml file to verify (xml type file).

Example :

```
# ASSERT_XML sample-dataset.xml IS VALID USING XSD reference-file.xsd
```

First file to process :


```

1 <?xml version="1.0"?>
2 <root>
3   <limb>
4     <limb>
5       <leaf>green</leaf>
6       <leaf>moth eaten</leaf>
7     </limb>
8   <limb>
9     <leaf>rotten</leaf>
10    <leaf>rotten</leaf>
11    <leaf>green</leaf>
12  </limb>
13 </limb>
14 </root>
15

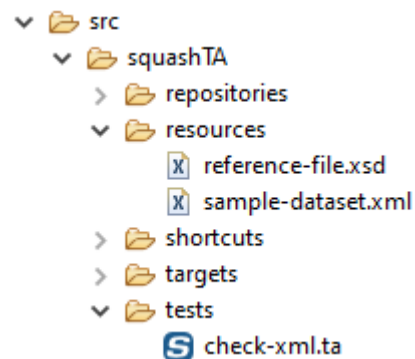
```

Second file to process :



```
1 <xsd:schema
2   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3   elementFormDefault="unqualified">
4   <xsd:element name="root" type="thingyRoot" />
5   <xsd:complexType name="thingyRoot">
6     <xsd:sequence>
7       <xsd:element name="limb" type="limb"/>
8     </xsd:sequence>
9   </xsd:complexType>
10  <xsd:complexType name="limb">
11    <xsd:sequence>
12      <xsd:choice minOccurs="0" maxOccurs="unbounded">
13        <xsd:element name="limb" type="limb" />
14        <xsd:element name="leaf" type="xsd:string" />
15      </xsd:choice>
16    </xsd:sequence>
17  </xsd:complexType>
18 </xsd:schema>
19
```

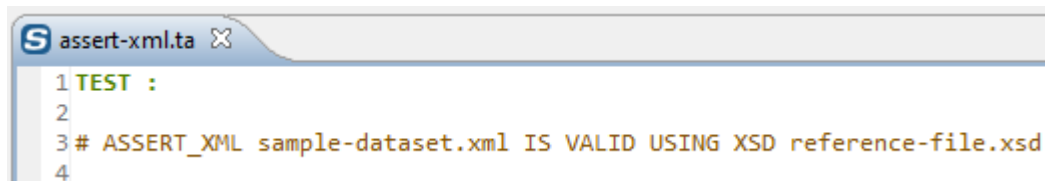
The folder containing the resources to process :



```

v src
  v squashTA
    > repositories
    v resources
      reference-file.xsd
      sample-dataset.xml
    > shortcuts
    > targets
    v tests
      S check-xml.ta
```

SKF script :



```
S assert-xml.ta
1 TEST :
2
3 # ASSERT_XML sample-dataset.xml IS VALID USING XSD reference-file.xsd
4
```

ASSERT_XML {actual_file} SIMILAIRE TO {expected_file_path} USING {config}

What ?

This macro apply will verify if an xml file matches another xml. A config file can be used to change the comparison engine.

Underlying instructions :

```
LOAD {actual_file} AS __actual_{%r1}.file
LOAD {expected_file_path} AS __expected_{%r1}.file
CONVERT __expected_{%r1}.file TO file(param.relativepath) AS __expected_{%r2}.file
CONVERT __expected_{%r2}.file TO xml (structured) AS __expected_{%r3}.xmlfile

CONVERT __actual_{%r1}.file TO xml (structured) AS __actual_{%r4}.xmlfile

ASSERT __expected_{%r3}.xmlfile IS similaire WITH __actual_{%r4}.xmlfile USING
↳{config}
```

> Input :

- {actual_file} : The name of the file to compare (xml type file).
- {expected_file_path} : The name of the file to be compared to (xml type file).
- {config} : The name of the loaded configuration resource (file type resource). It can be used to change the default comparison engine from jaxb to xmlunit, through a 'comparateur:xmlunit' entry. The default comparator can also be changed directly with \$(comparateur:xmlunit). A xsd resource can be specified here if using the jaxb comparator.

Example :

```
# ASSERT_XML sample-dataset-1.xml SIMILAIRE TO sample-dataset-2.xml USING config-
resource.file
```

First file to process :



Second file to process :

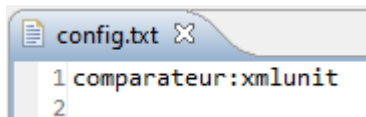


```

1 <?xml version="1.0"?>
2 <root>
3   <limb>
4     <limb>
5       <leaf>moth eaten</leaf>
6       <leaf>green</leaf>
7     </limb>
8   <limb>
9     <leaf>green</leaf>
10    <leaf>moth eaten</leaf>
11  </limb>
12 </limb>
13 </root>
14

```

Third file to process :

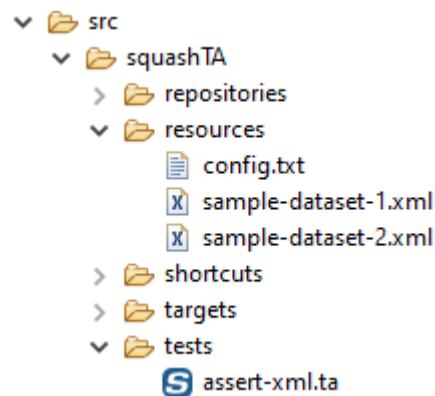


```

1 compareur:xmlunit
2

```

The folder containing the resources to process :

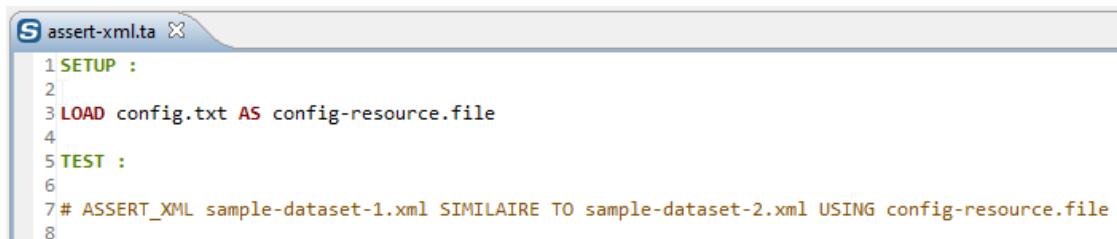


```

v src
  v squashTA
    > repositories
    v resources
      config.txt
      sample-dataset-1.xml
      sample-dataset-2.xml
    > shortcuts
    > targets
    v tests
      S assert-xml.ta

```

SKF script :

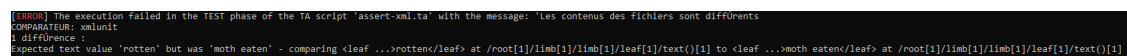


```

1 SETUP :
2
3 LOAD config.txt AS config-resource.file
4
5 TEST :
6
7 # ASSERT_XML sample-dataset-1.xml SIMILAIRE TO sample-dataset-2.xml USING config-resource.file
8

```

Console output in DEBUG mode :



```

[ERROR] The execution failed in the TEST phase of the TA script 'assert-xml.ta' with the message: 'Les contenus des fichiers sont différents'
comparateur: xmlunit
a différence :
Expected text value 'rotten' but was 'moth eaten' - comparing <leaf ...>rotten</leaf> at /root[1]/limb[1]/limb[1]/leaf[1]/text()[1] to <leaf ...>moth eaten</leaf> at /root[1]/limb[1]/limb[1]/leaf[1]/text()[1]

```


6.7.3 MEN XML Checker Plugin - Advanced Users

MEN XML Checker Plugin - Converters

From File to XSD

Category-Name : *structured*

What ?

This *structured* converter will convert a `file` type resource to a `xsd` type resource.

```
CONVERT {resourceToConvert<Res:file>} TO xsd (structured) AS {converted<Res:xsd>}
```

> Input :

- `resourceToConvert<Res:file>`: The name of the resource to convert (`file` type resource).

> Output :

- `converted<Res:xsd>`: The name of the converted resource (`xsd` type resource).

Example :

```
LOAD schema.xsd AS schema-resource.file
CONVERT schema-resource.file TO xsd (structured) AS schema-resource.xsd
```

MEN XML Checker Plugin - Asserts

Contents :

- *'file.xml' is 'valid'*
- *'file1.xml' is 'similaire' with 'file2.xml'*

'file.xml' is 'valid'

What ?

This assertion verifies if a xml file is valid. If the assertion is verified the test continues, else the test fails.

`ASSERT {resourceToAssert<Res:xml>} IS valid`

> Input :

- `{resourceToAssert<Res:xml>}` : The name of the resource to assert (xml type resource).

Example :

```
LOAD schema.xsd AS schema-resource.file
LOAD sample.xml AS sample-resource.file
CONVERT sample-resource.file TO file (param.relativedate) AS sample-resource-relative-date.file
CONVERT sample-resource-relative-date.file TO xml (structured) AS sample-resource-relative-date.xml
ASSERT sample-resource-relative-date.xml IS valid USING schema-resource.file
```

'file1.xml' is 'similaire' with 'file2.xml'

What ?

This assertion verifies if a xml file matches another xml file. If the assertion is verified the test continues, else the test fails.

`ASSERT {resourceToAssert<Res:xml>} IS similaire WITH {resourceToCompareTo<Res:xml>} USING {config}`

> Input :

- `{resourceToAssert<Res:xml>}` : The name of the resource to assert (xml type resource).
- `{resourceToCompareTo<Res:xml>}` : The name of the resource to compare to (xml type resource).
- `{config}` : The name of the loaded configuration resource (file type resource). It can be used to change the default comparison engine from jaxb to xmlunit, through a 'comparateur:xmlunit' entry. The default comparator can also be changed directly with `$(comparateur:xmlunit)`. A `xsd` resource can be specified here if using the jaxb comparator.

Example :

```
LOAD actual.xml AS actual-resource.file
CONVERT actual-resource.file TO xml (structured) AS actual-resource.xml
LOAD expected.xml AS expected-resource.file
CONVERT expected-resource.file TO file (param.relativedate) AS expected-resource-relative-date.file
CONVERT expected-resource-relative-date.file TO xml (structured) AS expected-resource-relative-date.xml
ASSERT actual-resource.xml IS similaire WITH expected-resource-relative-date.xml USING $(comparateur:xmlunit)
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin allows xml validation and comparison.

6.8 SAHI Plugin

6.8.1 SAHI Plugin - Resources

Contents :

- *script.sahi*
- *result.sahi*

script.sahi

Category-name : *script.sahi*

What ?

script.sahi is a resource type that represents a script for Sahi robot which is designed to test Web GUI. This resource can be used for standalone sahi scripts but also for sahi scripts with dependencies (e.g. on other sahi script or file to download).

result.sahi

Category-name : *result.sahi*

What ?

result.sahi is a resource type that represents the status of a Sahi script. Sahi scripts usually embed their own assertions and their results are fed back to SKF using this resource. It allows further processing once the script is over while holding the results in memory.

6.8.2 SAHI Plugin - Macros

Contents :

- *# EXECUTE_SAH {script} ON {server} USING {conf}*
- *# EXECUTE_SAH_BUNDLE {bundlepath} WITH MAIN SCRIPT {scriptpath} ON {server} USING {conf}*

EXECUTE_SAH {script} ON {server} USING {conf}

What ?

This macro will load a sahi script and a sahi configuration file. Then it will execute the script against the server using the configuration. Finally it will test if the result of the operation is a success. Note that the given sahi script must be standalone : it must not depend on any other files, e.g. script includes or files to be uploaded. When this is the case, you must use the following shortcut instead.

Underlying instructions :

```
LOAD {script} AS __temp{%%rand1}.file
CONVERT __temp{%%rand1}.file TO file(param.relativepath) AS __temp{%%rand2}.file
CONVERT __temp{%%rand2}.file TO script.sahi(script) AS __temp{%%rand3}.sahi

LOAD {conf} AS __temp{%%rand4}.file

EXECUTE execute WITH __temp{%%rand3}.sahi ON {server} USING __temp{%%rand4}.file AS __
↳temp{%%rand5}.result
ASSERT __temp{%%rand5}.result IS success
```

> Input :

- {script} : The path to the sahi script to execute relative to the root of the repository (standalone script).
- {server} : The name (in the context) of the target which corresponding to the SUT (http type target).
- {conf} : The path to the sahi configuration file relative to the root of the repository.

Example :

```
#EXECUTE_SAH gui-test/sahi/main/test1.sah ON SUT_website USING gui-test/sahi/conf/basic-conf.properties
```

EXECUTE_SAHY_BUNDLE {bundlepath} WITH MAIN SCRIPT {scriptpath} ON {server} USING {conf}**What ?**

This macro executes a sahi script using a sahi configuration file and against a server, just like above. The difference is the script is now allowed to have dependencies (dependencies should be provided).

Underlying instructions :

```
LOAD {bundlepath} AS __temp_{%%r1}.file
CONVERT __temp_{%%r1}.file TO file(param.relativedate) AS __temp_{%%r2}.file
CONVERT __temp_{%%r2}.file TO script.sahi AS __temp_{%%r3}.bundle USING $(mainpath :
↪{scriptpath})

LOAD {conf} AS __temp_{%%r4}.file
CONVERT __temp_{%%r4}.file to properties(structured) AS __temp_{%%r5}.properties

EXECUTE execute WITH __temp_{%%r3}.bundle ON {server} USING __temp_{%%r5}.properties_
↪AS __temp_{%%r6}.result
ASSERT __temp_{%%r6}.result IS success
```

> Input :

- {bundlepath} : The path to the root directory of the bundle relative to the root of the repository (see bundle resource in commons component reference documentation). This root directory should contain in the script and all its dependencies.
- {scriptpath} : The path to the main script in the bundle (RELATIVE to the bundle path).
- {server} : The name (in the context) of the target corresponding to the SUT (http type target).
- {conf} : The path to the sahi configuration file relative to the root of the repository.

Example :

```
#EXECUTE_SAHY_BUNDLE gui-test/sahi WITH MAIN SCRIPT main/test1.sah ON SUT_website USING gui-
test/sahi/conf/basic-conf.properties
```

6.8.3 SAHI Plugin - Advanced Users**SAHI Plugin - Converters****Contents :**

- *From file to script.sahi*
 - *Standalone Sahi script*
 - *Sahi script with dependencies*

From file to script.sahi

Category-name : *script*

What ?

This *script* converter will convert a *file* type resource to a *script.sahi* type resource.

Two cases :

Standalone Sahi script

In the case of a standalone script, it doesn't depend on other files (e.g. other sahi script to be included or files to be downloaded). The initial resource references the sahi script. You could use the syntax below :

```
CONVERT {scriptToConvert<Res:file>} TO script.sahi (script) AS {converted<Res:script.sahi>}
```

> Input :

- {scriptToConvert<Res:file>} : The name of the resource to convert (*file* type resource). This resource references a sahi script (e.g. by using a *LOAD* instruction on the sahi script path).

> Output :

- {converted<Res:script.sahi>} : The name of the converted resource (*script.sahi* type resource).

Example :

```
LOAD path/to/sahi_script.sah AS sahi_script.file  
CONVERT sahi_script.file TO script.sahi (script) AS sahi_script.sahi
```

Sahi script with dependencies

In the case of a script which depends on other file, you have to use the syntax below. The initial resource references a directory. This directory should contains in its tree the whole resources needed by the main sahi script. You also have to define where is the main sahi script to execute (*mainScriptPath* of the *USING* clause). This path should be *RELATIVE* to the directory referenced by the initial resource.

```
CONVERT {rootDirectory<Res:file>} TO script.sahi (script) AS {converted<Res:script.sahi>} USING {main-  
ScriptPath<Res:file>}
```

> Input :

- `{rootDirectory<Res:file>}` : Name of the resource which references the root directory. This root directory should contain the whole files needed to execute the sahi script.
- `{mainScriptPath<Res:file>}` : Name of the configuration resource. The content of the file should be : 'mainpath:relativePathToMainSahiScript' (Note : you could use an inline definition). This path to main sahi script should be relative to the directory given as rootDirectory.

> Output :

- `{converted<Res:script.sahi>}` : The name of the converted resource (`script.sahi` type resource).

Example :

```
LOAD path/to/rootDirectory AS root_directory.file
CONVERT root_directory.file TO script.sahi (script) AS sahi_script.sahi USING $(main-
path:relative/path/to/main/sahi_script.sah)
```

SAHI Plugin - Command**'execute' 'script.sahi' on 'http'****What ?**

This command will execute a sahi script against an http SUT using a configuration file.

