# EXCELREST

## 1. Overview

The EXCELREST aims to be a scriptable RESTful test client that uses re-usable cases written on a excel spreadsheet making easier to maintain, create new test cases, avoiding re-work and making web services testing straightforward, extensible and effective. The tool itself is platform independent and can be used to test other services and projects that feature RESTful web interfaces.

EXCELREST is a tool based on Java that will send requests using different HTTP methods to test the REST API service. Using the EXCELREST, the team can manipulate the URL, request method, body and header content.

The tool itself is platform independent and can be used to test other services and projects that feature RESTful web interfaces.

## 2. Instalation

### 2.1. Requirements

* Operational system:
  + <http://www.oracle.com/technetwork/java/javase/config-417990.html>
* Software:
  + Java JRE 7

### 2.2. Installing

1. Java JRE 7:
   * <http://docs.oracle.com/javase/7/docs/webnotes/install/index.html>
2. EXCELREST source code:
   * git clone ssh://*USERNAME*@cgit-pro.austin.hpicorp.net:29418/excelrest

## 3. Running

### 3.1. Sintax

java -jar tillamook-tests.jar /config=config.xml /build=build\_number

/config=CONFIG\_FILE - Sets the config file to be used

/build=BUILD\_NUMBER - Sets the build number that will be tested

### 3.2. The Config File

The config file that is used describes basic target server information. The syntax is as follows:

<configuration>

<execution>

<testcase run=*"TRUE|FALSE"*> <!-- Sets if the test cases should be executed or not -->

<input>xls\_input\_file</input> <!-- Sets the location were the test cases are defined-->

<output>xls\_output\_file</output> <!-- Sets the location were the results should be generated-->

<rallydev>

<updateTestCase run=*"TRUE|FALSE"*></updateTestCase> <!-- Sets if the test cases should be updated on RallyDev reflecting the test cases defined in the input spreadsheet or not -->

<registerTestResult run=*"TRUE|FALSE"*></registerTestResult> <!-- Sets if the results should be registered on RallyDev for each test case or not -->

</rallydev>

<postmail run=*"TRUE|FALSE"*></postmail> <!-- Sets if an email with the results should be sent or not -->

</testcase>

<defectreport run=*"TRUE|FALSE"*> <!-- Sets if the defect report should be executed or not -->

<output>xls\_output\_file</output> <!-- Sets the location were the results should be generated-->

<state-list> <!-- Sets the states that should be query [submitted, open, fixed, closed] -->

<state>state</state>

<state>state</state>

</state-list>

<postmail run=*"TRUE|FALSE"*></postmail> <!-- Sets if an email with the results should be sent or not -->

</defectreport>

</execution>

<queue> <!-- Sets the queue configuration used to test the rabbitMQ interface (doesn’t work with proxy ) -->

<username>username</username> <!—Default is ‘guest’ -->

<password>password</password> <!—Default is ‘guest’ -->

<vhost>vhost</vhost> <!—Default is ‘/’ -->

<host>xx.xxx.xx.xxx</host> <!—were ‘x’ specify the IP address of the host-->

</queue>

<rest> <!-- Sets the configuration used to test the RESTful web interfaces -->

<url>[http://XX.XXX.XX.XXX:PPPP</url](http://XX.XXX.XX.XXX:PPPP%3c/url)> <!—were ‘x’ specify the IP address of the host and ‘P’ the port-->

</rest>

<rallydev> <!-- Sets the configuration used to access the RESTful web interfaces -->

<url>https://rally1.rallydev.com/</url>

<username>alessandro.hunhoff@hp.com</username>

<password>passwordtest01!</password>

<project>Tillamook</project>

</rallydev>

<proxy> <!-- Sets the proxy configuration -->

<host>proxy\_host</host>

<port>port\_number</port>

</proxy>

<email> <!-- Sets the email configuration -->

<smtpName>xx.xxx.xx.xxx</smtpName> <!—were ‘x’ specify the IP address of the smtp server-->

<msgTxt>Message body</msgTxt>

<subjectTxt>Message subject</subjectTxt>k

<fromAddress>sender email address</fromAddress>

<toAddress-list>

<toAddress>recipient email address</toAddress>

<toAddress>recipient email address</toAddress>

</toAddress-list>

</email>

</configuration>

### 3.3. Runtime Feedback and Logs

After finished, the executable will exit with code 0 if successful or -1 if any case failed. This is useful if you’re using the tool to validate the system and check if it’s ok. Either way, the reports locations can be configured on the config file and log4j.properties file.

