The RobotResults2RQM tool

Tran Duy Ngoan (RBVH/ECM1)

May 05, 2022

CONTENTS:

CHAPTER

ONE

ROBOTRESULTS2RQM TOOL'S DOCUMENTATION

1.1 Introduction:

RobotResults2RQM tool provides the ability to interact with RQM resources (test plan, test case, build, ...).

RobotResults2RQM tool uses RqmAPI to:

- get resource: by given ID or all vailable entities of resource type.
- update resource: by given ID.
- create new resource: with resource templates under RQM_templates folder

1.2 Sample RobotFramework Testcase:

For test case management, we need some tracable information such as version, testcase ID, component, ... to manage and track testcase(s) on RQM.

So, this information can be provided in **Metadata** (for the whole testsuite/execution info: version, build, ...) and **[Tags]** information (for specific testcase info: component, testcase ID, requirement ID, ...).

Sample Robot testcase with the neccessary information for importing to RQM:

```
*** Settings ***
Metadata    project       ROBFW              # Test Environment
Metadata    version_sw    SW_VERSION_0.1     # Build Record
Metadata    component     Import_Tools       # Component - is used for test case
Metadata    machine       %{COMPUTERNAME}    # Hostname
Metadata    team-area     Internet Team RQM   # team-area (case-sensitive)

*** Test Cases ***
Testcase 01
   [Documentation]   This test is traceable with provided tcid
   [Tags]   TCID-1001   FID-112   FID-111    robotfile-https://github.com/test-
↪fullautomation
   Log      This is Testcase 01

Testcase 02
   [Documentation]  This new testcase will be created if -createmissing argument
                ...  is provided when importing
   [Tags]   FID-113  robotfile-https://github.com/test-fullautomation
   Log      This is Testcase 02
```

1.3 Tool features:

By default, tool will base on provided arguments (see [Usage](#usage)) and *tcid* information in Robot test case(s) to:

- Login the RQM server and verify the provided `project`, `testplan`.
- Create build record and test environment (if already provided in Robot test case and not existing on RQM) for execution.
- Create new Test Case Execution Record - TCER (if it is not existing) bases on test case ID and `testplan` ID.
- Create new Test Case Execution Result which contents the detail and result state of test case.
- Link all test case(s) to provided `testplan`.

Besides, RobotResults2RQM tool also supports to:

- Create new test case(s) which is not existing on RQM (do not have *tcid* information) while importing result(s) to RQM with optional argument `-createmissing`.
- Update information (such as testcase name, Requirement ID, ...) of existing testcase on RQM with optional argument `-updatetestcase`.

1.4 Robot Testcase information on RQM:

For more detail how the RobotFramework testcase information is displayed on RQM, please refer be mapping table:

RQM data		RobotFramework	
Resource	**Attribute/Field**	**Testsuite/Testcase**	**Output.xml**
Build Record	Title	**Metadata** version_sw *${Build}*	`//suite/metadata/item[@name="version_sw"]`
Test Environment	Title	**Metadata** project *${Environment}*	`//suite/metadata/item[@name="project"]`
Test Case	ID	**[Tags]** tcid-*xxx*	`//suite/test/tags/tag[@text="tcid-*xxx*"]`
	Name	test name	`//suite/test/@name`
	Team Area	**Metadata** team-area *${Team_Area}*	`//suite/metadata/item[@name="team-area"]`
	Description	test doc - [Documentation]	`//suite/test/doc/@text`
	Owner	provided `user` in cli	
	Component/Categories	**Metadata** component *${Component}*	`//suite/metadata/item[@name="component"]`
	Requirement ID	**[Tags]** fid-*yyy*	`//suite/test/tags/tag[@text="fid-*yyy*"]`
	Robot File	**[Tags]** robotfile-*zzz*	`//suite/test/tags/tag[@text="robotfile-*zzz*"]`
Test Case Execution Record (TCER)	Owner	provided `user` in cli	
	Team Area	**Metadata** team-area *${Team_Area}*	`//suite/metadata/item[@name="team-area"]`
	Test Plan	Interaction URL to provided `testplan` in cli	
	Test Case	Interaction URL to provided testcase ID: - Testcase ID which provided in [Tags]: tcid-*xxx* - Generated testcase ID if argument `-createmissing` is used in cli	`//suite/test/tags/tag[@text="tcid-*xxx*"]`
	Test Environment	**Metadata** project *${Environment}*	`//suite/metadata/item[@name="project"]`
Test Result	Owner	provided `user` in cli	
	Tested By	provided `user` in cli - userid must be used (value as username Metadata: tester does not work now)	
	Team Area	**Metadata** team-area *${Team_Area}*	`//suite/metadata/item[@name="team-area"]`
	Actual Result	Test case result (PASSED, FAILED, UNKNOWN)	`//suite/test/status/@status`
	Host Name	**Metadata** machine *%{COMPUTERNAME}*	`//suite/metadata/item[@name="machine"]`
	Test Plan	Interaction URL to provided `testplan` in cli	
	Test Case	Interaction URL to provided testcase ID: - Testcase ID which provided in **[Tags]**: tcid-*xxx* - Generated testcase ID if `-createmissing` is used	`//suite/test/tags/tag[@text="tcid-*xxx*"]`
	Test Case Execution Record	Interaction URL to TCER ID	
	Build	**Metadata** version_sw *${Build}*	`//suite/metadata/item[@name="version_sw"]`
	Start Time	Test case start time	`//suite/test/status/@starttime`
	End Time	Test case end time	`//suite/test/status/@endtime`
	Total Run Time	Calculated from start and end time	
	Test Log	Test case message log	`//suite/test/status/@text`