```
EXECUTE execute WITH {sahiScript<Res:script.sahi>} ON {server<Tar:http>} USING {sahiConf<Res:file>},
[{{mainPath<Res:file>}}] AS {sahiResult<Res:result.sahi>}
```

> Input :

- `{sahiScript<Res:script.sahi>}` : The name (in the context) which references either a sahi script or a bundle containing a test or a test suite to execute (`script.sahi` type resource)
- `{sahiConf<Res:file>}` : The name of the sahi configuration file (`file` type resource). The instruction supports to receive directly a `file` type resource instead of a converted resource in 'properties'. It's mandatory and can be defined via an inline instruction. The referenced file contains a list of key / value separated with the character '=' and one property per line. Possible keys are :
 - `browserType` (mandatory) : name of the browser. It should reference the "name" of a browser-Type define in the file `browser_types.xml` of the sahi proxy. You can find this file in `SAHI_PROXY_HOME/userdata/config` (The proxy should have been launched at least one time in order to the file exist). It's also possible to retrieve the content of the file through a web browser by using the url : http://SAHI_HOST:SAHI_PORT/_s_/dyn/ConfigureUI_readFile?fileName=config/browser_types.xml
 - `sahi.proxy.host` (optional) : Name of the machine where is the sahi proxy. Default value is localhost.
 - `sahi.proxy.port` (optional) : Port used by the sahi proxy. Default value is 9999.
 - `sahi.thread.nb` (optional) : Number of browser instance to launch in parallel. Default value is 1.

- `report.format` (optional) : Report type. Default value is `html`. The other possible value is `junit`.
- `timeout` (since **version 1.7.0** - optional) : The time, in milliseconds, **Squash TF** should wait before giving up the execution. Default value is set to 60 seconds (was 30s before **version 1.7.2**).
- `{mainPath<Res:file>}` (optional) : This `file` type resource is necessary to the instruction when the `script.sahi` type resource is a bundle. It can also be defined via an inline instruction. It contains only one key / value separated with the character `:` and with `'mainpath'` as the key. It corresponds to the path, relatively to the bundle root to the `sahi` file defining the `sahi` test suite.
- `{server<Tar:http>}` : The name in (the context) of the target corresponding to the SUT (`http` type target).

> Output :

- `{sahiResult<Res:result.sahi>}` : The name of the resource which contains the result of the `sahi` command execution (`sahi.result` type resource).

Example :

```
LOAD sahi-scripts/test.sah AS sahi-script.file
CONVERT sahi-script.file TO script.sahi (script) AS test.script
LOAD configuration/sahi-conf.properties AS conf
EXECUTE execute WITH test.script ON Connexion-gui USING conf AS result
```

Remark : In the case where the `script.sahi` type resource is a bundle, the instruction need the configuration key `'mainpath'`. It can be obtained via the `USING` clause or via a `script.sahi` type resource. This configuration key is optional in both instructions but must be defined in one of them. If its defined in both, so the value indicated in the command instruction prime.

SAHI Plugin - Asserts

'result.sahi' is 'success'

What ?

This assertion verifies if a `sahi` execution succeed. If the assertion is verified the test continues else the test failed.

```
ASSERT {sahiResult<Res:result.sahi>} IS success
VERIFY {sahiResult<Res:result.sahi>} IS success
```

Note : For differences between `ASSERT` and `VERIFY` assertion mode see [this page](#).

> Input :

- `{sahiResult<Res:result.sahi>}` : The name of the resource which contains the result of a `sahi` execution command (`result.sahi` type resource).

Example :

```
LOAD sahi-scripts/test.sah AS sahi-script.file
CONVERT sahi-script.file TO script.sahi (script) AS test.script
LOAD configuration/sahi-conf.properties AS conf
EXECUTE execute WITH test.script ON Connexion-gui USING conf AS result
ASSERT result IS success
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

The Sahi plugin is part of the base package shipped with SKF. It is automatically installed if you choose the default project configuration for your test project. However, as it is packaged as a separate plugin, you can exclude it from the test project (and avoid downloading and installing its dependencies).

This plugin provides all the elements needed to execute a sahi script in SKF.

Overview :

To use sahi script in your **Squash TF** project, you have to :

- Create the sahi script and integrate it (and all its dependencies) in the resources directory of your **Squash TF** project. If you have a sahi script with dependencies you will have to create a file to define the path to your script. More details below.
- Put a *sahi_config.properties* file in this resources directory (the name of this file has no importance). This file should contain the definition of the browser you want to use to execute the script.

Example of sahi_config.properties file content :

```
// For firefox your file should contain :
browserType=firefox

// For Internet explorer your file should contain :
browserType=ie
```

- Define the http Target which represents your SUT.

6.9 Selenium Plugin

6.9.1 Selenium Plugin - Introduction

Contents :

- *Overview*
- *Organizing selenium-related files in your Squash TF project*
- *Importing legacy Selenium tests*

Overview

The Selenium plugin for SKF currently allows the integration of the following types of java Selenium Tests :

- Selenium 2
 - junit 3-based tests
 - junit 4-based tests

Selenium Tests must be included in the test project as java sources, and their non-selenium dependencies (if any) packaged as maven2 dependencies and declared in the *squash-ta-mavin-plugin* dependencies. The source code and resources are compiled each time before the test is executed. For more details, see the *Converters section*.

If you are looking for information on how to import your existing selenium tests, see the *importing legacy tests section*.

NB : A working installation of the target browser is required on the machine where the tests are run.

Organizing selenium-related files in your Squash TF project

'File to script.java.selenium2' converter works from a `file` resource that is in fact a bundle (a group of files). This group of files may contain java sources and various resources. All sources and resources used by the test must be included in the file bundle.

This means that they must be all grouped in a directory that will be loaded as a `file` resource, and then converted to a `script.java.selenium2` resource.

As in usual java code and resources, the directory structure defines packages in which the classes and resources are located. As in the maven convention, source files will be searched for in the '`<base>/java`' directory, and any directory under '`<base>/java`' will define a package level. Resources will be searched for in the same way under '`<base>/resources`'.

Regarding binary dependencies (as opposed to helper source code, which must be included in the selenium test resource directory), they must be provided as maven jars included in the plugin dependencies like so :

Extract from test project pom.xml file

```

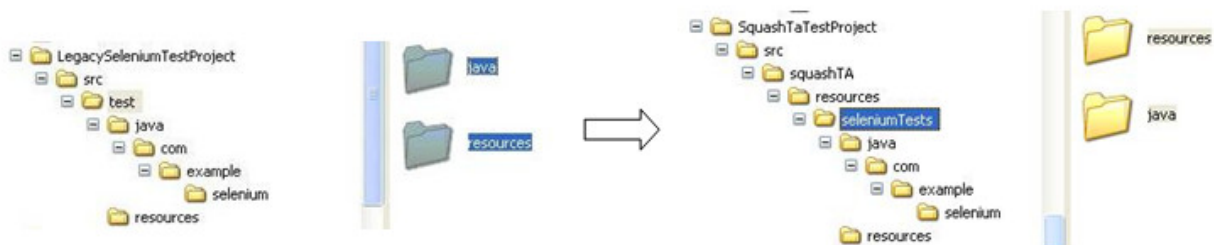
...
<plugin>
  <groupId>org.squashtest.ta</groupId>
  <artifactId>squash-ta-maven-plugin</artifactId>
  <version>1.1.0</version>

  <dependencies>
    ...
    <!-- example of a Selenium test dependency -->
    <dependency>
      <groupId>net.sourceforge.javacsv</groupId>
      <artifactId>javacsv</artifactId>
      <version>2.0</version>
    </dependency>
    ...
  </dependencies>
  ...
</plugin>
...

```

Importing legacy Selenium tests

To import your Selenium tests into your **Squash TF** test projects, just copy your test source tree (with code AND resources) under a single subdirectory in the 'squashTA/resources' directory. For example, if your test source code and resources were under 'src/test/java' and 'src/test/resources' respectively, you just have to copy the java and test directories in the 'squashTA/resources/seleniumTests' subdirectory :



Now, you just have to load 'seleniumTests' as a resource to use it in your test.

6.9.2 Selenium Plugin - Resources

Contents :

- *script.java.selenium2*
- *result.selenium*

script.java.selenium2

Category-name : *script.java.selenium2*

What ?

script.java.selenium2 is a resource type that is a Selenium 2 test written in the java language. This is basically a bundle containing the test code compiled from its sources and the associated resources. It may be used by '*execute*' '*script.java.selenium2*' command.

Here are the supported parameters :

- **mainpath** : as in the bundle resource, it defines the path to the main file from the base of the bundle (the base directory of your selenium tests). For more details, see the [Converters section](#).

result.selenium

Category-name : *result.selenium*

What ?

result.selenium is a resource type that holds the result of a Selenium test execution. It defines a Selenium execution status (success / failure), and in case of failure an attached failure report (an HTML file that follows the surefire format). This resource may be used by a specialized success assertion. For more details, see the [Assert section](#).

6.9.3 Selenium Plugin - Macros

EXECUTE_SELENIUM2 {bundlePath} WITH MAIN CLASS {mainClassName}**What ?**

This macro will compile the selenium 2 test suite contained in the specified bundle and execute the specified test suite (main class) from the bundle.

Underlying instructions :

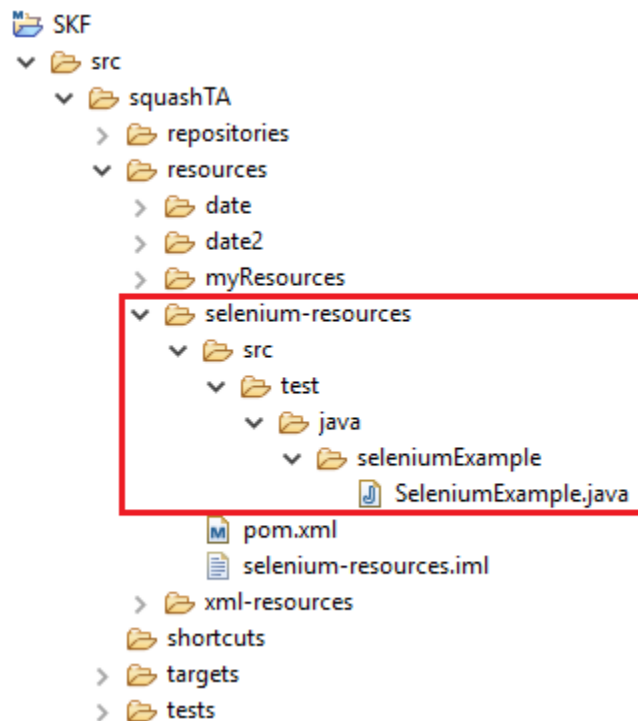
```
LOAD {bundlePath} AS __temp_{%%rand1}.file
CONVERT __temp_{%%rand1}.file TO script.java(compile) AS __temp_{%%rand2}.compiled
CONVERT __temp_{%%rand2}.compiled TO script.java.selenium2(script) USING $(
  ↳{mainClassName}) AS __temp_{%%rand3}.selenium
EXECUTE execute WITH __temp_{%%rand3}.selenium AS __temp_{%%rand4}.result
ASSERT __temp_{%%rand4}.result IS success
```

> Input :

- {bundlePath} : The path to the selenium bundle to execute relative to the resources repository. You have to point to the folder containing the java folder.
- {mainClassName} : The qualified name of the main class.

Example :

#	EXECUTE_SELENIUM2	path/to/selenium2	WITH	MAIN	CLASS
	com.example.selenium.TestSuite				

Selenium bundle location :

SKF script :

```
selenium2.ta
1 TEST :
2
3 # EXECUTE_SELENIUM2 selenium-resources/src/test WITH MAIN CLASS seleniumExample.SeleniumExample
4
```

Main class :

```
package seleniumExample;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.util.concurrent.TimeUnit;

public class SeleniumExample {

    WebDriver driver;
    WebDriverWait wait;

    @Before
    public void setUp() {
        try {
            driver = new FirefoxDriver();
            driver.get("https://www.bbc.com/news");

        }
        catch (Exception e) {
            System.err.println(e.getMessage());
        }
        driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
        wait = new WebDriverWait(driver, 30);
    }

    @Test
    public void randomTest() throws InterruptedException {
        driver.findElement(By.xpath("//input[@id='orb-search-q']")).sendKeys(
↪ "selenium");

        String value = driver.findElement(By.xpath("//button[@id=
↪ 'orb-search-button']")).getText();
        Assert.assertTrue("The search button doesn't exist", value.equals(
↪ "Search the BBC"));
    }

    @After
    public void tearDown() {
        //driver.quit();
    }
}
```

(continues on next page)

(continued from previous page)

```

    }
}

```

6.9.4 Selenium Plugin - Advanced Users

Selenium Plugin - Converters

From `script.java` to `script.java.selenium2`

Category-name : *script*

What ?

This *script* converter create a selenium test resource (`script.java.selenium2` type resource) which can be uses by the '*execute*' '*script.java.selenium2*' command from a `script.java` resource.

```

CONVERT {resourceToConvert<Res:script.java>} TO script.java.selenium2 (script) USING {conf<Res:file>} AS
{converted<Res:script.java.selenium2>}

```

> Input :

- `{resourceToConvert<Res:script.java>}` : The name (in the context) of the resource which references a java bundle which contains the source code, the compiled code and the resources of a selenium2 test suite.
- `{conf<Res:file>}` (optional) : The name of the resource which references a configuration file which can contain only one key :
 - `mainClass` : The qualified name of the main java class. The configuration must be supplied as a text file with one line containing the qualified name. If you give more, the last read line becomes the main class name. This parameter may be used if you have only one selenium test suite. On the other hand, if your selenium test bundle contains several test suite sharing helper code resources and dependencies, you may omit the main class name and rather give that parameter in the configuration of your various execute commands. It can be defined via an inline instruction.

> Output :

- `{converted<Res:script.java.selenium2>}` : The name of the converted resource (`script.java.selenium2` type resource).

Example :

```

LOAD selenium AS seleniumTestSource
CONVERT seleniumTestSource TO script.java (compile) AS seleniumTestCompiled
CONVERT      seleniumTestCompiled      TO      script.java.selenium2      (script)      USING
$(org.squashtest.Selenium2JUnit4WithResourceDependency) AS seleniumTest

```

Selenium Plugin - Commands

'execute' 'script.java.selenium2'

What ?

This command executes selenium 2 tests referenced as script.java.selenium2 resources.

EXECUTE execute WITH {selenium2Tests<Res:script.java.selenium2>} USING {conf<Res:file>} AS {result<Res:result.selenium>}
--

> Input :

- {selenium2Tests<Res:script.java.selenium2>} : The name (in the context) of the resource which references the java bundle containing the tests to execute (script.java.selenium2 type resource).
- {conf<Res:file>} (optional) : The name of the resource which references a configuration file which can contain only one key :
 - mainclass : The qualified name of the main java class. The configuration must be supplied as a text file with one line containing the qualified name. If you give more, the last read line becomes the main class name. It can be defined via an inline instruction. The format is <key>:<value>.

Note : The command needs this key. The command can retrieve it via the USING clause of the instruction or via the script.java.selenium2 type resource (See the *From script.java to script.java.selenium2* converter).

This configuration key is optional in each instruction (command and conversion) but it must be defined in at least one instruction. If the key is defined in the convert instruction and in the command instruction, the key in the command instruction prime.

> Output :

- {result<Res:result.selenium>} : The result of the test execution (result.selenium type resource).

Example :

LOAD selenium AS seleniumTestSource CONVERT seleniumTestSource TO script.java (compile) AS seleniumTestCompiled CONVERT seleniumTestCompiled TO script.java.selenium2 (script) USING \$(org.squashtest.Selenium2JUnit4) AS seleniumTest EXECUTE execute WITH seleniumTest AS seleniumResult

Selenium Plugin - Assert

‘result.selenium’ is ‘success’

What ?

