**Automation Tests**

**Development Documentation**

[General](#_1fob9te)

[Scenarios:](#_3znysh7)

[Sequence diagrams](#_4f1mdlm)

[Sequence diagram for get by id operationss](#_2u6wntf)

[Sequence diagram for get all operations](#_19c6y18)

[Sequence diagram for delete operations](#_3tbugp1)

[Given](#_2et92p0)

[When](#_3dy6vkm)

[Then](#_1t3h5sf)

[Project](#_2s8eyo1)

[Features naming criteria and composition](#_17dp8vu)

[Steps](#_lnxbz9)

[Prepare request](#_35nkun2)

[Authentification](#_1ksv4uv)

[Headers](#_44sinio)

[Parameters](#_2jxsxqh)

[CSRF Token](#_z337ya)

[Get Readable Keys](#_3j2qqm3)

[Make SQL queries](#_1y810tw)

[Logs](#_4i7ojhp)

[Make request](#_2xcytpi)

[Check response](#_1ci93xb)

[Check JSON and Code returned](#_3whwml4)

[JSON returned contains a single element (ex. Get by id):](#_2bn6wsx)

[JSON returned contains a list of elements (ex. Get all):](#_qsh70q)

[Check response with SQL](#_3as4poj)

[SQL returns a single element](#_1pxezwc)

[SQL returns a list of element](#_49x2ik5)

[Check Log](#_2p2csry)

[Queries](#_147n2zr)

# General

The database used by the automation test project is MySQL, which should be properly configured and containing all the necessary data to be able to execute the tests correctly. To see how to do this, refer to this [document](https://docs.google.com/a/globant.com/document/d/1rT_r0vr5JxGln1dk0UqRZLYqy4_EAaKKUTmkSHytKCQ/edit?usp=sharing).

## Scenarios:

**Scenario:** Name of the test

- format: <*Method>* to <*endpoint>* <*specific case>*

*Ex: GET to /api/v1/versions/<rk-version-id> with valid data*

[**Given**](#_3o7alnk) Steps to set up the needed environment to run the test given the specific case being tested.

[**When**](#_23ckvvd)Steps to properly run the test

[**Then**](#_ihv636)Verifies that the test has been run as expected

A way to include similar various cases on one scenario is the following:

**Scenario Outline:** PUT to /api/v1/nodes/<rk-node-id> with an invalid boolean field

**Given** I get a valid rk for **nodes** with "**nme**" equal to "**'node\_for\_axon\_agent'**"

**And** I get a CSRF token with the user "**administrator**"

**And** I provide the following headers:

**"""**

**{ "Accept": "application/json", "Content-type": "application/json" }**

**"""**

**And** I add to the headers the CSRF token for the user "**administrator**"

**And** I provide the following data:

**"""**

**{ "name": "New node name", "<field>": "<value>" }**

**"""**

**When** I send a PUT request to "**/api/v1/nodes/valid-rk-node**"

**Then** the response status should be **400**

**And** JSON should have the field "**error**" containing the pattern "**(.\*)**"

**Examples:**

**|** *field* **|** *value* **|**

**| isDisabled | INVALID |**

**| auditEnabled | INVALID |**

**| eventGeneratorEnabled | INVALID |**

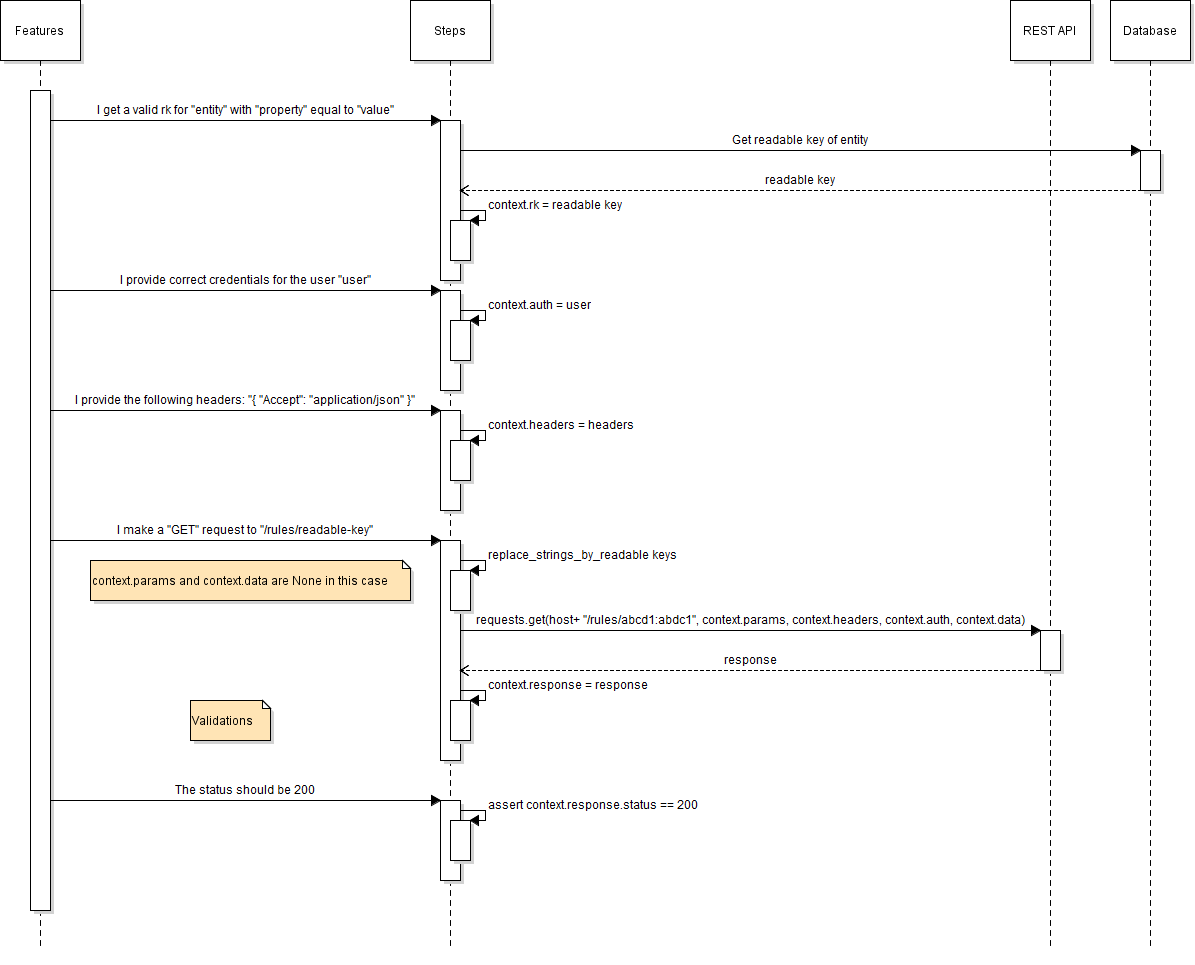
**| realTimeEnabled | INVALID |**

This scenario will be run once for each of the values in **Examples**.

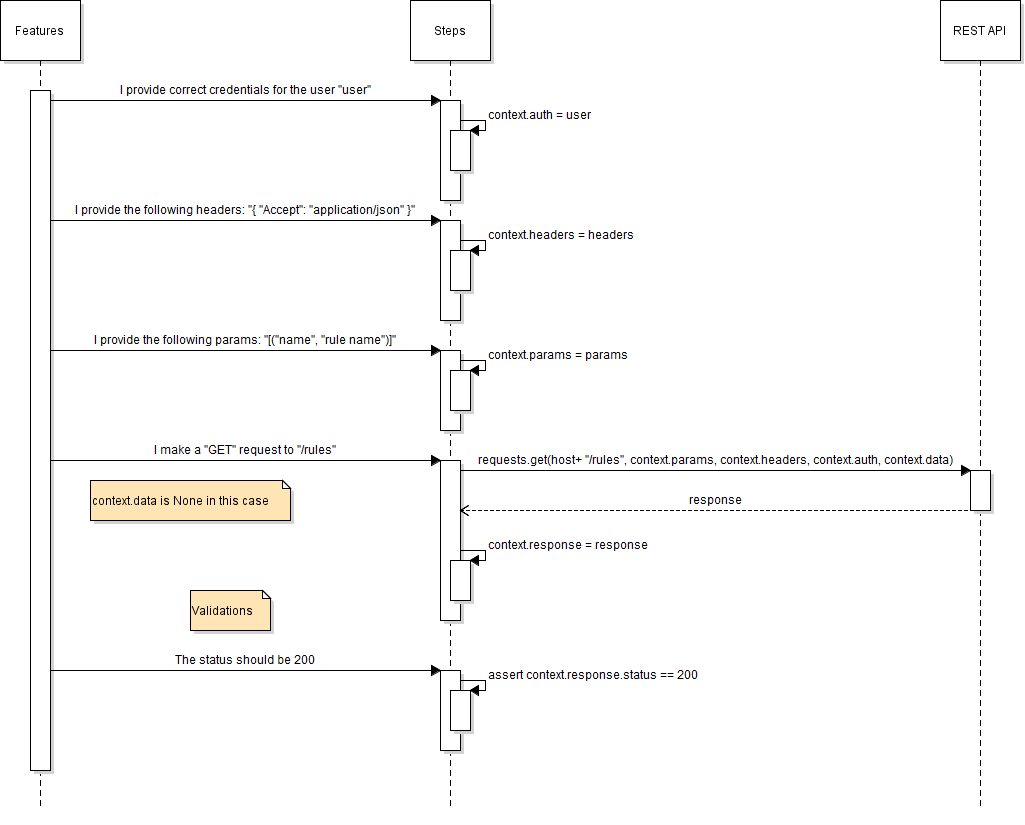
# 

# Sequence diagrams

## Sequence diagram for get by id operations



## Sequence diagram for get all operations



## Sequence diagram for delete operations

### 

### 

### Given

Depending on the nature of the test, each test should use different steps from: [Steps to prepare the request](#_35nkun2) and [CSRF Token](#_z337ya).