CHAPTER

TWO

ROBOTRESULTS2RQM PACKAGE

2.1 Module contents

class RobotResults2RQM.CRQM.**CRQMClient**(*user*, *password*, *project*, *host*)

Bases: object

CRQMClient class uses RQM REST APIs to get, create and update resources (testplan, testcase, test result, ...) on RQM - Rational Quality Manager

Resoure type mapping:

- buildrecord: Build Record
- configuration: Test Environment
- testplan: Test Plan
- testsuite: Test Suite
- suiteexecutionrecord: Test Suite Execution Record (TSER)
- testsuitelog: Test Suite Log
- testcase: Test Case
- executionworkitem: Test Execution Record (TCER)
- executionresult: Execution Result

```
NAMESPACES = {'ns1': 'http://schema.ibm.com/vega/2008/', 'ns10':
'http://open-services.net/ns/core#', 'ns11': 'http://open-services.net/ns/qm#',
'ns12': 'http://jazz.net/xmlns/prod/jazz/rqm/process/1.0/', 'ns13':
'http://www.w3.org/2002/07/owl#', 'ns14':
'http://jazz.net/xmlns/alm/qm/qmadapter/v0.1', 'ns15':
'http://jazz.net/xmlns/alm/qm/qmadapter/task/v0.1', 'ns16':
'http://jazz.net/xmlns/alm/qm/v0.1/executionresult/v0.1', 'ns17':
'http://jazz.net/xmlns/alm/qm/v0.1/catalog/v0.1', 'ns18':
'http://jazz.net/xmlns/alm/qm/v0.1/tsl/v0.1/', 'ns2':
'http://jazz.net/xmlns/alm/qm/v0.1/', 'ns20':
'http://jazz.net/xmlns/alm/qm/styleinfo/v0.1/', 'ns21':
'http://www.w3.org/1999/XSL/Transform', 'ns3': 'http://purl.org/dc/elements/1.1/',
'ns4': 'http://jazz.net/xmlns/prod/jazz/process/0.6/', 'ns5':
'http://jazz.net/xmlns/alm/v0.1/', 'ns6': 'http://purl.org/dc/terms/', 'ns7':
'http://www.w3.org/1999/02/22-rdf-syntax-ns#', 'ns8':
'http://jazz.net/xmlns/alm/qm/v0.1/testscript/v0.1/', 'ns9':
'http://jazz.net/xmlns/alm/qm/v0.1/executionworkitem/v0.1'}
```

```
RESULT_STATES = ['paused', 'inprogress', 'notrun', 'passed', 'incomplete',
'inconclusive', 'part_blocked', 'failed', 'error', 'blocked', 'perm_failed',
'deferred']
```

addTeamAreaNode(*root*, *sTeam*)

Append *team-area* node which contains URL to given team-area into xml template

Note: *team-area* information is case-casesensitive

Args: root : xml root object.

sTeam : team name to be added.

Returns: root : xml root object with addition *team-area* node.

config(*plan_id*, *build_name=None*, *config_name=None*, *createmissing=False*, *updatetestcase=False*, *suite_id=None*)

Configure RQMClient with testplan ID, build, configuration, createmissing, ...

- Verify the existence of provided testplan ID.
- Verify the existences of provided build and configuration names before creating new ones.