This assertion checks that the selenium test suite execution was successful. If not, it gives the list of failed selenium tests in the failure message, and attaches the execution report in the surefire format produced by the selenium test suite execution as failure context resource.

```
ASSERT {seleniumResult<Res:result.selenium>} IS success
VERIFY {seleniumResult<Res:result.selenium>} IS success
```

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- {seleniumResult<Res:result.selenium>} : The name of the resource (In the context) which contains the result of a selenium execution command (result.selenium type resource).

Example :

```
LOAD selenium AS seleniumTestSource
CONVERT seleniumTestSource TO script.java (compile) AS seleniumTestCompiled
CONVERT seleniumTestCompiled TO script.java.selenium2 (script) USING
$(org.squashtest.Selenium2JUnit3WithMvnDependency) AS seleniumTest
EXECUTE execute WITH seleniumTest AS seleniumResult
ASSERT seleniumResult IS success
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

The Selenium plugin is part of the base package shipped with SKF. It is automatically installed if you choose the default project configuration for your test project. However, as it is packaged as a separate plugin, you can exclude it from the test project (and avoid downloading and installing its dependencies).

This plugin provides all the elements needed to execute selenium 2 and 3 tests in SKF.

If you need to execute selenium 1 tests, please check the [Selenium Plugin Legacy](#).

6.10 Selenium Plugin Legacy

6.10.1 Selenium Plugin Legacy - Introduction

Contents :

- [Overview](#)
- [How to use the Selenium Legacy Plugin](#)
- [Managing your selenium server in ecosystem environment scripts](#)
- [Organizing selenium-related files in your Squash-TF project](#)
- [Importing legacy Selenium tests](#)

Overview

The Selenium Plugin Legacy for SKF currently allows the integration of the following types of java Selenium Tests :

- Selenium 1
 - junit 3-based tests
 - junit 4-based tests

Selenium Tests must be included in the test project as java sources, and their non-selenium dependencies (if any) packaged as maven2 dependencies and declared in the *squash-ta-maven-plugin* dependencies. The source code and resources are compiled each time before the test is executed. For more details, see the [Converters section](#).

For Selenium 1 tests, you may at your convenience use the included selenium server management system (see [Managing your selenium server in ecosystem environment scripts](#) section), or use a selenium server already available in your testing environment.

If you are looking for information on how to import your existing selenium tests, see the [importing legacy tests section](#).

NB : a running installation of the browser used by your test is of course required on the machine your selenium RC server runs on.

How to use the Selenium Legacy Plugin

In order to be able to use the Selenium Legacy Plugin, you will need to add the following dependency inside your project pom.xml :

```
...
<plugin>
  <groupId>org.squashtest.ta</groupId>
  <artifactId>squash-ta-maven-plugin</artifactId>
  <version>${project.version}</version>
```

(continues on next page)

(continued from previous page)

```

<dependencies>
  ...
  <dependency>
    <groupId>org.squashtest.ta.plugin</groupId>
    <artifactId>squash-ta-plugin-selenium-one-legacy</artifactId>
    <version>${project.version}</version>
  </dependency>
  ...
</dependencies>
  ...
</plugin>
  ...

```

Managing your selenium server in ecosystem environment scripts

To launch and kill your selenium server as part of the ecosystem environment management, proceed as follows :

- In your ecosystem setup.ta script, insert the following code :

```
LOAD seleniumConf.properties AS seleniumConf.file CONVERT seleniumConf.file TO configuration.selenium AS seleniumConf EXECUTE launch WITH seleniumConf AS seleniumServer
```

- In your ecosystem teardown.ta script, insert the following code :

```
EXECUTE cleanup WITH seleniumServer AS ignoredResource
```

NB : Remember, resources defined in the ecosystem setup and teardown scripts can only be seen in these scripts, not in test scripts. However, a resource defined in the ecosystem setup script will be seen in the corresponding ecosystem teardown script, as the ‘seleniumServer’ resource above.

- Create the ‘seleniumConf.properties’ file in your test resources. The minimal content is as follows :

```
squashtest.ta.selenium=yes
```

NB : Any value is accepted as long as this key exists. See the `configuration.selenium` resource type documentation for useful parameters.

Organizing selenium-related files in your Squash-TF project

'File to script.java.selenium1' converter works from a `file` resource that is in fact a bundle (a group of files). This group of files may contain java sources and various resources. All sources and resources used by the test must be included in the file bundle.

This means that they must be all grouped in a directory that will be loaded as a `file` resource, and then converted to a `script.java.selenium1` resource.

As in usual java code and resources, the directory structure defines packages in which the classes and resources are located. As in the maven convention, source files will be searched for in the '`<base>/java`' directory, and any directory under '`<base>/java`' will define a package level. Resources will be searched for in the same way under '`<base>/resources`'.

Regarding binary dependencies (as opposed to helper source code, which must be included in the selenium test resource directory), they must be provided as maven jars included in the plugin dependencies like so :

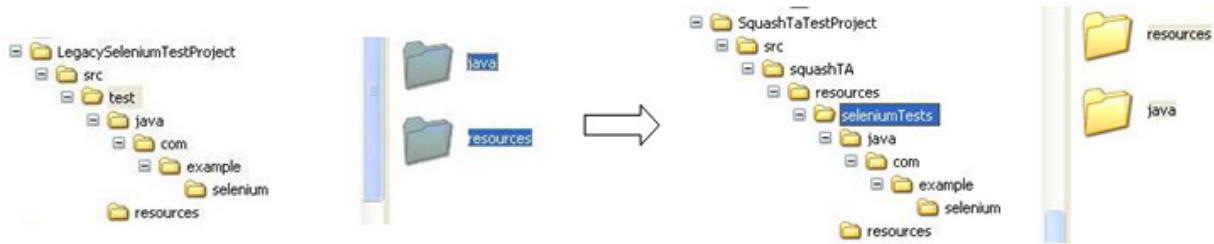
Extract from test project pom.xml file

```
...
<plugin>
  <groupId>org.squashtest.ta</groupId>
  <artifactId>squash-ta-maven-plugin</artifactId>
  <version>1.1.0</version>

  <dependencies>
    ...
    <!-- example of a Selenium test dependency -->
    <dependency>
      <groupId>net.sourceforge.javacsv</groupId>
      <artifactId>javacsv</artifactId>
      <version>2.0</version>
    </dependency>
    ...
  </dependencies>
  ...
</plugin>
...
```

Importing legacy Selenium tests

To import your Selenium tests into your **Squash TF** test projects, just copy your test source tree (with code AND resources) under a single subdirectory in the 'squashTA/resources' directory. For example, if your test source code and resources were under 'src/test/java' and 'src/test/resources' respectively, you just have to copy the java and test directories in the 'squashTA/resources/seleniumTests' subdirectory :



Now, you just have to load 'seleniumTests' as a resource to use it in your test.

6.10.2 Selenium Plugin Legacy - Resources

Contents :

- *configuration.selenium*
- *script.html.selenium*
- *script.java.selenium1*
- *result.selenium*

configuration.selenium

Category-name : *configuration.selenium*

What ?

configuration.selenium is the configuration for the Selenium Server used by 'launch' 'configuration.selenium' command.

Here are the most usefull parameters :

- *squashtest.ta.selenium* (mandatory) : The value doesn't matter.
- *squashtest.ta.selenium.port* : if you don't use the default 4444 port.
- *squashtest.ta.selenium.probe.interval* : how many milliseconds between two queries to check if the Selenium Server is online.
- *squashtest.ta.selenium.probe.attempts* : how many check queries before deciding that the server startup failed and putting the launch command in error.

script.html.selenium

Category-name : *script.html.selenium*

What ?

script.html.selenium is a resource type that is a Selenium 1 test “selenese”. This is basically a bundle.

script.java.selenium1

Category-name : *script.java.selenium1*

What ?

script.java.selenium1 is a resource type that is a Selenium 1 test written in the java language. This is basically a bundle containing the test code compiled from its sources and the associated resources. It may be used by ‘*execute*’ ‘*script.java.selenium1*’ command.

Here are the supported parameters :

- **mainpath :** as in the bundle resource, it defines the path to the main file from the base of the bundle (the base directory of your selenium tests). For more details, see the [Converters section](#).
-

result.selenium

This resource is the same than the one used in the main Selenium Plugin. For more informations, please read the following [page](#).

6.10.3 Selenium Plugin Legacy - Macros

EXECUTE_SELENIUM1 {bundlePath} WITH MAIN CLASS {mainClassName}**What ?**

This macro will compile the selenium 1 test suite contained in the specified bundle and execute the specified test suite (main class) from the bundle

Underlying instructions :

```
LOAD {bundlePath} AS __temp_{{%rand1}}.file
CONVERT __temp_{{%rand1}}.file TO script.java(compile) AS __temp_{{%rand2}}.compiled
CONVERT __temp_{{%rand2}}.compiled TO script.java.selenium1(script) USING $(
↪{mainClassName}) AS __temp_{{%rand3}}.selenium
EXECUTE execute WITH __temp_{{%rand3}}.selenium AS __temp_{{%rand4}}.result
ASSERT __temp_{{%rand4}}.result IS success
```

> Input :

- {bundlePath} : The path to the selenium1 bundle to execute relative to the root of the repository.
- {mainClassName} : The qualified name of the main class.

Example :

```
# EXECUTE_SELENIUM1 path/to/selenium1 WITH MAIN CLASS com.example.selenium.TestSuite
```

6.10.4 Selenium Plugin Legacy - Advanced Users**Selenium Plugin Legacy - Converters****Contents :**

- *From file ...*
 - ... to configuration.selenium
 - ... to script.html.selenium
- *From script.java to script.java.selenium1*

From file ...

... to configuration.selenium

Category-name : *configuration.selenium*

What ?

This converter creates a `configuration.selenium` resource which can be used to launch a Selenium Server for selenium RC tests.

```
CONVERT {resourceToConvert<Res:file>} TO configuration.selenium (configuration.selenium) AS {converted<Res:configuration.selenium>}
```

> Input :

- `{resourceToConvert<Res:file>}` : The name (in the context) of the resource which references a configuration file for a Selenium sever ('.properties').

> Output :

- `{converted<Res:configuration.selenium>}` : The name of the converted resource (`configuration.selenium` type resource).

Example :

```
LOAD selenium/selenium-conf.properties AS selenium-conf.file
CONVERT selenium-conf.file TO configuration.selenium AS selenium-conf
```

... to script.html.selenium

Category-name : *script*

What ?

This *script* converter will convert a `file` type resource to a `script.html.selenium` type resource.

```
CONVERT {resourceToConvert<Res:file>} TO script.html.selenium (script) USING {conf<Res:file>} AS {converted<Res:script.html.selenium>}
```

> Input :

- `{resourceToConvert<Res:file>}` : The name (in the context) of the resource which references the root directory containing the sources needed for a selenium test suite at the format "selenese".
- `{conf<Res:file>}` : The name of the resource which references a configuration file which can contain two keys :

- `mainpath` : The path (relative to the root directory of the bundle) to the html file containing the test suite.
- `browser` : The browser to use (to retrieve the list of possible values see : <http://stackoverflow.com/questions/1317055/how-to-run-google-chrome-with-selenium-rc>)

It can be defined via an inline instruction.

> Output :

- `{converted<Res:script.html.selenium>}` : The name of the converted resource (`script.html.selenium` type resource).

Example :

```
LOAD path/to/rootDirectory AS directory
CONVERT directory TO script.html.selenium (script) USING (mainpath:relative/path/to/suite.html) AS sele-
nese.bundle
```

From `script.java` to `script.java.selenium1`

Category-name : *script*

What?

This *script* converter create a selenium test resource (`script.java.selenium1` type resource) which can be used by the '*execute*' '*script.java.selenium1*' command from a `script.java` resource.

```
CONVERT {resourceToConvert<Res:script.java>} TO script.java.selenium1[:converter-name:'script'] USING
{conf<Res:file>} AS {converted<Res:script.java.selenium1>}
```

> Input :

- `{resourceToConvert<Res:script.java>}` : The name (in the context) of the resource which references a java bundle which contains the source code, the compiled code and the resources of a selenium1 test suite.
- `{conf<Res:file>}` (optional) : The name of the resource which references a configuration file which can contain only one key :
 - `mainClass` : The qualified name of the main java class. The configuration must be supplied as a text file with one line containing the qualified name. If you give more, the last read line becomes the main class name. This parameter may be used if you have only one selenium test suite. On the other hand, if your selenium test bundle contains several test suite sharing helper code resources and dependencies, you may omit the main class name and rather give that parameter in the configuration of your various execute commands. It can be defined via an inline instruction.

> Output :

- {converted<Res:script.java.selenium1>} : The name of the converted resource (script.java.selenium1 type resource).

Example :

```
LOAD selenium AS seleniumTestSource
CONVERT seleniumTestSource TO script.java (compile) AS seleniumTestCompiled
CONVERT seleniumTestCompiled TO script.java.selenium1 (script) USING $(org.squashtest.Selenium1JUnit3)
AS seleniumTest
```

Selenium Plugin Legacy - Commands

Contents :

- *'execute' 'script.html.selenium'*
- *'execute' 'script.java.selenium1'*
- *'launch' 'configuration.selenium'*

'execute' 'script.html.selenium'

What ?

This command executes HTML format Selenium Suites ("selenese" format).

```
EXECUTE execute WITH {seleneseTest<Res:script.html.selenium>} ON {webSUT<Tar:http>} USING
{conf<Res:file>} AS {result<Res:result.selenium>}
```

> Input :

- {seleneseTest<Res:script.html.selenium>} : The name (in the context) of the resource which references the selenium test to execute (script.html.selenium type resource).
- {webSUT<Tar:http>} : The name in (the context) of the target corresponding to the SUT (http type target).
- {conf<Res:file>} (optional) : The name of the resource which references a configuration file which can contain two keys separated with comma :
 - mainpath : The path (relative to the root directory of the bundle) to the html file containing the test suite.
 - browser : The browser to use (<http://stackoverflow.com/questions/1317055/how-to-run-google-chrome-with-selenium-rc>).

It can be define via an inline instruction. The format is <key>:<value>.

Note : The command needs the two keys (mainpath and browser). The command can retrieve them via the USING clause of the instruction or via `script.html.selenium` type resource (see the [From file to script.html.selenium](#) converter).

Those two configuration keys are optional in each instruction (command and conversion) but each one must be defined in at least one instruction. If a key is defined in the convert instruction and in the command instruction, the key in the command instruction predominates.

> Output :

- {result<Res:result.selenium>} : The result of the test execution (result.selenium type resource).

Example :

`EXECUTE execute WITH seleneseTest ON TargetWeb USING conf-file AS result`

'execute' 'script.java.selenium1'

What ?

This command executes selenium 1 tests referenced as `script.java.selenium1` resources.

`EXECUTE execute WITH {selenium1Tests<Res:script.java.selenium1>} USING {conf<Res:file>} AS {result<Res:result.selenium>}`

> Input :

- {selenium1Tests<Res:script.java.selenium1>} : The name (in the context) of the resource which references the java bundle containing the tests to execute (`script.java.selenium1` type resource).
- {conf<Res:file>} (optional) : The name of the resource which references a configuration file which can contain only one key :
 - mainclass : The qualified name of the main java class. The configuration must be supplied as a text file with one line containing the qualified name. If you give more, the last read line becomes the main class name. It can be defined via an inline instruction. The format is <key>:<value>.

Note : The command needs this key. The command can retrieve it via the USING clause of the instruction or via `script.java.selenium1` type resource (See the [From script.java to script.java.selenium1](#) converter). This configuration key is optional in each instruction (command and conversion) but it must be defined in at least one instruction. If the key is defined in the convert instruction and in the command instruction, the key in the command instruction prime.

> Output :

- {result<Res:result.selenium>} : The result of the test execution.(result.selenium type resource).

Example :

```
LOAD selenium AS seleniumTestSource
CONVERT seleniumTestSource TO script.java (compile) AS seleniumTestCompiled
CONVERT seleniumTestCompiled TO script.java.selenium1 (script) USING $(org.squashtest.Selenium1JUnit4)
AS seleniumTest
EXECUTE execute WITH seleniumTest AS seleniumResult
```

'launch' 'configuration.selenium'

What ?

To execute a selenium1 test at java format a selenium server can be started. This command launches a Selenium Server instance (formerly SeleniumRC server) following the configuration described by a `configuration.selenium` resource. This command produces a `process` type resource which can be used with the `cleanup` command to kill it.