Queries to the database to retrieve readable keys are done with steps from: [Get Readable Keys](#_3j2qqm3); and to retrieve elements steps from [Make SQL queries](#_1y810tw) should be used. The proper queries used by this steps can be found on [Queries](#_32hioqz).

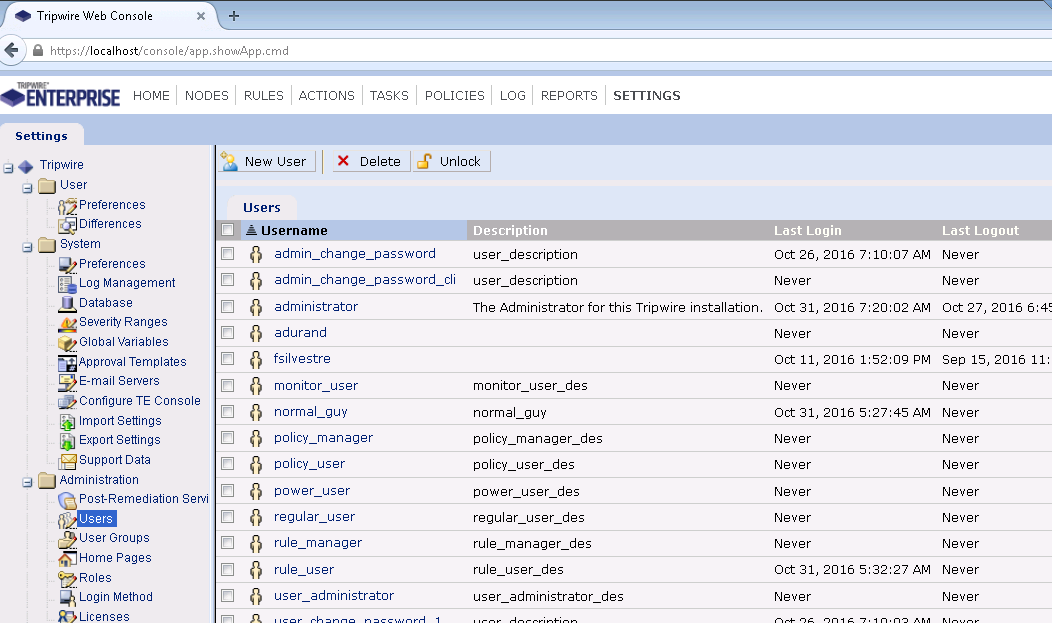
In this part is where the following steps are done:

* Authenticate the user ([\*](#tyjcwt))
  + I get a CSRF token with the user "**administrator**"
  + I add to the headers the CSRF token for the user "**administrator**"
  + I provide **correct** credentials for the user "**rule\_user**"
  + I provide **incorrect** credentials for the user "**administrator**"
* Set up the request header
  + **"Accept": "application/json"**
  + **"Content-type": "application/json"**
* Sets the parameters of the request
* Gets needed readable keys
* Performs SQL queries to retrieve elements, used to compare them later with the result of the request to the API.

Users: Each one has one or more Roles associated with different sets of permissions.

Ex. - Administrator has full control. Can change system settings including user management and security.

* Monitor\_user has no control. Can view data but cannot add or change data.

To search for all the users and their roles, go to: Console > Settings > Administration > Users

Additionally, the users and passwords saved to be used by the automation tests can be found on the automation project on the file:

**te-rest-api-automation/mapping/mapping.py**

### When

On this part the requests are sent. The steps that could be found here: [Make request](#_2xcytpi).

One way to make request using a rk on the path, is to use a code string which will be replaced by the method:

**string\_to\_readable\_key.replace\_strings\_by\_readable\_keys**

This function replaces a substring (the code string, if found) inside a given string, with the corresponding readable key previously saved on **context.rk\_entity**.

Ex.

Previously: **Given** I get a valid rk for **roles** with "**twprin.nme**" equal to "**'role\_name'**"

Which will save:

context.rk\_role = base36.encode(role[**'mappingId'**]) + **':'** + base36.encode(role[**'objectId'**])

Then, the step: **When** I send a "**GET**" request to "**/api/v1/roles/valid-rk-role**"

Will in turn execute the following code:

**if** hasattr(context, **'rk\_role'**):

string = string.replace(**'valid-rk-role'**, context.rk\_role)

And the String will be replaced to:

**“/api/v1/roles/-1y2p0ij32e8c6:-1y2p0ij323ii6”**

### Then

The steps used to verify that the result returned by the API, given the sent request, could be found on [Check response](#_1ci93xb).

In this part is where the following steps are done:

* Verify the expected status code (\*)
* In case of list of elements:
  + Verify the number of elements returned
  + Verify the attributes of every element returned
  + Verify that an element is contained on the JSON returned
* In case of a single element returned
  + Verify that it (/does not) contains a field, or a parameter
  + Verify that a field or a parameter contains a value
  + Verify that it follows a certain pattern
* In case of error
  + JSON should have the field "**error**" containing the pattern "**(.\*)**"

The specific codes and its meanings can be found either on Swagger, under “Reponse Message” on each of the endpoints; or if the endpoint is on development, on the user story’s details.