Args: plan_id : testplan ID of RQM project for importing result(s).

build_name (optional) [the *Build Record* for linking result(s).] Set it to *None* if not be used, the empty name '' may lead to error.

config_name (optional) [the *Test Environment* for linking result(s).] Set it to *None* if not be used, the empty name '' may lead to error.

createmissing (optional) [in case this argument is set to *True*,] the testcase without *tcid* information will be created on RQM.

updatetestcase (optional) [in case this argument is set to *True*,] the information of testcase on RQM will be updated bases on robot testfile.

suite_id (optional) : testsuite ID of RQM project for importing result(s).

Returns: None.

createBuildRecord(*sBuildSWVersion*, *forceCreate=False*)

Create new build record.

Note: Tool will check if build record is already existing or not (both on RQM and current execution).

Args: sBuildSWVersion : build version - *Build Record* name.

forceCreate (optional) : if True, force to create new build record without existing verification.

Returns:

returnObj: a response dictionary which contains status, ID, status_code and error message.

{ 'success' [False, 'id': None, 'message': '',] 'status_code': '' }.

createBuildRecordTemplate(*buildName*)

Return build record template from provided build name

Args: buildName : *Build Record* name.

Returns: xml template as string.

`createConfiguration`(*sConfigurationName*, *forceCreate=False*)

Create new configuration - test environment.

Note: Tool will check if configuration is already existing or not (both on RQM and current execution).

Args: sConfigurationName : configuration - *Test Environment* name.

forceCreate (optional) : if True, force to create new Test Environment without existing verification.

Returns:

returnObj: a response dictionary which contains status, ID, status_code and error message. { 'success' : False, 'id': None, 'message': '', 'status_code': '' }

`createConfigurationTemplate`(*confName*)

Return configuration - Test Environment template from provided configuration name

Args: buildName : configuration - *Test Environment* name.

Returns: xml template as string.

`createExecutionResultTemplate`(*testcaseID*, *testcaseName*, *testplanID*, *TCERID*, *resultState*, *startTime=''*, *endTime=''*, *duration=''*, *testPC=''*, *testBy=''*, *lastlog=''*, *buildrecordID=''*, *sTeam=''*, *sOwnerID=''*)

Return testcase execution result template from provided information

Args: testcaseID : testcase ID.

testcaseName : testcase name.

testplanID : testplan ID for linking.

TCERID : testcase execution record (TCER) ID for linking.

resultState : testcase result status.

startTime : testcase start time.

endTime (optional) : testcase end time.

duration (optional) : testcase duration.

testPC (optional) : test PC which executed testcase.

testBy (optional) : user ID who executed testcase.

lastlog (optional) : traceback information (for Failed testcase).

buildrecordID (optional) : *Build Record* ID for linking.

sTeam (optional) : team name for linking.

sOwnerID (optional) : user ID of testcase owner.

Returns: xml template as string.

`createResource`(*resourceType*, *content*)

Create new resource with provided data from template by POST method.

Args: resourceType : resource type.

content: xml template as string.

Returns:

returnObj: a response dictionary which contains status, ID, status_code and error message. { 'success' : False, 'id': None, 'message': '', 'status_code': '' }

`createTCERTemplate`(*testcaseID*, *testcaseName*, *testplanID*, *confID=''*, *sTeam=''*, *sOwnerID=''*)

Return testcase execution record template from provided information

Args: testcaseID : testcase ID.

testcaseName : testcase name.

testplanID : testplan ID for linking.

confID (optional) : configuration - *Test Environment* for linking.

sTeam (optional) : team name for linking.

sOwnerID (optional) : user ID of testcase owner.

Returns: xml template as string.

`createTSERTemplate`(*testsuiteID*, *testsuiteName*, *testplanID*, *confID=''*, *sOwnerID=''*)

Return testsuite execution record (TSER) template from provided configuration name

Args: testsuiteID : testsuite ID.

testsuiteName : testsuite name.

testplanID : testplan ID for linking.

confID (optional) : configuration - *Test Environment* ID for linking.

sOwnerID (optional) : user ID of testsuite owner.

Returns: xml template as string.

`createTestcaseTemplate`(*testcaseName*, *sDescription=''*, *sComponent=''*, *sFID=''*, *sTeam=''*, *sRobotFile=''*, *sTestType=''*, *sASIL=''*, *sOwnerID=''*, *sTCtemplate=None*)

Return testcase template from provided information.