```
EXECUTE launch WITH {seleniumServerConfig<Res:configuration.selenium>} AS {seleniumServerProcess<Res:process>}
```

> Input :

- {seleniumServerConfig<Res:configuration.selenium>} : The name (in the context) of the resource which references a configuration file permitting to start a Selenium server (`configuration.selenium` type resource).

> Output :

- {seleniumServerProcess<Res:process>} : The name (in the context) of the resource which references a process linked to the Selenium Server (`process` type resource).

Example :

```
LOAD selenium/selenium-rc-conf.properties AS selenium-rc-conf.file
CONVERT selenium-rc-conf.file TO configuration.selenium AS selenium-rc-conf
EXECUTE launch WITH selenium-rc-conf AS seleniumServer
```

Selenium Plugin Legacy - Assert

‘result.selenium’ is ‘success’

The Selenium Plugin Legacy uses the same assert than the main Selenium Plugin. For more informations, please read the following [page](#).

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

The main Selenium Plugin handles Selenium 2 and 3 but not Selenium 1 anymore. If you need the integration of Selenium 1 tests into SKF, you can use Selenium Plugin Legacy which is an add-on of the main Selenium Plugin allowing the integration of those tests.

This plugin excludes the Selenium 3 dependency from the main Selenium Plugin and uses a Selenium 2 dependency which handles Selenium 1 and 2.

This plugin provides all the elements needed to execute selenium 1 tests in SKF.

Caution: If you use the legacy plugin, you won’t be able to execute Selenium 1 and Selenium 3 in the same project.

6.11 SoapUI Plugin

6.11.1 SoapUI Plugin - Resources

Contents :

- *script.soapui*
- *result.soapui*

script.soapui

Category-name : *script.soapui*

What ?

script.soapui is a resource type that represents a SoapUI workspace. This resource can reference either a single xml workspace file as produced by SoapUI or (since **1.7**) a bundle that contains an xml workspace.

result.soapui

Category-name : *result.soapui*

What ?

result.soapui is a resource type that represents the result of the execution of SoapUI tests. It is produced by the *'execute' 'script.soapui'* command.

6.11.2 SoapUI Plugin - Macros

Contents :

- *# EXECUTE_SOAPUI {soapui_script}*
- *# EXECUTE_SOAPUI {soapui_script} WITH TEST_SUITE {testsuites}*
- *# EXECUTE_SOAPUI {soapui_script} WITH TEST_SUITE {testsuites} AND TEST_CASE {testcases}*
- *# EXECUTE_SOAPUI_BUNDLE {soapui_bundle} WITH PROJECT {projectpath}*
- *# EXECUTE_SOAPUI_BUNDLE {soapui_bundle} WITH PROJECT {projectpath} AND TEST_SUITE {testsuites}*
- *# EXECUTE_SOAPUI_BUNDLE {soapui_bundle} WITH PROJECT {projectpath} AND TEST_SUITE {testsuites} AND TEST_CASE {testcases}*

EXECUTE_SOAPUI {soapui_script}

What ?

This macro loads and executes a {soapui_script}. It then verifies the success of the execution.

Underlying instructions :

```
LOAD {soapui_script} AS __soapui_script{%%rand1}.file
CONVERT __soapui_script{%%rand1}.file TO script.soapui(structured) AS __soapui_script{
↳ %%rand2}.soapui

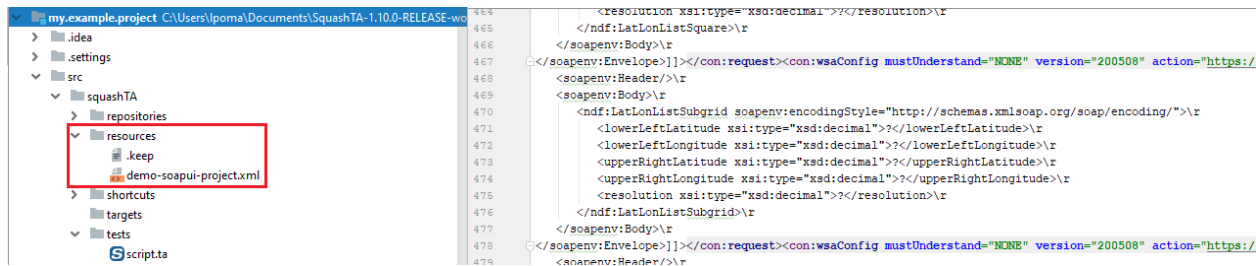
EXECUTE execute WITH __soapui_script{%%rand2}.soapui AS __exec{%%rand3}.result
ASSERT __exec{%%rand3}.result IS success
```

> Input :

- {soapui_script} : path to a SoapUI xml workspace file. It will be converted to a soapui.script resource.

Example :

```
# EXECUTE_SOAPUI path/to/soapui-script.xml
```

SoapUI Project as a .xml file in the resources directory :**SKF script :**

```
TEST :
```

```
# EXECUTE_SOAPUI demo-soapui-project.xml
```

Result output on error in the SKF script :

In this example we misspelled the name of the SoapUI project in the macro.

```
[INFO] Squash TF : testing...
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test script.ta
[ERROR] The execution raised an error in the TEST phase of the TA script 'script.ta' with the message: 'The resource SCOPE_TEST:demo-soapui-projec.xml cannot be found. Please note that the resource name must not contains any space.'
```

Result output on failure of a test case in the SoapUI project :**# EXECUTE_SOAPUI {soapui_script} WITH TEST_SUITE {testsuites}****What ?**

This macro loads the {soapui_script} and executes the given {testsuites}. It then verifies the success of the execution.

Underlying instructions :

```
[INFO] Squash TF : testing...
[INFO] Beginning execution of ecosystem tests
[INFO] Beginning execution of test script.ta
[WARN] No SLF4J binding, falling back to sysout logger for class org.squashtest.ta.plugin.commons.library.java.LoggingFixer
[WARN] No SLF4J binding, falling back to sysout logger for class org.squashtest.ta.plugin.soapui.library.SoapUiProcessExecutor
[ERROR] The execution failed in the TEST phase of the TA script 'script.ta' with the message: 'SoapUI test failed. Following are the test cases that failed:
onListZipCode - NDFDgenByDayLatLonList Test Case' from test suite 'Complex TestSuite' has status : FAILED

[INFO] Exporting results
[INFO] Cleaning resources
[INFO] Squash TF : build complete.
[INFO] All the files from C:\Users\lpoma\AppData\Local\Temp\Squash_TA\20190927_152653_3657691685378349531205 were properly deleted.
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
```

```
LOAD {soapui_script} AS __soapui_script{%%rand1}.file
CONVERT __soapui_script{%%rand1}.file TO script.soapui(structured) AS __soapui_script{
↪%%rand2}.soapui

EXECUTE execute WITH __soapui_script{%%rand2}.soapui USING $(soapui.test.suites:
↪{testsuites};soapui.test.cases:{testcases}) AS __exec{%%rand5}.result
ASSERT __exec{%%rand5}.result IS success
```

> Input :

- {soapui_script} : path to a SoapUI xml workspace file. It will be converted to a soapui.script resource.
- {testsuites} : names of the test suites of the SoapUI workspace to execute. It could be one test suite or a list of comma separated test suites to execute.

Example :

```
# EXECUTE_SOAPUI path/to/soapui-script.xml WITH TEST_SUITE testsuite_name
```

SKF script with 2 test suites :

TEST :

```
# EXECUTE_SOAPUI demo-soapui-project.xml WITH TEST_SUITE SimpleTestSuite, Complex TestSuite
```

EXECUTE_SOAPUI {soapui_script} WITH TEST_SUITE {testsuites} AND TEST_CASE {testcases}

What ?

This macro loads a SoapUI xml workspace and executes the given test case(s). The macro then verifies the success of the execution.

Underlying instructions :


```

LOAD {soapui_script} AS __soapui_script{%%rand1}.file
CONVERT __soapui_script{%%rand1}.file TO script.soapui(structured) AS __soapui_script{
↳%%rand2}.soapui

EXECUTE execute WITH __soapui_script{%%rand2}.soapui USING $(soapui.test.suites:
↳{testsuites};soapui.test.cases:{testcases}) AS __exec{%%rand5}.result
ASSERT __exec{%%rand5}.result IS success

```

> Input :

- {soapui_script} : path to a SoapUI xml workspace file. It will be converted to a soapui.script resource.
- {testsuites} : names of test suites of SoapUI workspace to execute. It could be one test suite or a list of comma separated test suites to execute.
- {testcases} : names of test cases to execute in the test suite. It could be only one test case or a comma separated list of test cases.

Example :

```
# EXECUTE_SOAPUI path/to/soapui-script.xml WITH TEST_SUITE issueServiceTest AND
TEST_CASE retrieveIssue,issueExists
```

SKF script :

TEST :

```
# EXECUTE_SOAPUI demo-soapui-project.xml WITH TEST_SUITE SimpleTestSuite AND TEST_CASE LatLonListZipCode Test Case
```

EXECUTE_SOAPUI_BUNDLE {soapui_bundle} WITH PROJECT {projectpath}**What ?**

This macro executes a SoapUI script embedded in a bundle. The macro then verifies the success of the execution.

Underlying instructions :

```

LOAD {soapui_bundle} AS __soapui_bundle{%%rand1}.file
CONVERT __soapui_bundle{%%rand1}.file TO script.soapui(structured) AS __soapui_bundle{
↳%%rand2}.soapui

EXECUTE execute WITH __soapui_bundle{%%rand2}.soapui USING $(soapui.project.path:
↳{projectpath}) AS __exec{%%rand3}.result
ASSERT __exec{%%rand3}.result IS success

```

> Input :

- {soapui_bundle} : path to the SoapUI bundle to load.

- {projectpath} : path to the SoapUI xml workspace file (relative to the root of the bundle).

Example :

```
# EXECUTE_SOAPUI_BUNDLE soapui WITH PROJECT soapui-integration-tests.xml
```

EXECUTE_SOAPUI_BUNDLE {soapui_bundle} WITH PROJECT {projectpath} AND TEST_SUITE {testsuites}

What ?

This macro executes the given test suites of the loaded SoapUI bundle. The macro then verifies the success of the execution.

Underlying instructions :

```
LOAD {soapui_bundle} AS __soapui_bundle{%%rand1}.file
CONVERT __soapui_bundle{%%rand1}.file TO script.soapui(structured) AS __soapui_bundle{
↳ %%rand2}.soapui

EXECUTE execute WITH __soapui_bundle{%%rand2}.soapui USING $(soapui.project.path:
↳ {projectpath});soapui.test.suites:{testsuites}) AS __exec{%%rand5}.result
ASSERT __exec{%%rand5}.result IS success
```

> Input :

- {soapui_bundle} : path to SoapUI bundle to load.
- {projectpath} : path to SoapUI xml workspace file (relative to the root of the bundle).
- {testcases} : names of testsuites of soapui workspace to execute. It could be one test suite or a list of comma separated test suites to execute.

Example :

```
# EXECUTE_SOAPUI_BUNDLE soapui WITH PROJECT soapui-integration-tests.xml AND TEST_SUITE is-
sucServiceTest
```

EXECUTE_SOAPUI_BUNDLE {soapui_bundle} WITH PROJECT {projectpath} AND TEST_SUITE {testsuites} AND TEST_CASE {testcases}

What ?

This macro loads a SoapUI bundle and executes the given test cases. The macro then verifies the success of the execution.

Underlying instructions :

```
LOAD {soapui_bundle} AS __soapui_bundle{%%rand1}.file
CONVERT __soapui_bundle{%%rand1}.file TO script.soapui(structured) AS __soapui_bundle{
↳%%rand2}.soapui

EXECUTE execute WITH __soapui_bundle{%%rand2}.soapui USING $(soapui.project.path:
↳{projectpath};soapui.test.suites:{testsuites};soapui.test.cases:{testcases}) AS __
↳exec{%%rand5}.result
ASSERT __exec{%%rand5}.result IS success
```

> Input :

- {soapui_bundle} : path to the SoapUI bundle to load.
- {projectpath} : path to SoapUI xml workspace file (relative to the root of the bundle).
- {testsuites} : names of test suites of SoapUI workspace to execute. It could be one test suite or a list of comma separated test suites to execute.
- {testcases} : names of test cases to execute in the test suite. It could be only one test case or a comma separated list of test cases.

Example :

```
# EXECUTE_SOAPUI_BUNDLE soapui WITH PROJECT soapui-integration-tests.xml AND TEST_SUITE is-
sueServiceTest AND TEST_CASE retrieveIssue,issueExists
```

6.11.3 SoapUI Plugin - Advanced Users

SoapUI Plugin - Converters

From 'file' to 'script.soapui'

Category-name : *structured*

What ?

This *structured* script converter will convert `xml` file resources to `script.soapui` resources. The converted resource is then ready for execution through the *'execute'* *'script.soapui'* command.

CONVERT	{xmlSoapUI<Res:file>}	TO	script.soapui	(script)	[USING	\$(soapui.project.path:<soapui.project.path>)] AS {converted<Res:script.soapui>}
---------	-----------------------	----	---------------	----------	--------	--

> Input :

- {xmlSoapui<Res:file>} : The name of the resource to convert (`file` type resource). This resource references a single `xml` workspace file as produced by SoapUI or (since **1.7**) a bundle containing such a file.
- soapui.project.path (since **1.7**) : In case `xmlSoapui` is a bundle, we can indicate here the path to the `xml` workspace file relatively to the bundle's root.

> Output :

- {converted<Res:script.soapui>} : The name of the converted resource (`script.soapui` type resource).

Note :

- If `script.soapui` is a bundle, we must indicate the path of the `xml` workspace file either in the convert or the command instruction.
- If the path is indicated in both, the command instruction prevails.
- If it is not indicated, the test will fail.

Example :

LOAD path/to/soapui_script.xml AS soapui_script.file
CONVERT soapui_script.file TO script.soapui (structured) AS soapui_script.soapui

SoapUI Plugin - Command

'execute' 'script.soapui'

What ?

This command executes the test suite defined by the `script.soapui` resource given as input. It is executed on the (implicit) void target because the SUT target is defined by the SoapUI workspace and cannot be overridden.

EXECUTE execute WITH {soapuiScript<Res:script.soapui>} AS {soapuiResult<Res:result.soapui>} [USING \$(<soapui.test.suites>;<soapui.test.cases>;<soapui.project.path>)]

> Input :

- `{soapuiScript<Res:script.soapui>}` : This resource references a single xml workspace file as produced by SoapUI, or (since **1.7**) a bundle containing such a file (`script.soapui` type resource).
- `soapui.test.suites` : Expected value is the comma separated list of test suite names to execute. If this key is not defined or if it is an empty string, then all test suites are selected.
- `soapui.test.cases` : Expected value is the comma separated list of the test case names to execute in the selected test suites. If this key is not defined or if its value is an empty string, then all test cases are selected.
- `soapui.project.path` (since **1.7**) : In case `script.soapui` is a bundle, we can indicate here the path to the xml workspace file relatively to the bundle's root.

> Output :

- `{soapuiResult<Res:result.soapui>}` : The name of the resource which contains the result of the SoapUI command execution (`soapui.result` type resource).

Note :

- If `script.soapui` is a bundle, we must indicate the path of the xml workspace file either in the convert or the command instruction.
- If the path is indicated in both, the command instruction prevails.
- If it is not indicated, the test will fail.

Example :

```
LOAD path/to/soapui_script.xml AS soapui_script.file
CONVERT soapui_script.file TO script.soapui (structured) AS soapui_script.soapui
EXECUTE execute WITH soapui_script.soapui USING $(soapui.test.suites=suite1,suite2;soapui.test.cases=tc1,tc2)
AS result
```

SoapUI Plugin - Assert**'result.soapui' is 'success'****What ?**

This assertion verifies if a soapUI execution is a success. If the assertion is verified then the test continues. In the other case, the test fails.