In general, the used HTTP Status Codes are:

**200 / 201 / 204: Success**

**400: Malformed data supplied**

**401: Unauthorized (Unauthenticated user)**

**403: Forbidden - required permissions**

**404: Element not found**

# Project

To see all the scenarios that currently exists on TE refer to [Automation Tests Cases](https://docs.google.com/a/globant.com/document/d/1J0D7UXjyIF7h0Q731yHX5_uAvZYJukJpyacSO9Hfo4g/edit?usp=sharing).

For each of the endpoints, the following cases should exists:

(Note: these are recommended general cases, but the actual cases will depend on the specific endpoint)

* All the error codes returned by the endpoint (not all of them apply to all the endpoint, check which ones are contemplated by the endpoint)
  + 400 - Malformed data supplied
    - Usually this error is returned when a parameter is sent in an incorrect format. There should be one case for each incorrect format, for each parameter.
      * It is highly recommended to group this cases into one scenario outline.
  + 401 - Unauthorized
    - Usually this is tested providing incorrect credentials for user “administrator”
  + 403 - Forbidden
    - Usually this is tested providing correct credentials for user “rule\_user” (but, the user will depend on the required permissions needed for the endpoint)
  + 404 - Not found
    - Usually this error is returned when looking for an element that does not exist.
* Success Cases (200 /201/ 204):
  + On GET endpoints:
    - Verify that number of elements returned are what should be expected according to the parameters
    - Control the content of the elements returned
    - Compare with the data from the database
  + On POST endpoints:
    - Verify that updated fields are returned updated on the JSON responded
    - Verify that fields not updated are kept the same on the JSON responded
    - Verify that not updatable fields cannot be overwritten
  + On DELETE endpoints:
    - (When possible) Verify that the element has been deleted performing a GET to the element, and not receiving it.
  + On PUT endpoints:
    - Verify that not updatable fields cannot be overwritten
    - Verify that data sent is present on returned JSON

Don’t forget to include:

* incompatible values/parameters/formats cases,
* special cases (ex. Axon nodes cannot have ipAddresses),
* Verify if a modification/run should (or shouldn’t) perform a change on the system (ex. Write a log)

### Features naming criteria and composition

Automation tests cases are written on scenarios, which in turn are contained on a feature.

* **Endpoint.feature**: CRUD (Get by id/Update/Create/Delete)
* **Endpoint\_queries.feature**: Get all
* **Endpoint\_function.feature:** (function being: root, parentGroup, retire, run, execute, remediation, etc)
* **Endpoint\_links.feature:** Get/Creates/Deletes links between Groups

The different endpoints can be found at the Swagger UI:

# 

# 

# Steps

## Prepare request

### Authentification

Gets the correct/incorrect password saved on mapping.py for the selected user, and sends them to step *‘***I provide credentials with username "(.\*)" and password "(.\*)"’:**

@step(**'I provide (correct) credentials for the user "(.\*)"'**)

@step(**'I provide (incorrect) credentials for the user "(.\*)"'**)

Sets the request authentication as: HTTPBasicAuth(username, password)

@step(**'I provide credentials with username "(.\*)" and password "(.\*)"'**)

### Headers

Sets the request’s headers:

@step(**'I provide the following headers'**)

Ex. I provide the following headers:

**"""**

**{ "Accept": "application/json", "Content-type": "application/json" }**

**"""**

### Parameters

Adds the following parameter to the request body:

@step(**'I provide the following params'**)

Ex. **And** I provide the following params:

**"""**

**[("id", "valid-rk-policy"), ("name", "MS Windows Server 2008 R2 DC - CIS v2.1.0")]**

**"""**

Step used to send a request body to the API:

@step(**'I provide the following data'**)

Ex. I provide the following data:

**"""**

**{**

**"severity": 10**

**}**

**"""**

### CSRF Token

@step(**'I get a CSRF token with the user "(.\*)"'**):

This step runs inside the following steps:

1. **Given I provide the following headers: {"Accept": "application/json"}**
2. **Given I provide correct credentials for the user "<User>"**
3. **When I send a GET request to "/api/v1/csrf-token"**
4. **Then the response status should be 200**

Finally, it saves an element

context.users[user] = {

**'csrf\_name'**: response\_json[**'tokenName'**],

**'csrf\_value'**: response\_json[**'tokenValue'**],

**'jsessionid'**: context.response.cookies[**'JSESSIONID'**]

Where response\_json is the json returned by **/csrf-token**

@step(**'I add to the headers the CSRF token for the user "(.\*)"'**):

Adds the data saved on the previous step, and inserts it on the request header

## Get Readable Keys

This steps are used to look into the database for the rk of an entity with a key equal to certain value. The entities are the ones between brackets (reports, users, user-groups, etc). The keys are the attributes of the entity.

The readable key is formed by encoding with base36 the mapping id and the object id found on the database, and concatenating both with a ‘**:**’ in between.