### 3.4. Reports

By default, the test tool generated reports in two different formats: XLS (containing all test results) and TXT (for a detailed execution report). Both files are generated as specified in the config files (XML configuration file and log4j.properties file).

There is an option in the config file to configure if an email should be sent after the test execution with the results.

### 3.5. DEFECT Reports

The STAT generates reports by accessing the RallyDev RESTful web interface, enabling the automation of defect reports in XLS format. In order to use this feature the config file (XML configuration file) need to be configured informing all the defectreport and rallydev parameters.

# 4. Writing Tests

The tests are described in a XLS spreadsheet that should have two different tabs, one for each type of test that should be executed. The queue\_tests tab represents all RabbitMQ tests that will use this interface and, the rest\_tests tab represents all RESTful web interface tests.

The system is fully integrated with the RallyDev RESTful interface, using the XLS spreadsheet as data input. All test cases will be created or updated, on RallyDev according the information provided in this spreadsheet if the RallyDev option were set TRUE in the config file. Also, all execution will be registered on RallyDev if registerTestResult were set TRUE.

## 4.1. TEST CASES XLS File

### 4.1.1. Format

**Default information:**

* **tc\_id** – there is two option to specify the test case ID:
  + **Using RallyDev:** this field can be filled by the user with the current RallyDev test case ID or, the user can ***let it blank*** and execute the STAT with the updateTestCase parameter set as true in the XML config file. Executing the updateTestCase, new test cases will be created on RallyDev for all the IDs that were not found automatically updating the spreadsheet with the new ID. When the ID is found the test case on RallyDev will be updated with the information contained in the test cases spreadsheet.

IMPORTANT: running the STAT with RalllyDev updateTestCase both tables should be closed before start the STAT execution in order to successfully be able to update the tables.

* + **Not using RallyDev:** the user should type one unique ID for each test case in order to use the result of one test case on another.
* **tc\_name** – the test case name;
* **tc\_description** – the test case description;
* **tc\_assertions –** the verification that should be executed;
* **tc\_vars –** the variables that should be used;
* **tc\_execute –** define if the test case should be executed ‘TRUE’ or not ‘FALSE’;

**queue\_tests** tab:

* **tc\_queue\_to\_publish –** the queue were the information should be sent;
* **tc\_message\_to\_publish –** the message that should be sent to the queue;
* **tc\_queue\_to\_consume –** the queue that should received the information already processed;

**rest\_tests** tab:

* **tc\_path –** Target path that will be joined with the “URL” field from the config file to send the request;
* **tc\_method –** Can be either GET, POST, PUT or DELETE
* **tc\_body –** set the request body;
* **tc\_headers –** set the request header;

### 4.1.2. Variables

The local var can be used to replace some information during runtime. This information can be set directly as a local var containing exactly what need to be replaced or, the information can be get from a response of another test case. The following format should be respected: **type=name=content**

Where the type can be: local\_var, jsonkey and, response.

* **local\_var type:** the name will be searched in all test cases fields and replace by the content;
* **jsonkey type:** the name will be searched in the response of the test case informed in the content field. Also, the jsonkey word will be search and replaced in the current test case by the value found in the response of the previous search;
* **response type:** the name will be searched in the response of the test case informed in the content field. Also, the name word will be search and replaced in the current test case by the value found in the response of the previous search;

### 4.1.3. ASSERTIONS

The assertions are used to validate the request response and the following format should be respected: **type=content.**

Where the type can be: status, file.exists, string.contains and, special case.

* **Status:** Compares the response status with the expected value. No additional arguments.  
  Example:
  + status=200
* **file.exists:** check if a file were downloaded in the location specified. The file is removed after the execution.

Example:

* + file.exists=C:\git\tillamook\tillamook-tests\file.tmp
* **String.Contains:** check if the string specified can be found in the response body;

Example:

* + string.contains={"status" : {"error" : false
* **Special.Case:** Special cases are non-standard requests that use actual code from the within the test tool to execute tasks. Special cases are used for situations where only requests aren’t enough to perform an action or run a test.

Example:

* + special.Case=isEmpty

This special case will verify if the RabbitMQ queues are empty after finish the test execution.