```
ASSERT {soapuiResult<Res:result.soapui>} IS success
VERIFY {soapuiResult<Res:result.soapui>} IS success
```

Note : For differences between ASSERT and VERIFY assertion mode see [this page](#).

> Input :

- `{soapuiResult<Res:result.soapui>}` : The name of the resource which contains the result of a soapUI execution command (`result.soapui` type resource).

Example :

```
LOAD path/to/soapui_script.xml AS soapui_script.file
CONVERT soapui_script.file TO script.soapui (structured) AS soapui_script.soapui
EXECUTE execute WITH soapui_script.soapui USING $(soapui.test.suites=suite1,suite2;soapui.test.cases=tc1,tc2)
AS soapuiResult
ASSERT soapuiResult IS success
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

The SoapUI plugin is part of the base package shipped with SKF. It is automatically installed if you choose the default project configuration for your test project. However, as it is packaged as a separate plugin, you can exclude it from the test project (and avoid downloading and installing its dependencies).

This plugin includes all the necessary components to execute test cases from SoapUI workspaces as part of a **Squash TF** test project.

6.12 SSH/SFTP Plugin

6.12.1 SSH/SFTP Plugin - Target

SSH

Category-name : *ssh.target*

What ?

The *ssh.target* represents a ssh server used for the execution of commands. This is mainly used as SUT specification for batch testing.

Configuration : A simple .properties file dropped in the targets directory of your test project. To tag the file as an ssh configuration file, the first line must have the following shebang mark : **#!ssh**. All keys in this file begin with the prefix *squashtest.ta.ssh*.

Available parameters :

- *squashtest.ta.ssh.hostname* (mandatory) : Host to connect to.
- *squashtest.ta.ssh.port* : Port to connect to. This parameter is optional, if it is omitted or empty the default SSH port will be used.
- *squashtest.ta.ssh.username* (mandatory) : Username to use for connection.
- *squashtest.ta.ssh.password* (mandatory) : Password to use for connection.

Example of valid configuration file :

```
#!ssh
squashtest.ta.ssh.hostname=integrationBox
squashtest.ta.ssh.username=tester
squashtest.ta.ssh.password=tester
```

6.12.2 SSH/SFTP Plugin - Resources

Contents:

- *query.shell*
- *result.shell*

query.shell

Category-name : *query.shell*

@See : Since **Squash TA 1.7.0**, this resource moved to *Local Process Plugin*.

result.shell

Category-name : *result.shell*

@See : Since **Squash TA 1.7.0**, this resource moved to *Local Process Plugin*.

6.12.3 SSH/SFTP Plugin - Macros

Contents :

- `# EXECUTE_SSH $({command_content}) ON {target} AS {result}`
- `# EXECUTE_SSH $({command_content}) ON {target} AS {result} WITHIN {timeout} ms`
- `# EXECUTE_SSH_SCRIPT {script} ON {target} AS {result}`
- `# EXECUTE_SSH_SCRIPT {script} ON {target} AS {result} WITHIN {timeout} ms`

EXECUTE_SSH \$({command_content}) ON {target} AS {result}

What ?

This macro will execute an inline command on a SSH server

Underlying instructions :

```
DEFINE $({command_content}) AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell AS __commandLine{%%rand2}
EXECUTE execute WITH __commandLine{%%rand2} ON {target} AS {result}
ASSERT {result} IS success
```

> Input :

- `{command_content}` : It corresponds to the text of the shell command to execute.
- `{target}` : The name (in the context) of the SSH server to use. (`ssh.target type target`).

> Output :

- `{result}` : The name of the resource which references the result of the command.(`result.shell type resource`).

Example :

```
# EXECUTE_SSH $(echo "hello world") ON ssh-server AS result
```

EXECUTE_SSH \$({command_content}) ON {target} AS {result} WITHIN {timeout} ms

What ?

This macro will execute an inline command on a SSH server.

Underlying instructions :

```
DEFINE $({command_content}) AS __command{%%rand1}
CONVERT __command{%%rand1} TO query.shell AS __commandLine{%%rand2}
EXECUTE execute WITH __commandLine{%%rand2} ON {target} USING ${timeout:{timeout}} AS
↪{result}
ASSERT {result} IS success
```

> Input :

- {target} : The name (in the context) of the SSH server to use (ssh.target type target).
- {command_content} : It corresponds to the text of the shell command to execute.
- {timeout} : Maximal time authorized for the command execution (in milliseconds).

> Output :

- {result} : The name of the resource which references the result of the command(result.shell type resource).

Example :

```
# EXECUTE_SSH $(echo "hello world") ON ssh-server AS result WITHIN 15000 ms
```

EXECUTE_SSH_SCRIPT {script} ON {target} AS {result}

What ?

This macro will execute a shell script on a SSH server.

Underlying instructions :

```
LOAD {script} AS __{%r1}.file
CONVERT __{%r1}.file TO file(param.relativeDate) AS __{%r2}.file
CONVERT __{%r2}.file TO query.shell AS __{%r3}.script
EXECUTE execute WITH __{%r3}.script ON {target} AS {result}
ASSERT {result} IS success
```

> Input :

- {target} : The name (in the context) of the SSH server to use (ssh.target type target).
- {script} : It corresponds to the path of the shell script to execute.

> Output :

- {result} : The name of the resource which references the result of the command (result.shell type resource).

Example :

```
# EXECUTE_SSH_SCRIPT shell/shell-script.txt ON ssh-server AS result
```

EXECUTE_SSH_SCRIPT {script} ON {target} AS {result} WITHIN {timeout} ms

What ?

This macro will execute a shell script on a SSH server.

Underlying instructions :

```
LOAD {script} AS __{%r1}.file
CONVERT __{%r1}.file TO file(param.relativeDate) AS __{%r2}.file
CONVERT __{%r2}.file TO query.shell AS __{%r3}.script
EXECUTE execute WITH __{%r3}.script ON {target} USING $(timeout:{timeout}) AS
→{result}
ASSERT {result} IS success
```

> Input :

- {target} : The name (in the context) of the SSH server to use (ssh.target type target).
- {script} : It corresponds to the path of the shell script to execute.
- {timeout} : Maximal time authorized for the command execution (in milliseconds).

> Output :

- {result} : The name of the resource which references the result of the command(result.shell type resource).

Example :

```
# EXECUTE_SSH_SCRIPT shell/shell-script.txt ON ssh-server AS result WITHIN 15000 ms
```

6.12.4 SSH/SFTP Plugin - Advanced Users

SSH/SFTP Plugin - Converter

From file to query.shell

Category-name : *query*

@See : Since **Squash TA 1.7.0**, this converter moved to *Local Process Plugin*.

SSH/SFTP Plugin - Commands

Contents :

- *'execute' 'query.shell' on 'ssh'*
- *'put' 'file' on 'SFTP'*
- *'get' 'file' on 'SFTP'*
- *'delete' 'file' on 'SFTP'*

'execute' 'query.shell' on 'ssh'

What ?

It allows to execute a command line over SSH.

```
EXECUTE execute WITH {query<Res:query.shell>} ON {<Tar:ssh.target>} AS {result<Res:Result.shell>} [ USING $(timeout : <n>) ]
```

> Input :

- {query<Res:query.shell>} : The name of the resource referencing a file which includes one or several shell command lines (query.shell type resource).
- {<Tar:ssh.target>} : The name (in the context) of the SSH server to use (ssh.target type target).

- `<n>` : An integer that represents time in milliseconds. It's the time to wait before the command execution times out. It can be defined via an inline instruction : `$(timeout : ...)`

Note : If the timeout property is not defined here, we use the timeout property of `query.shell` resource (set to 5s by default).

> Output :

- `{result<Res:result.shell>}` : The name of the resource which contains the shell commands result (`result.shell` type resource).

Example :

```
LOAD shell/shell_command.txt AS command.file
CONVERT command.file TO query.shell USING $(timeout:15000) AS commandLine
EXECUTE execute WITH commandLine ON ssh_server AS result
```

'put' 'file' on 'SFTP'

What ?

This command allows to put a file on a SFTP server.

```
EXECUTE put WITH {<Res:file>} ON {<Tar:ssh.target>} AS $() USING $(remotepath : <distantPath> )
```

> Input :

- `{<Res:file>}` : The name of the resource which references the file to put on the SFTP server (`file` type resource).
- `{<Tar:ssh.target>}` : The name (in the context) of the SFTP server to use (`ssh.target` type target).
- `<distantPath>` : It corresponds to the file path on the SFTP server, relatively to the home directory.

Remark : If in `<distantPath>` some directories don't exist on the server, they are then created.

Example :

```
LOAD path/toto.xml AS toto
EXECUTE put WITH toto ON SFTP-server USING $(remotepath : toto.xml) AS $()
```

‘get’ ‘file’ on ‘SFTP’

What ?

This command allows to get a file from a SFTP server.

```
EXECUTE get WITH $() ON {<Tar:ssh.target>} AS {result<Res:file>} USING $(remotepath : <distantPath> )
```

> Input :

- {<Tar:ssh.target>} : The name (in the context) of the SFTP server to use (ssh.target type target).
- <distantPath> : It corresponds to the file path on the SFTP server, relatively to the home directory of the file you want to get.

> Output :

- {result<Res:file>} : The name of the resource which references the file you get from the SFTP server (file type resource).

Example :

```
EXECUTE get WITH $() ON SFTP-server USING $(remotepath :sample.zip) AS zip
```

‘delete’ ‘file’ on ‘SFTP’

What ?

This command allows to delete a file on a SFTP server.

```
EXECUTE delete WITH $() ON {<Tar:ssh.target>} AS $() USING $(remotepath : <distantPath> [,failIfDoesNotExist : false])
```

> Input :

- {<Tar:ssh.target>} : The name (in the context) of the SFTP server to use (ssh.target type target).
- <distantPath> : It corresponds to the file path on the SFTP server, relatively to the home directory of the file you want to delete.
- 'failIfDoesNotExist : false' : It allows to specify to SKF that the test must not fail if the resource we're trying to delete doesn't exist.

> Output :

- {result<Res:file>} : The name of the resource which references the file you get from the SFTP server (file type resource).

Remarks :

- `<distantPath>` can indicate a file OR a directory. To represent a directory, the path should end with the character `'/'`.
- The deletion of a directory is recursive : deletion of all sub-directories and files.

```
EXECUTE delete WITH $() ON SFTP-server USING $( remotepath : path/to/myfile.txt, failIfDoesNotExist: false)
AS $()
```

SSH/SFTP Plugin - Asserts

Contents :

- `'result.shell'` is `'success'`
- `'result.shell'` is `'failure'` with `{expected return code}`
- `'result.shell'` does `'contain'` `{regex}`
- `'result.shell'` does `'not.contain'` `{regex}`

'result.shell' is 'success'

@See : Since **Squash TA 1.7.0**, this resource moved to *Local Process Plugin*.

'result.shell' is 'failure' with {expected return code}

@See : Since **Squash TA 1.7.0**, this resource moved to *Local Process Plugin*.

'result.shell' does 'contain' {regex}

@See : Since **Squash TA 1.7.0**, this resource moved to *Local Process Plugin*.

'result.shell' does 'not.contain' {regex}

@See : Since **Squash TA 1.7.0**, this resource moved to *Local Process Plugin*.

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin provides all the elements needed to interact with an SSH server.

6.13 XML Functions Plugin

6.13.1 XML Functions Plugin - Resources

xslt

Category-name : *xslt*

What ?

xslt is a resource representing a styling document allowing xml transformations.

6.13.2 XML Functions Plugin - Macros

XML Functions Plugin - Macros - Create File

Contents :

- *# CREATE FILE {output} FROM {input} BY APPLYIN XSLT {stylesheet}*
- *# CREATE FILE {output} FROM {input} BY APPLYIN XSLT {stylesheet} USING {xslt_config}*
- *# CREATE FILE {output} FROM XML RESOURCE {input} BY APPLYIN XSLT {stylesheet}*
- *# CREATE FILE {output} FROM XML RESOURCE {input} BY APPLYIN XSLT {stylesheet} USING {xslt_config}*

CREATE FILE {output} FROM {input} BY APPLYIN XSLT {stylesheet}

What ?

This macro applies a stylesheet to a xml file in order to create a file resource and logs the result in the console at DEBUG level.

Underlying instructions :

```
LOAD {input} AS __input{%%r1}.xml.file
CONVERT __input{%%r1}.xml.file TO xml(structured) AS __input{%%r1}.xml

LOAD {stylesheet} AS __stylesheet{%%r1}.xslt.file
CONVERT __stylesheet{%%r1}.xslt.file TO xml(structured) AS __stylesheet{%%r1}.xslt.xml
CONVERT __stylesheet{%%r1}.xslt.xml TO xslt AS __stylesheet{%%r1}.xslt

CONVERT __input{%%r1}.xml TO file(xslt) USING __stylesheet{%%r1}.xslt AS {output}
EXECUTE log WITH {output} USING $(logLevel: DEBUG,multiline: yes) AS $()
```

> Input :

- {input} : The name of the file to convert (xml type file).
- {stylesheet} : The name of the stylesheet file (xslt type file).

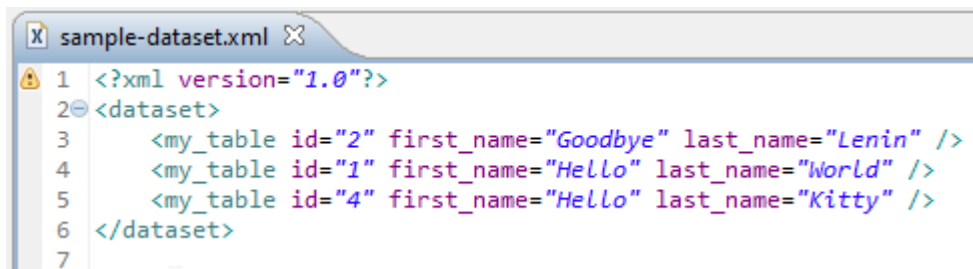
> Output :

- {output} : The name of the converted resource (file type resource).

Example :

```
# CREATE FILE sample-dataset-resource.xml FROM sample-dataset.xml BY APPLYIN XSLT ta-
ble1n2sorted.xslt
```

First file to process :



```
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7
```

Second file to process :


```

1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="."/></xsl:copy-of>
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="."/></xsl:copy-of>
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17

```

The folder containing the resources to process :

```

v src
  v squashTA
    > repositories
    v resources
      sample-dataset.xml
      table1n2sorted.xslt
    > shortcuts
    > targets
    v tests
      create-file.ta

```

SKF script :

```

1 TEST :
2
3 # CREATE FILE sample-dataset-resource.xml FROM sample-dataset.xml BY APPLYIN XSLT table1n2sorted.xslt
4

```

Console output in DEBUG mode :

```

> EXECUTE log WITH sample-dataset-resource.xml USING {__temp10}} AS {__temp11}}
[DEBUG] Creating instance of bean 'writeLogCommand'
[DEBUG] Finished creating instance of bean 'writeLogCommand'
[DEBUG] <?xml version="1.0" encoding="UTF-8"?><dataset><my_table id="1" first_name="Hello" last_name="World"/><my_table id="2" first_name="Goodbye" last_name="Lenin"/></dataset>

```

CREATE FILE {output} FROM {input} BY APPLYIN XSLT {stylesheet} USING {xslt_config}

What ?

This macro applies a stylesheet and a config resource to a xml file in order to create a file resource and logs the result in the console at DEBUG level. Additional configurations can be done with a config file.

Underlying instructions :

```
LOAD {input} AS __input{%%r1}.xml.file
CONVERT __input{%%r1}.xml.file TO xml(structured) AS __input{%%r1}.xml

LOAD {stylesheet} AS __stylesheet{%%r1}.xslt.file
CONVERT __stylesheet{%%r1}.xslt.file TO xml(structured) AS __stylesheet{%%r1}.xslt.xml
CONVERT __stylesheet{%%r1}.xslt.xml TO xslt AS __stylesheet{%%r1}.xslt

CONVERT __input{%%r1}.xml TO file(xslt) USING __stylesheet{%%r1}.xslt,{xslt_config}
↪ AS {output}
EXECUTE log WITH {output} USING $(logLevel: DEBUG,multiline: yes) AS $()
```

> Input :

- {input} : The name of the file to convert (xml type file).
- {stylesheet} : The name of the stylesheet file (xslt type file).
- {xslt_config} : The name of the loaded configuration resource (file type resource, from a properties type file). It can be used to normalize the output (normalize = true).