Example of a readable key: **-1y2p0ij32e8c6:-1y2p0ij323ii6**

(Mapping Id : Object Id)

@step(**'I get a valid rk for (reports) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (users) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (user-groups) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (roles) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (permissions) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (homepages) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (version) with "(.\*)" (equal to|like) "(.\*)"'**)

@step(**'I get a valid rk for (previous version) with "(.\*)" (equal to|like) "(.\*)"'**)

@step(**'I get a valid rk for (baseline version) with "(.\*)" (equal to|like) "(.\*)"'**)

@step(**'I get a valid rk for (element) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (version\_audit) with "(.\*)" like "(.\*)"'**)

@step(**'I get a valid rk for (nodes) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (node-groups) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (smart-node-groups) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a second valid rk for (node-groups) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a second valid rk for (smart-node-groups) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (rules) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (task) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (action) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a second valid rk for (action) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (rule-groups) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (policies) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a second valid rk for (policies) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (policy-test-results) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a second rk for (policy-test-results) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (policy-tests) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid id for (version-request) with "(.\*)" (equal to|like) "(.\*)"'**)

@step(**'I get a valid rk for (waivers) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (custom-properties) with "(.\*)" equal to "(.\*)"'**)

@step(**'I get a valid rk for (custom-properties) with "(.\*)" like "(.\*)"'**)

## Make SQL queries

This steps retrieve the elements found on the database, searching on the corresponding table given an entity, by a given key equal (or like) a given value.

The results are saved on **context.sql\_result**, which can contain a list of elements depending the case.

Each of the attributes of the elements are saved on **context.sql\_result[‘attribute’].**

Additionally, some elements will save a variable **context.sql\_result[‘length’]** with the number of elements returned by the query. (Max value: 1000 - corresponding to the paging limit)

@step(**'I make a SQL query to obtain latest element versions with'**)

@step(**'I make a SQL query to obtain historical element versions with'**)

@step(**'I make a SQL query to obtain (task) with "(.\*)" equal to "(.\*)"'**)

@step(**'I make a SQL query to obtain a (policyTest) with "(.\*)" equal to "(.\*)"'**)

@step(**'I make a SQL query to obtain a (policyTestResult) with "(.\*)" equal to "(.\*)"'**)

@step(**'I make a SQL query to obtain historical (policyTestResults) with'**)

@step(**'I make a SQL query to obtain latest (policyTestResults) with'**)

@step(**'I make a SQL query to obtain the last (remediationRequest) with "(.\*)" equal to "(.\*)"'**)

@step(**'I make a SQL query to obtain the last (version\_audit) with "(.\*)" like "(.\*)"'**)

### Logs

Verifies that a certain String is present on the log table:

@step(**'The following msg is present in the Log: "(.\*)"'**)

Searches for log messages found on the log table, with certain attributes

@step(**'I make a SQL query to obtain logMessages with'**)

@step(**'I make a SQL query to obtain (logMessages) with like'**)

@step(**'I make a SQL query to obtain last (logMessages) by "(.\*)"'**)

Delete log table ‘LE’:

@step(**'I clean the log'**)

## Make request

Gets the variable context.host which contains the string “**https://localhost**” and adds the path specified to get the complete path.

Ex. **When** I send a POST request to "**/api/v1/roles**" → Complete\_path: **https://localhost/api/v1/roles**

It sends the request to the corresponding complete path, adding the parameters from context.params and/or context.data, the headers from context.headers, the authentication from context.auth (all of them set on previous steps).

@step(**'I send a GET request to "(.\*)"'**)

@step(**'I send a POST request to "(.\*)"'**)

@step(**'I send a PUT request to "(.\*)"'**)

@step(**'I send a DELETE request to "(.\*)"'**)

This step uses the function string\_to\_readable\_key.replace\_strings\_by\_readable\_keys before executing the corresponding step from above:

@step(**'I send a "(.\*)" request to "(.\*)"'**)

Ex. **When** I send a "**GET**" request to "**/api/v1/roles/valid-rk-role**"

Will replace the String **valid-rk-role** with the String found on context.rk\_role (if found), created on one of the “Get readable Key” steps

Compares two requests:

@step(**'I send two "(.\*)" requests. To "(.\*)" and to "(.\*)"'**)

## Check response

### Check JSON and Code returned

This are the steps used to verify that the responded bodies are as expected.

@step(**'the response status should be (\d+)'**): compares status code returned with an expected code.

@step(**'the responses should be equal'**): compares two responses, and verifies that they are equal (code and json returned)

#### JSON returned contains a single element (ex. Get by id):

Verify that the element contains a field with a certain value:

@step(**'JSON should have the field "(.\*)" containing the value "(.\*)"'**)

@step(**'JSON should have the field "(.\*)" containing the value (\d+)'**)

@step(**'JSON should have the field "(.\*)" containing the value (True|False)'**)

Ex. **And** JSON should have the field "**policyName**" containing the value "**custom compliance**"

Verify that the element contains an array attribute with a certain value:

@step(**'JSON should have the property "(.\*)" containing the value "(.\*)"'**)

@step(**'JSON should have the property "(.\*)" containing the value (\d+)'**)

@step(**'JSON should have the property "(.\*)" containing the value (True|False)'**)

Ex. **And** JSON should have the property "**waivedTests.0.nodeId**" containing the pattern "**[-]?[A-Za-z0-9]+:[-]?[A-Za-z0-9]+$**"

Verify that the element does not contain a certain attribute:

@step(**'JSON should not have the field "(.\*)"'**)

Verify that the element has an attribute containing the following text:

@step(**'JSON should have the field "(.\*)" containing the following'**)

Ex. **And** JSON should have the field "**Type**" containing the following:

**"""**

**{ "value": "File", "type": "Type" }**

**"""**

Verify that the element has an attribute which contains an array with a certain number of elements:

@step(**'JSON field "(.\*)" should contain (\d+) elements'**)

Verify that the element is the same as the expected:

@step(**'JSON should be'**)

Verify that the element has a field following a certain pattern:

@step(**'JSON should have the field "(.\*)" following the pattern "(.\*)"'**)

Ex. JSON should have the field "**creationTime**" containing the pattern "**(\d){4}\-(\d){2}\-(\d){2}T(\d){2}:(\d){2}:(\d){2}.(\d){3}Z**"

Verify that the element has a field containing a certain pattern:

@step(**'JSON should have the field "(.\*)" containing the pattern "(.\*)"'**)

Verify that the element has an array attribute contains a certain pattern:

@step(**'JSON should have the property "(.\*)" containing the pattern "(.\*)"'**)

Ex. JSON should have the property "**requestData.policyTestResultIds.0**" containing the pattern "**valid-rk-policy-test-result**"

Verify that the element follows a certain pattern:

@step(**'JSON should follow the pattern "(.\*)"'**)

Verify that the element contains a certain pattern:

@step(**'JSON should contain the pattern "(.\*)"'**)

Converts the returned element’s encoding to ‘utf.8’, and verifies that the text contains a substring:

@step(**'the TEXT of the response should contain the substring "(.\*)"'**)

Verifies the number of remediation entries existing on the database:

@step(**'remediationEntry should have (\d+) elements'**)

#### JSON returned contains a list of elements (ex. Get all):

Verify the number of elements returned:

@step(**'JSON list should contain (\d+) elements'**)

@step(**'JSON list should contain less or equal than (\d+) elements'**)

@step(**'JSON list should contain more or equal than (\d+) elements'**)

Verify that at least one of the elements returned is the following:

@step(**'the JSON list should contain the following element'**)

Ex. And the JSON list should contain the following element:  
 """  
 { "waivedTests":  
 [{  
 "policyTestId": "valid-rk-policy-test",  
 "nodeId": "valid-rk-node"  
 }]  
 }  
 """

Verify that all of the elements on the JSON (do not) have a certain attribute:

@step(**'the JSON list should have attribute "(.\*)"'**)

@step(**'the JSON list should not have attribute "(.\*)"'**)

Verify that all of the elements on the JSON contains a field with a certain value

@step(**'the JSON list should have field "(.\*)" containing the following value "(.\*)"'**)

@step(**'the JSON list should have field "(.\*)" containing the following number (\d+)'**)

@step(**'the JSON list should have field "(.\*)" containing the following boolean (True|False)'**)

### Check response with SQL

This steps could be used after having done one SQL query(\*) on a previous step, and compares the responded JSON with the results found on the DB.

#### SQL returns a single element

Compares an attribute from the element returned on the JSON, and verifies that it is equal to the same attribute from the SQL result:

@step(**'JSON should have a string field "(.\*)" equal to the SQL result'**)

@step(**'JSON should have a integer field "(.\*)" equal to the SQL result'**)

@step(**'JSON should have a boolean field "(.\*)" equal to the SQL result'**)

Ex.

**Scenario:** GET to /api/v1/versions/<rk-version-id> with valid data

**Given** I get a valid rk for **version** with "**enames.EN\_DISPLAY**" **like** "**'%New Text Document.txt'**"

**And** I provide the following headers:

**"""**

**{ "Accept": "application/json" }**

**"""**

**And** I provide **correct** credentials for the user "**administrator**"

**When** I send a "**GET**" request to "**/api/v1/versions/valid-rk-version**"

**Then** the response status should be **200**

**And** JSON should have the field "**id**" containing the pattern "**[-]?[A-Za-z0-9]+:[-]?[A-Za-z0-9]+$**"

**And JSON should have a string field "elementName" equal to the SQL result**

**And** JSON should have the field "**elementId**" containing the pattern "**[-]?[A-Za-z0-9]+:[-]?[A-Za-z0-9]+$**"

**And JSON should have a string field "nodeName" equal to the SQL result**

**And** JSON should have the field "**nodeId**" containing the pattern "**[-]?[A-Za-z0-9]+:[-]?[A-Za-z0-9]+$**"

**And JSON should have a string field "ruleName" equal to the SQL result**

**And** JSON should have the field "**ruleId**" containing the pattern "**[-]?[A-Za-z0-9]+:[-]?[A-Za-z0-9]+$**"

**And JSON should have a boolean field "exists" equal to the SQL result**

**And JSON should have a integer field "severity" equal to the SQL result**

**And** JSON should have the field "**timeDetected**" containing the pattern "**(\d){4}\-(\d){2}\-(\d){2}T(\d){2}:(\d){2}:(\d){2}.(\d){3}Z**"