> Output :

- {output} : The name of the converted resource (file type resource).

Example :

```
# CREATE FILE sample-dataset-resource.xml FROM sample-dataset.xml BY APPLYIN XSLT ta-
ble1n2sorted.xslt USING config-resource.file
```

First file to process :

```
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7
```

Second file to process :

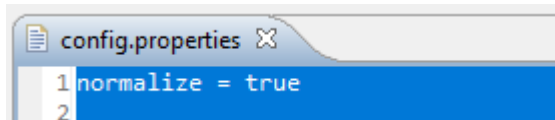


```

1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="."/></xsl:copy-of>
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="."/></xsl:copy-of>
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17

```

Third file to process :

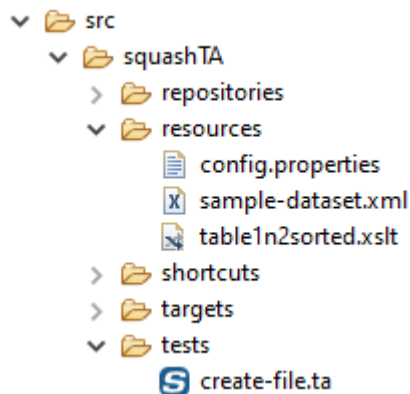


```

1 normalize = true
2

```

The folder containing the resources to process :

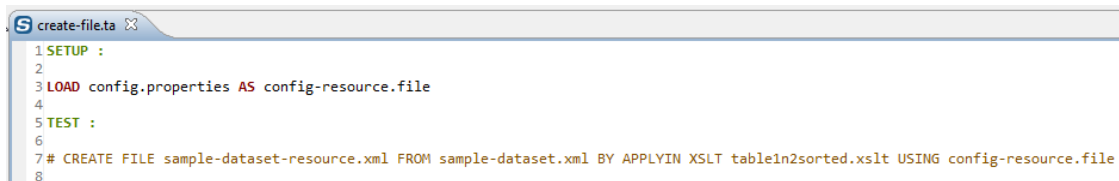


```

v src
  v squashTA
    > repositories
    v resources
      config.properties
      sample-dataset.xml
      table1n2sorted.xslt
    > shortcuts
    > targets
    v tests
      create-file.ta

```

SKF script :



```

1 SETUP :
2
3 LOAD config.properties AS config-resource.file
4
5 TEST :
6
7 # CREATE FILE sample-dataset-resource.xml FROM sample-dataset.xml BY APPLYIN XSLT table1n2sorted.xslt USING config-resource.file
8

```

Console output in DEBUG mode :

```
> EXECUTE log WITH sample-dataset-resource.xml USING {__tempI0}} AS {__tempI1}}
[DEBUG] Creating instance of bean 'writeLogCommand'
[DEBUG] Finished creating instance of bean 'writeLogCommand'
[DEBUG] <?xml version="1.0" encoding="UTF-8"?>
<dataset>
<my_table id="1" first_name="Hello" last_name="World"/>
<my_table id="2" first_name="Goodbye" last_name="Lenin"/>
</dataset>
```

CREATE FILE {output} FROM XML RESOURCE {input} BY APPLYIN XSLT {stylesheet}

What ?

This macro applies a stylesheet to a loaded xml resource in order to create a file resource and logs the result in the console at DEBUG level.

Underlying instructions :

```
LOAD {stylesheet} AS __stylesheet{%%r1}.xslt.file
CONVERT __stylesheet{%%r1}.xslt.file TO xml(structured) AS __stylesheet{%%r1}.xslt.xml
CONVERT __stylesheet{%%r1}.xslt.xml TO xslt AS __stylesheet{%%r1}.xslt

CONVERT {input} TO file(xslt) USING __stylesheet{%%r1}.xslt AS {output}
EXECUTE log WITH {output} USING $(logLevel: DEBUG,multiline: yes) AS $()
```

> Input :

- {input} : The name of the loaded resource to convert (xml type resource).
- {stylesheet} : The name of the stylesheet file (xslt type file).

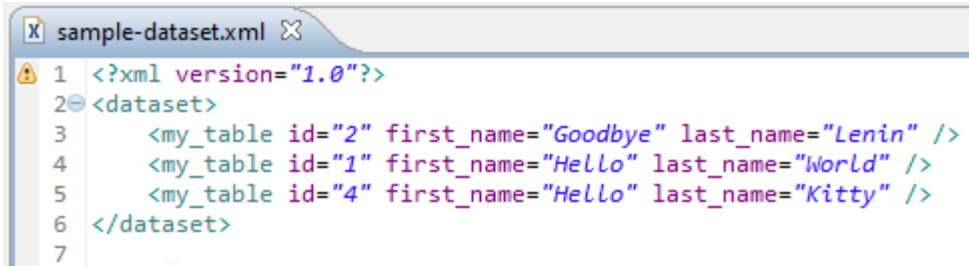
> Output :

- {output} : The name of the converted resource (file type resource).

Example :

```
# CREATE FILE sample-dataset-resource-output.xml FROM XML RESOURCE sample-dataset-
resource.xml BY APPLYIN XSLT table1n2sorted.xslt
```

First file to process :

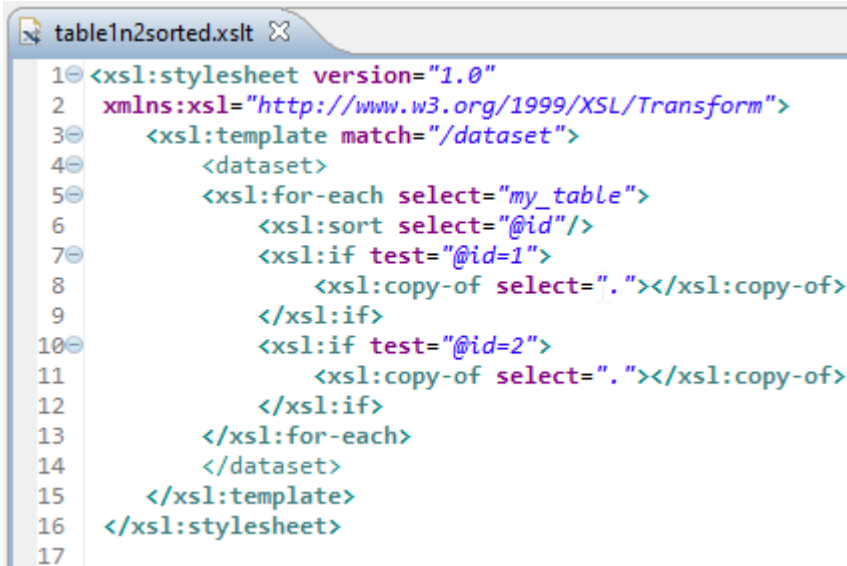


```

1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7

```

Second file to process :

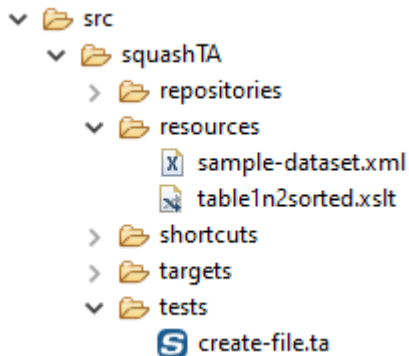


```

1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="."/></xsl:copy-of>
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="."/></xsl:copy-of>
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17

```

The folder containing the resources to process :

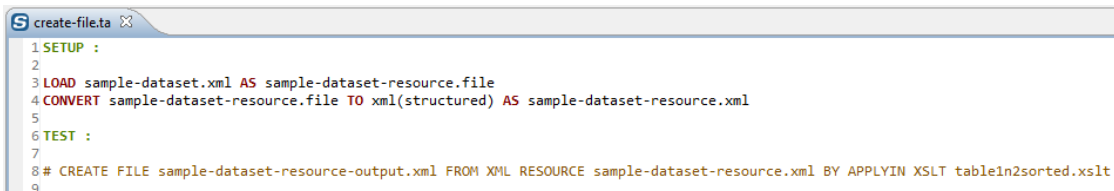


```

src
├── squashTA
│   ├── repositories
│   ├── resources
│   │   ├── sample-dataset.xml
│   │   └── table1n2sorted.xslt
│   ├── shortcuts
│   ├── targets
│   └── tests
│       └── create-file.ta

```

SKF script :



```

1 SETUP :
2
3 LOAD sample-dataset.xml AS sample-dataset-resource.file
4 CONVERT sample-dataset-resource.file TO xml(structured) AS sample-dataset-resource.xml
5
6 TEST :
7
8 # CREATE FILE sample-dataset-resource-output.xml FROM XML RESOURCE sample-dataset-resource.xml BY APPLYIN XSLT table1n2sorted.xslt
9

```

Console output in DEBUG mode :

```
> EXECUTE log WITH sample-dataset-resource-output.xml USING ((__tempI0)) AS ((__tempI1))
[DEBUG] Creating instance of bean 'writeLogCommand'
[DEBUG] Finished creating instance of bean 'writeLogCommand'
[DEBUG] <?xml version="1.0" encoding="UTF-8"?><dataset><my_table id="1" first_name="Hello" last_name="World"/><my_table id="2" first_name="Goodbye" last_name="Lenin"/></dataset>
```

CREATE FILE {output} FROM XML RESOURCE {input} BY APPLYIN XSLT {stylesheet} USING {xslt_config}

What ?

This macro applies a stylesheet and a config resource to a loaded xml resource in order to create a file resource and logs the result in the console at DEBUG level. Additional configurations can be done with a config file.

Underlying instructions :

```
LOAD {stylesheet} AS __stylesheet{%%r1}.xslt.file
CONVERT __stylesheet{%%r1}.xslt.file TO xml(structured) AS __stylesheet{%%r1}.xslt.xml
CONVERT __stylesheet{%%r1}.xslt.xml TO xslt AS __stylesheet{%%r1}.xslt

CONVERT {input} TO file(xslt) USING __stylesheet{%%r1}.xslt,{xslt_config} AS {output}
EXECUTE log WITH {output} USING $(logLevel: DEBUG,multiline: yes) AS $()
```

> Input :

- {input} : The name of the loaded resource to convert (xml type resource).
- {stylesheet} : The name of the stylesheet resource (xslt type file).
- {xslt_config} : The name of the loaded configuration resource (file type resource, from a properties type file). It can be used to normalize the output (normalize = true).

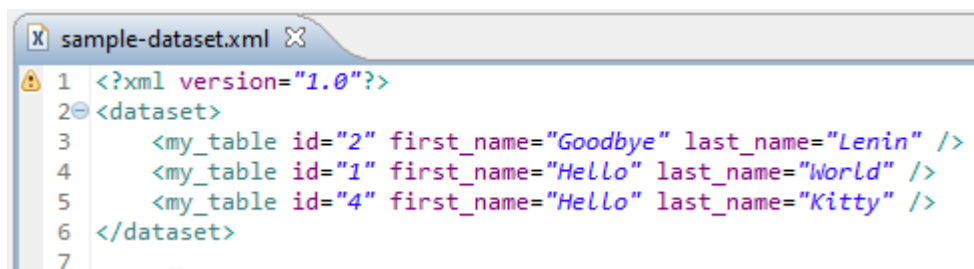
> Output :

- {output} : The name of the converted resource (file type resource).

Example :

```
# CREATE FILE sample-dataset-resource-output.xml FROM XML RESOURCE sample-dataset-resource.xml BY APPLYIN XSLT table1n2sorted.xslt USING config-resource.file
```

First file to process :



```
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7
```

Second file to process :

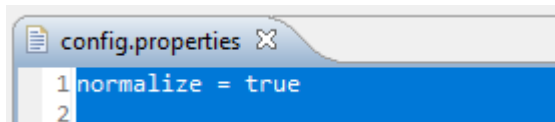


```

1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="."/></xsl:copy-of>
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="."/></xsl:copy-of>
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17

```

Third file to process :

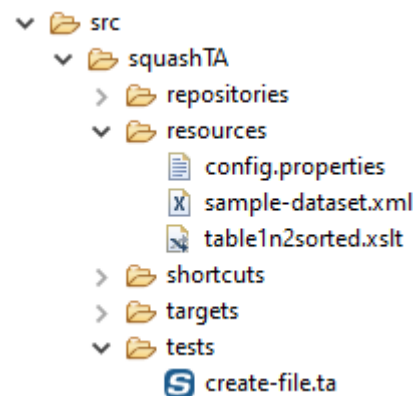


```

1 normalize = true
2

```

The folder containing the resources to process :

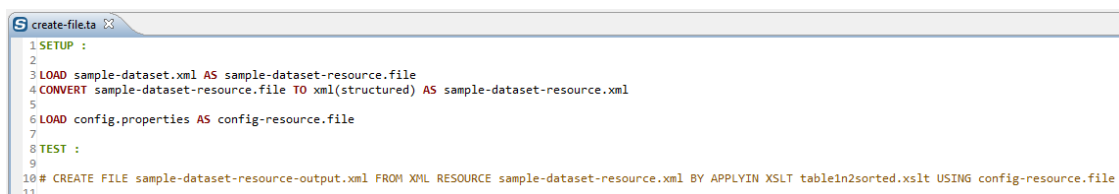


```

src
├── squashTA
│   ├── repositories
│   ├── resources
│   │   ├── config.properties
│   │   ├── sample-dataset.xml
│   │   └── table1n2sorted.xslt
│   ├── shortcuts
│   ├── targets
│   └── tests
└── create-file.ta

```

SKF script :



```

1 SETUP :
2
3 LOAD sample-dataset.xml AS sample-dataset-resource.file
4 CONVERT sample-dataset-resource.file TO xml(structured) AS sample-dataset-resource.xml
5
6 LOAD config.properties AS config-resource.file
7
8 TEST :
9
10 # CREATE FILE sample-dataset-resource-output.xml FROM XML RESOURCE sample-dataset-resource.xml BY APPLYIN XSLT table1n2sorted.xslt USING config-resource.file
11

```

Console output in DEBUG mode :

```
> EXECUTE log WITH sample-dataset-resource-output.xml USING {__tempI0} AS {__tempI1}
[DEBUG] Creating instance of bean 'writeLogCommand'
[DEBUG] Finished creating instance of bean 'writeLogCommand'
[DEBUG] <?xml version="1.0" encoding="UTF-8"?>
<dataset>
<my_table id="1" first_name="Hello" last_name="World"/>
<my_table id="2" first_name="Goodbye" last_name="Lenin"/>
</dataset>
```

XML Functions Plugin - Macros - Check XML

Contents :

- *# CHECK IF XML FILE {actual} FILTERED BY {xslt_filter} EQUALS {expected}*
- *# CHECK IF XML {actual} FILTERED BY {xslt_filter} EQUALS {expected}*
- *# CHECK IF XML {actual} FILTERED BY {xslt_filter} EQUALS {expected} USING {config}*

CHECK IF XML FILE {actual} FILTERED BY {xslt_filter} EQUALS {expected}

What ?

This macro apply a xslt filter to an actual and an expected xml file and then checks if the resulting resources match.

Underlying instructions :

```
LOAD {xslt_filter} AS __xslt_filter{%%r1}.file
CONVERT __xslt_filter{%%r1}.file TO xml(structured) AS __xslt_filter{%%r1}.xml
CONVERT __xslt_filter{%%r1}.xml TO xslt AS __xslt_filter{%%r1}.xslt

LOAD {actual} AS __actual{%%r1}.file
CONVERT __actual{%%r1}.file TO xml(structured) AS __actual{%%r1}.xml
CONVERT __actual{%%r1}.xml TO xml(xslt) USING __xslt_filter{%%r1}.xslt,
↪$(normalize:true) AS __filtered_actual{%%r1}.xml

LOAD {expected} AS __expected{%%r1}.file
CONVERT __expected{%%r1}.file TO xml(structured) AS __expected{%%r1}.xml
CONVERT __expected{%%r1}.xml TO xml(xslt) USING __xslt_filter{%%r1}.xslt,
↪$(normalize:true) AS __filtered_expected{%%r1}.xml

ASSERT __filtered_expected{%%r1}.xml IS similaire WITH __filtered_actual{%%r1}.xml_
↪USING $(comparateur:xmlunit)
```

> Input :

- {xslt_filter} : The name of the filter to apply (xslt type file).

- {actual} : The name of the file to compare (xml type file).
- {expected} : The name of the file to be compared to (xml type file).

Example :

```
# CHECK IF XML FILE sample-dataset-1.xml FILTERED BY table1n2sorted.xslt EQUALS sample-dataset-2.xml
```

First file to process :

```
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7
```

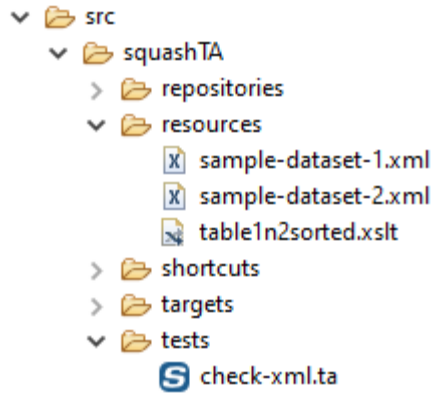
Second file to process :

```
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="1" first_name="Hello" last_name="World" />
4   <my_table id="4" first_name="Hello" last_name="Kitty" />
5   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
6 </dataset>
7
```

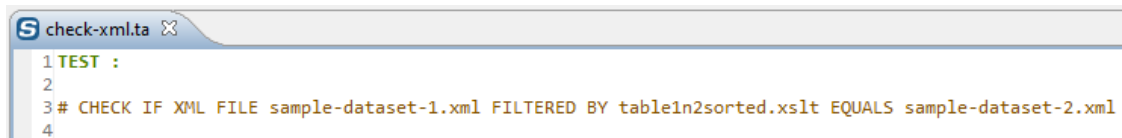
Third file to process :

```
1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="." />
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="." />
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17
```

The folder containing the resources to process :



SKF script :



CHECK IF XML {actual} FILTERED BY {xslt_filter} EQUALS {expected}

What ?

This macro apply a xslt filter to an actual and an expected loaded xml resources and then checks if the resulting resources match.

Underlying instructions :

```
LOAD {xslt_filter} AS __xslt_filter{%%r1}.file
CONVERT __xslt_filter{%%r1}.file TO xml(structured) AS __xslt_filter{%%r1}.xml
CONVERT __xslt_filter{%%r1}.xml TO xslt AS __xslt_filter{%%r1}.xslt

CONVERT {expected} TO xml(xslt) USING __xslt_filter{%%r1}.xslt,$(normalize:true) AS __
↳filtered_expected{%%r1}.xml

CONVERT {actual} TO xml(xslt) USING __xslt_filter{%%r1}.xslt,$(normalize:true) AS __
↳filtered_actual{%%r1}.xml

ASSERT __filtered_actual{%%r1}.xml IS similaire WITH __filtered_expected{%%r1}.xml_
↳USING $(comparateur:xmlunit)
```

> Input :

- {xslt_filter} : The name of the filter to apply (xslt type file).
- {actual} : The name of the loaded resource to compare (xml type resource).
- {expected} : The name of the loaded resource to be compared to (xml type resource).

Example :

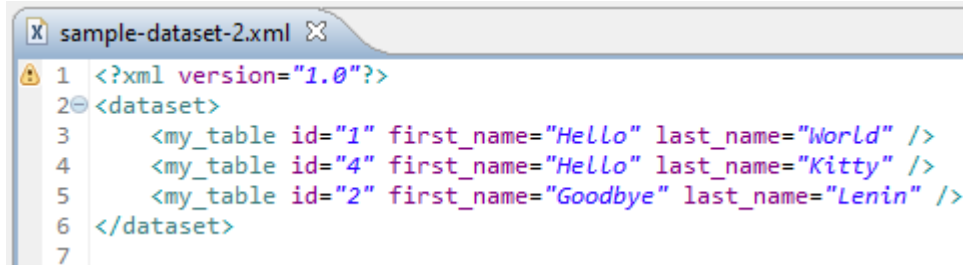
```
# CHECK IF XML sample-dataset-1-resource.xml FILTERED BY table1n2sorted.xslt EQUALS
sample-dataset-2-resource.xml
```

First file to process :



```
sample-dataset-1.xml
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7
```

Second file to process :



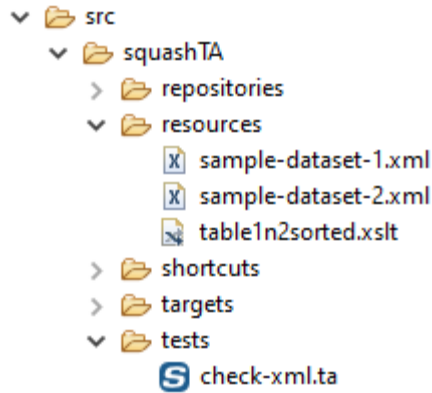
```
sample-dataset-2.xml
1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="1" first_name="Hello" last_name="World" />
4   <my_table id="4" first_name="Hello" last_name="Kitty" />
5   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
6 </dataset>
7
```

Third file to process :



```
table1n2sorted.xslt
1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="." /></xsl:copy-of>
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="." /></xsl:copy-of>
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17
```

The folder containing the resources to process :



SKF script :

```
check-xml.ta
1 SETUP :
2
3 LOAD sample-dataset-1.xml AS sample-dataset-1-resource.file
4 CONVERT sample-dataset-1-resource.file TO xml(structured) AS sample-dataset-1-resource.xml
5
6 LOAD sample-dataset-2.xml AS sample-dataset-2-resource.file
7 CONVERT sample-dataset-2-resource.file TO xml(structured) AS sample-dataset-2-resource.xml
8
9 TEST :
10
11 # CHECK IF XML sample-dataset-1-resource.xml FILTERED BY table1n2sorted.xslt EQUALS sample-dataset-2-resource.xml
12
```

CHECK IF XML {actual} FILTERED BY {xslt_filter} EQUALS {expected} USING {config}

What ?

This macro apply a xslt filter to an actual and an expected loaded xml resources and then checks if the resulting resources match. Additional configurations can be done with a config file.

Underlying instructions :

```
LOAD {xslt_filter} AS __xslt_filter{%%r1}.file
CONVERT __xslt_filter{%%r1}.file TO xml(structured) AS __xslt_filter{%%r1}.xml
CONVERT __xslt_filter{%%r1}.xml TO xslt AS __xslt_filter{%%r1}.xslt

CONVERT {expected} TO xml(xslt) USING __xslt_filter{%%r1}.xslt,$(normalize:true) AS __
↪filtered_expected{%%r1}.xml

CONVERT {actual} TO xml(xslt) USING __xslt_filter{%%r1}.xslt,$(normalize:true) AS __
↪filtered_actual{%%r1}.xml

LOAD {config} AS config.file
CONVERT config.file TO properties(structured) AS config.properties
```

(continues on next page)

(continued from previous page)

```

DEFINE $(comparateur=xmlunit) AS default_comparator
CONVERT default_comparator TO properties(structured) AS default_comparator.properties
DEFINE $(comparateur:${comparateur}) AS comparateur.pattern
CONVERT comparateur.pattern TO file(param) AS comparateur.conf.in USING config.
↳properties
CONVERT comparateur.conf.in TO file(param) AS comparateur.conf USING default_
↳comparator.properties

ASSERT __filtered_actual{%%r1}.xml IS similaire WITH __filtered_expected{%%r1}.xml_
↳USING comparateur.conf,config.properties

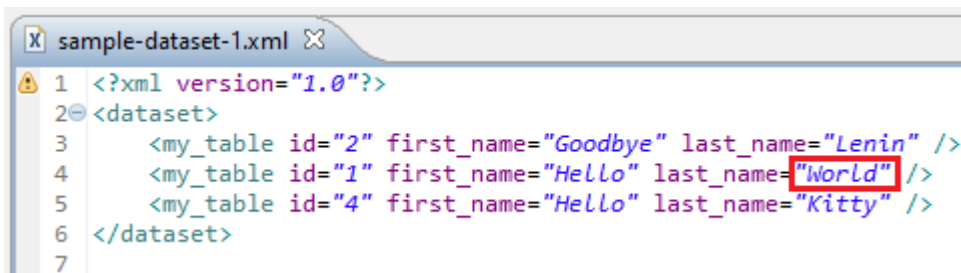
```

> Input :

- {xslt_filter} : The name of the filter to apply (xslt type resource).
- {actual} : The name of the loaded resource to compare (xml type resource).
- {expected} : The name of the loaded resource to be compared to (xml type resource).
- {config} : The name of the configuration file. It can be used to change the default comparison engine from xmlunit to jxb, through a 'comparateur = jaxb' entry, or to give a name to the actual and expected resource (actualName = <put name here>, expectedName = <put name here>).

Example :


```
# CHECK IF XML sample-dataset-1-resource.xml FILTERED BY table1n2sorted.xslt EQUALS
sample-dataset-2-resource.xml USING config.properties
```

First file to process :


```

1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
4   <my_table id="1" first_name="Hello" last_name="World" />
5   <my_table id="4" first_name="Hello" last_name="Kitty" />
6 </dataset>
7

```

Second file to process :


```

1 <?xml version="1.0"?>
2 <dataset>
3   <my_table id="1" first_name="Hello" last_name="Universe" />
4   <my_table id="4" first_name="Hello" last_name="Kitty" />
5   <my_table id="2" first_name="Goodbye" last_name="Lenin" />
6 </dataset>
7

```

Third file to process :

```

1 <xsl:stylesheet version="1.0"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:template match="/dataset">
4     <dataset>
5       <xsl:for-each select="my_table">
6         <xsl:sort select="@id"/>
7         <xsl:if test="@id=1">
8           <xsl:copy-of select="."></xsl:copy-of>
9         </xsl:if>
10        <xsl:if test="@id=2">
11          <xsl:copy-of select="."></xsl:copy-of>
12        </xsl:if>
13      </xsl:for-each>
14    </dataset>
15  </xsl:template>
16 </xsl:stylesheet>
17

```

Fourth file to process :

```

1 actualName = The sample dataset with the last_name "World"
2 expectedName = The sample dataset with the last_name "Universe"
3

```

The folder containing the resources to process :

```

v src
  v squashTA
    > repositories
    v resources
      config.properties
      sample-dataset-1.xml
      sample-dataset-2.xml
      table1n2sorted.xslt
    > shortcuts
    > targets
    v tests
      S check-xml.ta

```

SKF script :

```

S check-xml.ta
1 SETUP :
2
3 LOAD sample-dataset-1.xml AS sample-dataset-1-resource.file
4 CONVERT sample-dataset-1-resource.file TO xml(structured) AS sample-dataset-1-resource.xml
5
6 LOAD sample-dataset-2.xml AS sample-dataset-2-resource.file
7 CONVERT sample-dataset-2-resource.file TO xml(structured) AS sample-dataset-2-resource.xml
8
9 TEST :
10
11# CHECK IF XML sample-dataset-1-resource.xml FILTERED BY table1n2sorted.xslt EQUALS sample-dataset-2-resource.xml USING config.properties
12

```

Console output in DEBUG mode :

```
[ERROR] The execution failed in the TEST phase of the TA script 'check-xml.ta' with the message: 'Les contenus des fichiers sont différents'
EXPECTED: The sample dataset with the last_name "Universe"
ACTUAL: The sample dataset with the last_name "World"
COMPARATEUR: xmlunit
différence :
Expected attribute value 'Universe' but was 'World' - comparing <my_table last_name="Universe"...> at /dataset[1]/my_table[1]/@last_name to <my_table last_name="World"...> at /dataset[1]/my_table[1]/@last_name
```

6.13.3 XML Functions Plugin - Advanced Users

XML Functions Plugin - Converters

Contents:

- *From XML ...*
 - ... to XSLT
 - ... to File
 - ... to XML

From XML ...

... to XSLT

Category-Name : *structured.xslt*

What ?

This *structured.xslt* converter will convert a `xml` type resource to a `xslt` type resource.

```
CONVERT {resourceToConvert<Res:xml>} TO xslt (structured.xslt) AS {converted<Res:xslt>}
```

> Input :

- `resourceToConvert<Res:xml>` : The name of the resource to convert (`xml` type resource).

> Output :

- `converted<Res:xslt>` : The name of the converted resource (`xslt` type resource).

Example :

```
LOAD stylesheet.xslt AS stylesheet-resource.file
CONVERT stylesheet-resource.file TO xml (structured) AS stylesheet-resource.xml
CONVERT stylesheet-resource.xml TO xslt (structured.xslt) AS stylesheet-resource.xslt
```

... to File

Category-Name : *xslt*

What ?

This *xslt* converter will convert a `xml` type resource to a `file` type resource. A stylesheet can be applied to the `xml` resource.

```
CONVERT {resourceToConvert<Res:xml>} TO file (xslt) [USING {config<Res:xslt>}] AS {converted<Res:file>}
```

> Input :

- `resourceToConvert<Res:xml>` : The name of the resource to convert (`xml` type resource).
- `config<Res:xslt>` : The name of the configuration resource (`xslt` type resource).

> Output :

- `converted<Res:file>` : The name of the converted resource (`file` type resource).

Example :

```
LOAD sample.xml AS sample-resource.file
CONVERT sample-resource.file TO xml (structured) AS sample-resource.xml
LOAD stylesheet.xslt AS stylesheet-resource.file
CONVERT stylesheet-resource.file TO xml (structured) AS stylesheet-resource.xml
CONVERT stylesheet-resource.xml TO xslt (structured.xslt) AS stylesheet-resource.xslt
CONVERT sample-resource.xml TO file (xslt) USING stylesheet-resource.xslt AS final-sample-resource.file
```

... to XML

Category-Name : *xslt*

What ?

This *xslt* converter will convert a `xml` type resource to another `xml` type resource. A stylesheet can be applied to the `xml` resource.

```
CONVERT {resourceToConvert<Res:xml>} TO xml (xslt) [USING {config<Res:xslt>}] AS {converted<Res:xml>}
```

> Input :

- `resourceToConvert<Res:xml>` : The name of the resource to convert (xml type resource).
- `config<Res:xslt>` : The name of the configuration resource (xslt type resource).

> Output :

- `converted<Res:xml>` : The name of the converted resource (xml type resource).

Example :

```
LOAD sample.xml AS sample-resource.file
CONVERT sample-resource.file TO xml (structured) AS sample-resource.xml
LOAD stylesheet.xslt AS stylesheet-resource.file
CONVERT stylesheet-resource.file TO xml (structured) AS stylesheet-resource.xml
CONVERT stylesheet-resource.xml TO xslt (structured.xslt) AS stylesheet-resource.xslt
CONVERT sample-resource.xml TO xml (xslt) USING stylesheet-resource.xslt AS final-sample-resource.xml
```

This section will give you further details about the engine components (converters, commands or asserts) of the SKF which are used by the macros of this plugin.

This plugin provides the possibility to apply XSL transformations to XML type resources.

7.1 Automated Tests Rudiments

7.1.1 Good Practices

Contents :

- *Simplicity*
- *Functionnal Unicity*
- *Independence*
- *Static Data Test*
- *Dynamic Data Test*
- *Maintenance*

Automated tests progress is nearly similar to the one of manual tests. Nevertheless, automation has specificities conditioning the way which automated tests are designed and realized.

Simplicity

An automated test must be as simple as possible : easy to write, easy to read and easy to understand. The writing of complex tests increases the risk of error and so of false positive (test fail although the tested application isn't involved).

Functional Unicity

In the case of a manual acceptance test, it's common to follow long test procedures who are verifying a lot of functionalities of the tested system in one scenario.

It's necessary due to the specific constraints of manual acceptance tests : in order to test the application, the tester has to set-up pre-requisites and verify post-conditions. Several system functions are tested like this with only one test.