**And JSON should have a string field "promotionComment" equal to the SQL result**

**And JSON should have a integer field "outsideMaintenanceWindow" equal to the SQL result**

**And** JSON should have the field "**baselineVersion**" containing the pattern "**[-]?[A-Za-z0-9]+:[-]?[A-Za-z0-9]+$**"

**And** JSON should have the field "**timeReceived**" containing the pattern "**(\d){4}\-(\d){2}\-(\d){2}T(\d){2}:(\d){2}:(\d){2}.(\d){3}Z**"

Verifies that one of the elements contained on the JSON is equal to the element returned by the SQL result:

@step(**'the JSON list should contain the element equal to the SQL result'**)

#### SQL returns a list of element

Verifies that the number of elements returned on the JSON are the same as the ones returned by the SQL query:

@step(**'JSON list should have the same length that the SQL result'**)

### Check Log

Searches the log file, located on **tw\_home()\data\log\te-rest-api.log**, and validates that it contains certain Strings

(*tw\_home()*: variable set on teconfig containing the location of TE Server files)

@step(**'The following string is present in the last line of the Log: "(.\*)"'**)

@step(**'The following string is present in the second last line of the Log: "(.\*)"'**)

# Queries

Each of this functions receive **context.db** as parameter,

Additionally, they could receive:

* a **key** and/or a **value**.
* **Arguments**, with a number of keys/values

Most functions have two exceptions:

* Equal: Searches Where Key **=** Value
* Like: Searches Where Key **LIKE** Value

get\_mapping\_id\_for:

**SELECT pmId AS mappingId FROM pmap WHERE ‘Key’ = 'Value'**

get\_element\_version\_audit:

**SELECT vers.oid AS objectId**

**FROM vers JOIN elems ON elems.oid = vers.V\_EID**

**JOIN enames ON enames.EN\_OID = elems.E\_NMID**

**JOIN le ON vers.oid = le.le\_version**

**JOIN aele ON aele.oid = le.oid**

**JOIN AuditAppType ON AuditAppType.KM\_ID = aele.aele\_applicationTypeId**

**JOIN LMUsrnme on le.le\_usr=LMUsrnme.KM\_ID**

get\_version\_audit:

**SELECT en\_display as AuditName, vers.oid AS objectId, auditapptype.km\_key as process\_name, LMUsrnme.KM\_KEY AS username, le\_tm as timestamp, le\_msg as detail, auditeventtype.KM\_KEY as eventType, auditaccesstype.KM\_KEY as accessType**

**FROM vers JOIN elems ON elems.oid = vers.V\_EID JOIN enames ON enames.EN\_OID = elems.E\_NMID JOIN le ON vers.oid = le.le\_version**

**JOIN aele ON aele.oid = le.oid JOIN AuditAppType ON AuditAppType.KM\_ID = aele.aele\_applicationTypeId**

**JOIN LMUsrnme on le.le\_usr=LMUsrnme.KM\_ID JOIN auditeventtype on aele.aele\_eventTypeId=auditeventtype.KM\_ID**

**JOIN auditaccesstype on aele.aele\_accessTypeId=auditaccesstype.KM\_ID**

**ORDER BY le\_tm ASC**

get\_element\_version:

**SELECT vers.oid AS objectId, enames.EN\_DISPLAY as elementName, vers.V\_EID AS elementId, nodegroupable.nme AS nodeName, nodegroupable.ref\_pmId AS nodeMapping, elems.E\_NID AS nodeId, rulegroupable.nme AS ruleName, rulegroupable.ref\_pmId AS ruleMapping, elems.E\_RID AS ruleId, vers.V\_TYPE AS changeType, vers.V\_EXISTS AS "exists", vers.V\_SEV AS severity, vers.V\_TIME AS timeDetected, vers.V\_APPID AS promotionComment, vers.V\_PKG\_HASH AS packageInfoHash, vers.V\_OUT\_WIN AS outsideMaintenanceWindow, vers.V\_CTX\_ID AS scanId, attribs.F\_MD5 AS md5, attribs.F\_SHA AS sha1, attribs.F\_SHA256 AS sha256, attribs.F\_SHA512 AS sha512**

**FROM vers LEFT JOIN attribs ON attribs.F\_ID = vers.V\_ATTR JOIN elems ON elems.oid = vers.V\_EID JOIN grpbl AS nodegroupable ON nodegroupable.oid = elems.E\_NID JOIN grpbl AS rulegroupable ON rulegroupable.oid = elems.E\_RID JOIN enames ON enames.EN\_OID = elems.E\_NMID**

get\_element\_baseline\_version:

adds**: WHERE vers.V\_TYPE = '0' AND {0} = {1}**

Get\_element

**SELECT elems.oid AS objectId, grpbl.nme AS elementName**

**FROM te.elems JOIN te.grpbl ON elems.E\_RID = grpbl.oid**

Get\_node

**SELECT nme AS name, ref\_pmId AS mappingId, grpbl.oid AS objectId, grpbl.trackingId**

**FROM grpbl JOIN node ON grpbl.oid = node.oid**

Get\_x\_nodes: Get x number of nodes

Get\_node\_group

**SELECT ref\_pmId AS mappingId, grpbl.oid AS objectId**

**FROM grpbl JOIN pmap ON pmap.pmId = grpbl.ref\_pmId**

**WHERE pmT = "Node Group"**

Get\_smart\_node\_group

**SELECT ref\_pmId AS mappingId, grpbl.oid AS objectId**

**FROM grpbl JOIN pmap ON pmap.pmId = grpbl.ref\_pmId**

**WHERE pmT = "Smart Node Group"**

Get\_rule

**SELECT ref\_pmId AS mappingId, grpbl.oid AS objectId**

**FROM grpbl JOIN rule ON grpbl.oid = rule.oid**

Get\_rule\_group

**SELECT ref\_pmId AS mappingId, grpbl.oid AS objectId**

**FROM grpbl JOIN pmap ON pmap.pmId = grpbl.ref\_pmId**

**WHERE pmT = "Rule Group"**

Get\_policy

**SELECT ref\_pmId AS mappingId, grpbl.oid AS objectId**

**FROM grpbl JOIN policy ON grpbl.oid = policy.oid**

Get\_user

**SELECT twprin.ref\_pmId AS mappingId, twprin.oid AS objectId**

**FROM twprin JOIN twuser ON twprin.oid = twuser.oid**

Get\_role

**SELECT twprin.ref\_pmId AS mappingId,twprin.oid AS objectId**

**FROM twprin JOIN twrole ON twprin.oid = twrole.oid**

Get\_user\_group

**SELECT twprin.ref\_pmId AS mappingId twprin.oid AS objectId**

**FROM twprin JOIN twgroup ON twprin.oid = twgroup.oid**

Get\_permission

**SELECT ref\_pmId AS mappingId,oid AS objectId**

**FROM objperm**

Get\_homepage

**SELECT ref\_pmId AS mappingId,oid AS objectId**

**FROM hp\_homepage**

Get\_policy\_test\_result

**SELECT policytestresult.oid AS objectId, grpbl\_2.ref\_pmId AS mapPolicyTest, policytestresult.f\_policyTestID AS policyTestId, grpbl\_1.ref\_pmId AS mapNode, policytestresult.f\_nodeID AS nodeId, grpbl\_1.nme AS nodeLabel, grpbl\_2.nme AS policyTestName,**

**policytestresult.f\_state AS state, policytestresult.f\_time AS creationTime, v.oid AS elementVersionId, enames.EN\_DISPLAY as elementName,**

**v.V\_EID AS elementId, vpmap.pmId AS mapElementVersion, epmap.pmId AS mapElement**

**FROM policytestresult JOIN grpbl grpbl\_1 ON policytestresult.f\_nodeID = grpbl\_1.oid**

**JOIN grpbl grpbl\_2 ON policytestresult.f\_policyTestID = grpbl\_2.oid JOIN te.versiontestresult vtr ON vtr.f\_nodeID = policytestresult.f\_nodeID JOIN vers v ON vtr.f\_VersID = v.oid JOIN te.elems ON elems.oid = V.V\_EID JOIN te.enames ON enames.EN\_OID = elems.E\_NMID, (Select pmId from pmap where pmT = 'Version') AS vpmap,**

**(Select pmId from pmap where pmT = 'Element') AS epmap**

Get\_policy\_test

**SELECT grpbl.ref\_pmId AS mappingId, grpbl.oid AS objectId, grpbl.nme AS name,grpbl.dsc AS description**

**FROM policytest JOIN grpbl ON policytest.oid = grpbl.oid**

get\_version\_request

**SELECT \* FROM api\_command\_request**

Get\_waiver

**SELECT policywaiver.ref\_pmId AS mappingId, policywaiver.oid AS objectId, policywaiver.f\_name AS name, policywaiver.f\_description AS description, policywaiver.f\_policyId AS policyId, policywaiver.f\_startTime AS startTime, policywaiver.f\_expiration AS expiration, policywaiver.f\_grantedBy AS grantedBy, policywaiver.f\_responsible AS responsible,policywaiver.f\_closed AS closed**

**FROM policywaiver**

Get\_custom\_property

**SELECT ref\_pmId AS mappingId,oid AS objectId**

**FROM custtype**

Get\_remediation\_request

**SELECT \***

**FROM api\_command\_request**

**WHERE completionTime = (Select** *MAX***(completionTime) From api\_command\_request)**

Get\_task

Get\_report

Get\_action

**SELECT ref\_pmId AS mappingId, grpbl.oid AS objectId**

**FROM grpbl**

Get\_remediation\_entry

**SELECT re.oid AS ObjId, re.ref\_pmId AS mapId, nme**

**FROM te.remediationentry reJOIN remediationworkorder rwm ON rwm.oid=re.f\_parentId**

Get\_log\_messages

**SELECT oid as objectId, ref\_pmId as mappingId, le\_tm as time, le\_msg as message, lmusrnme.KM\_KEY as username, le.le\_lev as level, pmap.pmT as type**

**FROM le JOIN lmusrnme on le.le\_usr=lmusrnme.KM\_ID JOIN pmap on pmap.pmId = le.ref\_pmId**

Results\_to\_dict\_array: *"""Convert a result of a query to a list of dicts"""*

verify\_log

**SELECT le\_msg FROM le**

clean\_log

**DELETE FROM le**