Automated tests allow getting rid of those constraints because the setting-up of pre-requisites and verifications can be done without the tested application.

Each automated test can test only one function of the SUT (System Under Test).

This method has many advantages :

- Tests are shorter and easier.
 - The qualification of a function doesn't depend on the good functioning of another function.
 - During a campaign test, it's possible to test only parts of function of the SUT.
-

Independence

The two steps of setting-up the pre-requisites and cleaning-up the environment must assure that test cases are strictly independant from each other.

The execution of a test case should never depend of the previous test case's result.

This rule is essential for those reasons :

- **Campaign layout** : for each campaign, it must be possible to choose which tests and in which sequence they must be executed.
 - **Results interpretation** : when tests depend on each other, it's more complicated to interpret causes of failure. The failure of one test can lead to the failure of the next test. It's impossible to know if they failed because of the first test's failure or because of real issues.
-

Static Data Test

An automated test must be able to be replayed as many times as necessary and obtain each time the same result.

To make it possible, the simplest solution is to use identic data from one execution to another. This is particularly true for non-regression tests which are valid only if they are executed in strictly identical conditions.

This is possible thanks to the setting-up and cleaning-up steps of the environnement.

Dynamic Data Test

There are 2 exceptions to the previous rule. Some data tests can't be determined a priori because they depend on the context in which the test case is executed. Among those data, there are dates and data generated by the application.

1. DATES

All dataset containing dates is subject to expiration. For example, a contract which was active when tests were realized can expire after a certain period of time. This can lead to the failure of the tests who are using this data set.

To handle this problem, 2 strategies are possible :

- The first one is to upgrade frequently test cases. This solution requires to set-up a follow-up procedure of datasets maintenance which could be expensive.
- It's preferable to set-up a mechanism which allows to define the dates at the moment of the test execution, relatively to the day date (for example, the first Monday of the month, the first open-day of the year, the previous month. . .).

2. DATA GENERATED BY THE APPLICATION

Some data generated by the tested application can't be determined a priori, for example, identifiers or timestamps generated at the execution. Sometimes data in output of a test case must be used as input of the next test case. When it happens, it's necessary to store those data in order to use them later.

Maintenance

One of the main brakes to automation tests stands in the need of maintaining them. That's why automated tests concern stable functions of the tested system which are little set to expand.

Despite those precautions, features of the SUT are going to need maintenance. So we need to anticipate the features during the realization of the tests in order to minimize the maintenance charge.

1. DATA TEST CENTRALIZATION

Sometimes, because of the evolution of the data model for example, a test case must be revalued.

To minimize the maintenance charge, the data of a test must be centralized. Concretely, it means that the data of a test are replaced by parameters whose values are saved in one or several parameters file(s).

In the case of a test case reevaluation, only these parameters files are modified and not the totality of the test cases.

2. COMMON PARTS POOLING

The common steps of several test cases must be shared. So if a modification of the SUT affects a common step to several cases, corrections must be made at only one place. This implies :

- To create shared modules from which test cases will be built.
- To configure data of the shared parts in order to valorize them differently according to test cases.

7.1.2 Proceeding a Test Case / a Test Campagne

Contents :

- *Pre-conditions*
- *Test Steps*
- *Post-conditions Checking*
- *Cleaning Up*
- *Results Storage*
- *Test Campaign*

Pre-conditions

Just as manual tests, automated tests generally begin with a step of setting up pre-requisites before execution of test steps. Nevertheless the way to do it is slightly different between manual tests and automated tests.

In the case of a manual test, the acceptance tester often needs to use the tested application to set-up the pre-requisites of the test. In the example of a Man/Machine Interface permitting to create and consult users accounts : before being able to test the consultation function, we need to create a user account. This method has a major inconvenient : the qualification of a function depends directly of the good functioning of another function.

In the previous example, if the creation function has a blocking issue, consultation function can't be tested.

Unlike manual tests, automated tests allow to set-up pre-requisites of the test case without going through the tested application. In the previous example, it's possible to create the account directly in the database before consulting it with the application. So, the consultation function can be tested, even if the creation function doesn't work.

Test Steps

Test steps progress is similar for manual tests and automated tests. For each test step, the acceptance tester or automation interact with the SUT (System Under Test) and compare obtained result with expected result.

Post-conditions Checking

In some test cases, the execution of test steps is not enough to verify the good functioning of the SUT. The state of the system after the test steps progress must be verified too.

Most of the time it consists in verifying persistent data test inside a database or inside a file.

During a manual test, postconditions are often difficult to verify. Just as the setting-up of pre-requisites, the acceptance tester must use the tested application. In the previous example, the only way for the acceptance tester to verify the account creation is using the tested Man/Machine Interface.

In an automated test, verification of post-conditions can be done independently of the tested application. The account creation will be verified consulting directly the database. It is then possible to test a creation function even if the consultation function doesn't work.

In this way, automated tests never use the tested application to verify post-conditions.

Cleaning Up

In some cases, the test can also have a step of system cleaning after post-conditions verification. It allows to be sure that the tested system is reseted before the execution of the next test case.

This step can be omitted when the step of setting-up pre-requisites is enough to guarantee the state of the SUT. When this step exists, it is executed whatever the test result is (success, failure, error).

Results Storage

The result of each test case is saved after the execution.

3 kind of results are possible for an automated test :

- *Success*
- *Failure* : an assertion step failed (it obtained a result different than the expected result)
- *Error* : an error occurred during the test execution.

In the two last cases, saved result has a short explanatory message that allows to identify where the test crashed and if possible, the reason of the crash.

Test Campaign

Some test preconditions are common to all test cases and don't need to be implemented between each test. Those conditions are set-up once for all at the beginning of a campaign. After that all test cases are executed. And after the campaign execution, it can be necessary to clean-up the test environment (clean-up the database, stop server programs needed for the tests execution...).

At the end of an execution campaign, an execution report is created from tests results. This report describes the result of each test case : success, failure or error with an explanatory message in the two last cases.

Here is a figure showing the different steps of an automated tests campaign execution :

7.1.3 Glossary

Capture/Playback tool : A type of test execution tool where inputs are recorded during manual testing in order to generate automated test scripts that can be executed later (i.e. replayed). These tools are often used to support automated regression testing.

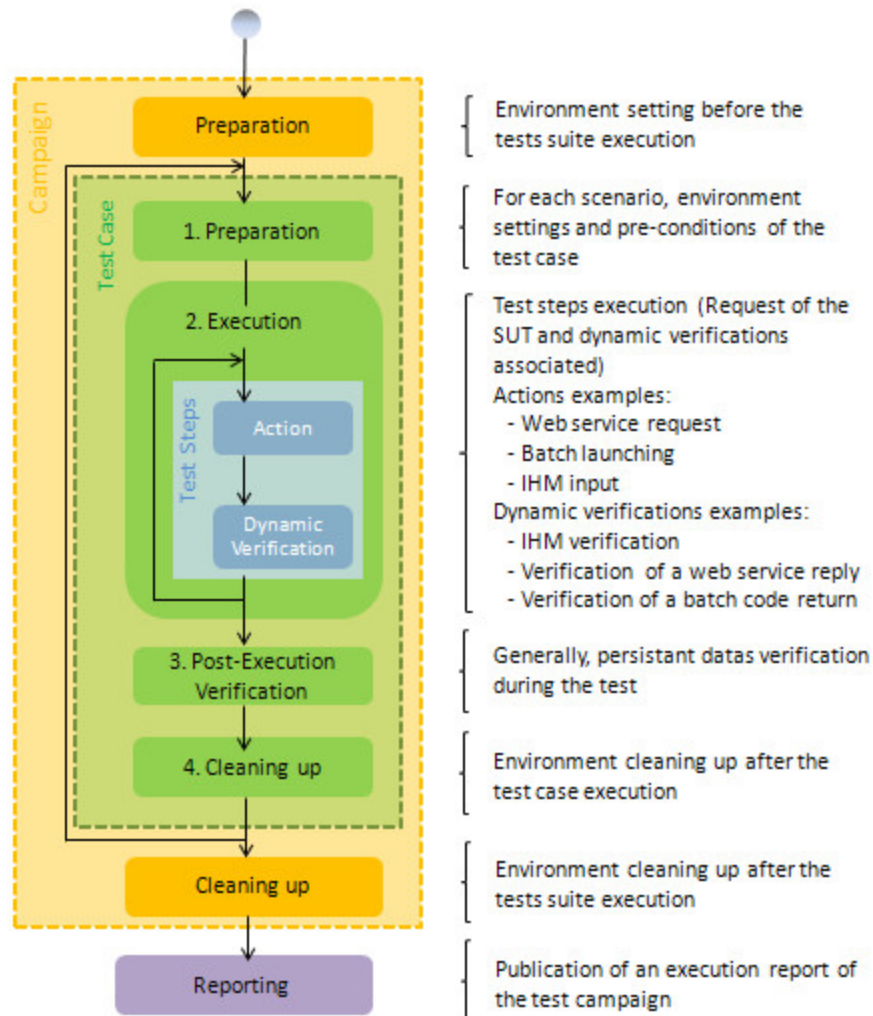
Driver : A software component or test tool that replaces a component that takes care of the control and/or the calling of a component or system.

DSL : Domain Specific Language. It's a language specifically created for a domain area.

Expected result : The behavior predicted by the specification, or another source, of the component or system under specified conditions.

Failure : Deviation of the component or system from its expected delivery, service or result.

False-fail (false-positive) result : A test result in which a defect is reported although no such defect actually exists in the test object.



Project Automation Modules : **Squash TF** components are grouped together as fonctionnal modules (Sahi module, Selenium module...). These modules are plugins, they can be added or removed of the framework based on fonctionnal needs.

Post-condition : Environmental and state conditions that must be fulfilled after the execution of a test or test procedure.

Pre-condition : Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure.

Regression testing : Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

Requirement : Extracted from the design documentation and business administration rules they come from, the requirements describe the application expected behaviors.

Step : Phase of the functional path set up in a script. Each step verifies an expected result.

SUT : System Under Test.

Test case : Functional path to execute in order to verify the conformity of functions. The test case is defined by a data set to determine, a script to execute and expected detailed results.

Test suite : A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one.

Test execution automation : The use of software, e.g. capture/playback tools, to control the execution of tests, the comparison of actual results to expected results, the setting up of test preconditions, and other test control and reporting functions.

- *Overview*
- *A small example to illustrate*
- *SKF benefits*

SKF (aka Squash Keyword Framework) is a Keyword oriented framework for test automation.

Here's a quick overview of our framework :

- A **test script** is written with a composition of **keywords**
- A **keyword** is an action written in (nearly) natural language. It's implemented by a **macro**
- A **macro** (also known as “shortcut” in SKF) is made of :
 - A **hook** : It's the signature of the macro. It describes how other macros and test scripts can call it
 - An **expansion** : It contains a list of **instructions** that'll be executed by the macro.
- An **instruction** can refer to different elements.
 - It can be :
 - * An user-defined macro
 - * A framework's builtin macro
 - * (If needed a framework's low level instruction)
 - It can also have some inputs and/or outputs that corresponds to :
 - * **Input** : The target / SUT definition, test data, testing scripts (ex : SQL query, SoapUI project, Selenium Java project, ...)
 - * **Output** : An element used for assertion or as input for other instructions

A small example to illustrate

Let's have a look at a small example

>> Here is a SKF test script :

```

..//Check the stock when a user buy a black cat

# I have 4 black cats in stock
# User logs in with login john.doe@foo.test and password XXX , buy 1 black cats and logs out
# I have 3 black cats in stock

```

This SKF test script is composed of three steps and use two keywords :

- # I have {number_of_pets} black cats in stock (Used two times)
- # User logs in with login {user_login} and password {user_password} , buy {number_of_cats} black cats and logs out

In order to make this SKF test script work we have to implement two macros.

>> The # I have {number_of_pets} black cats in stock macro :

```

# I have {number_of_pets} black cats in stock
=>

# EXECUTE_SQL db/query_black_cat_stock.sql ON myDb AS query_result
// Macro : # EXECUTE_SQL {file} ON {database} AS {result}
// Provided by the framework in the DB plugin

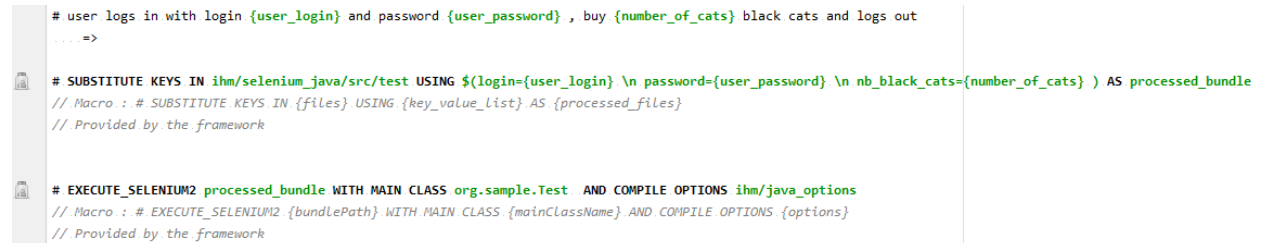
# ASSERT query_result HAS {number_of_pets} rows
// Macro : # ASSERT {query_result} HAS {number_of_pets} rows
// To define in the project (That's why it's red)

```

This macro has two instructions in its expansion :

- The first instruction is a macro provided by the framework in the database plugin (builtin macro). It has 2 input parameters and 1 output parameter:
 - Input :
 - * db/query_black_cat_stock.sql : A file with a SQL query
 - * myDb : A database named “myDb”
 - Output :
 - * query_result : Wrapped result of the query
- The second instruction is also a macro but it needs to be implemented. This instruction has 2 inputs :
 - query_result : Resource produced by the previous instruction
 - number_of_pets : Test data

>> The # User logs in with login {user_login} and password {user_password} , buy {number_of_cats} black cats and logs out macro :



```
# user logs in with login {user_login} and password {user_password} , buy {number_of_cats} black cats and logs out
...=>

# SUBSTITUTE_KEYS_IN ihm/selenium_java/src/test USING ${login={user_login} \n password={user_password} \n nb_black_cats={number_of_cats} ) AS processed_bundle
// Macro : # SUBSTITUTE_KEYS_IN {files} USING {key_value_list} AS {processed_files}
// Provided by the framework

# EXECUTE_SELENIUM2 processed_bundle WITH MAIN CLASS org.sample.Test AND COMPILE OPTIONS ihm/java_options
// Macro : # EXECUTE_SELENIUM2 {bundlePath} WITH MAIN CLASS {mainClassName} AND COMPILE OPTIONS {options}
// Provided by the framework
```

In this second macro the expansion is composed of two macros provided by the framework.

Note: Screenshots come from IntelliJ IDEA combined with our plugin for coloration and autocompletion

CHAPTER 10

SKF benefits

- The framework has been built with the **separation of concerns** principle in mind. This leads us to a **multi layer** approach :
 - **Top level** : Test script in natural language
 - **Middle level(s)** : Test implementation
 - **Ground level** : The framework with its modular architecture and resources
- Separating test scripts from their implementation brings more test **robustness** : test implementation changes whereas test script doesn't.
- The use of **natural language** for test scripts makes them more **readable** and easy to write. This makes the writing of SKF test scripts by QA tester possible.
- Implementation of keywords still requires technical skills (and you need to know how to use the targeted robot). However this aspect is **reduced** with the builtin macros provided by the framework.
- SKF is built on a **modular architecture** using plugins : one plugin for each type of test / robot. Each plugin brings the macros (and their associated low level instructions) needed to ease the implementation work. Our aim is to provide the widest set of builtin macro so that the user never have to use low level instructions.
- Its modular architecture gives the possibility to **extend** the capacity of the framework by creating new plugins.
- The writing, for either test scripts or macros, is eased with our IDE plugins (coloration and completion):
 - IntelliJ IDEA plugin
 - Eclipse tools

Note: Squash Keyword Framework (SKF) had a first life as Squash TA framework. The rebranding decision was taken when we decided to focus on the keyword approach. (You will surely find some reference to TA at some places). The changes accompanying this new approach is still a work in progress. Our first focus is a new IntelliJ IDEA plugin to ease the writing of test scripts.