Args: testcaseName : testcase name.

sDescription (optional) : testcase description.

sComponent (optional) : component which testcase is belong to.

sFID (optional) : function ID(requirement ID) for linking.

sTeam (optional) : team name for linking.

sRobotFile (optional) : link to robot file on source control.

sTestType (optional) : test type information.

sASIL (optional) : ASIL information.

sOwnerID (optional) : user ID of testcase owner.

sTCtemplate (optional) : existing testcase template as xml string. If not provided, template file under *RQM_templates* is used as default.

Returns: xml template as string.

`createTestsuiteResultTemplate`(*testsuiteID*, *testsuiteName*, *TSERID*, *lTCER*, *lTCResults*, *startTime=''*, *endTime=''*, *duration=''*, *sOwnerID=''*)

Return testsuite execution result template from provided configuration name

Args: testsuiteID : testsuite ID.

testsuiteName : testsuite name.

TSERID : testsuite execution record (TSER) ID for linking.

lTCER : list of testcase execution records (TCER) for linking.

lTCResults : list of testcase results for linking.

startTime (optional) : testsuite start time.

endTime (optional) : testsuite end time.

duration (optional) : testsuite duration.

sOwnerID (optional) : user ID of testsuite owner.

Returns: xml template as string.

`disconnect()`

Disconnect from RQM

`getAllBuildRecords()`

Get all available build records of project on RQM and store them into *dBuildVersion* property.

`getAllByResource(`*resourceType*`)`

Return all entries of provided resource by GET method.

Note: This method will try to fetch all entries in all pages of resource.

Args: resourrceType : the RQM resource type.

Returns: dReturn : a dictionary which contains response status, message and data.

`getAllConfigurations()`

Get all available configurations of project on RQM and store them into *dConfiguation* property.

`getAllTeamAreas()`

Get all available team-areas of project on RQM and store them into *dTeamAreas* property.

Example: { 'teamA' : '{host}/qm/process/project-areas/{project-id}/team-areas/{teamA-id}, 'teamB' : '{host}/qm/process/project-areas/{project-id}/team-areas/{teamB-id} }

`getResourceByID(`*resourceType*, *id*`)`

Return data of provided resource and ID by GET method

Args: resourrceType : the RQM resource type.

id : ID of resource.

Returns: res : response data of GET request.

`integrationURL(`*resourceType*, *id=None*, *forceinternalID=False*`)`

Return interaction URL of provided reource and ID.

Note: ID can be internalID (contains only digits) or externalID.

Args: resourceType : the RQM resource type (e.g: "testplan", "testcase", ...)

id (optional) [ID of given resource.] If given: the specified url to resource ID is returned. If *None*: the url to resource type (to get all entity) is returned.

forceinternalID (optional) : force to return the url of resource as internal ID.

Returns: integrationURL : interaction URL of provided reource and ID.

`linkListTestcase2Testplan(`*testplanID*, *lTestcases=None*`)`

Link list of test cases to provided testplan ID.

Args: testplanID : testplan ID to link given testcase(s).

lTestcases [list of testcase(s) to be linked with given testplan.] If None (as default), *lTestcaseIDs* property will be used as list of testcase.

Returns:

res [response object which contains status and error message.] { 'success' : False, 'message': '' }

`linkListTestcase2Testsuite`(*testsuiteID*, *lTestcases=None*)

Link list of test cases to provided testsuite ID

Args: testsuiteID : testsuite ID to link given testcase(s).

lTestcases [list of testcase(s) to be linked with given testsuite.] If None (as default), *lTestcaseIDs* property will be used as list of testcase.

Returns:

res [response object which contains status and error message.] { 'success' : False, 'message': '' }

`login`()

Log in RQM by provided user & password.

Note: When the authentication is successful, the JSESSIONID from cookies will be stored as header for later POST method.

Returns: True if successful, False otherwise.

`updateResourceByID`(*resourceType*, *id*, *content*)

Update data of provided resource and ID by PUT method.

Args: resourceType : resource type.

id : resource id.

content : xml template as string.

Returns: res : response object from PUT request.

`userURL`(*userID*)

Return interaction URL of provided userID

Args: userID : the user ID

Returns: userURL : the interaction URL of provided userID

`verifyProjectName`()

Verify the project name by searching it in *project-areas* XML response.

Note:

The found project ID will be stored and:

- required for *team-areas* request (project name cannot be used)
- used for all later request urls instead of project name

Returns:

- True if the authentication is successful.
- False if the authentication is failed.

`webIDfromGeneratedID`(*resourrceType*, *generateID*)

Return web ID (ns2:webId) from generate ID by get resource data from RQM.

Note:

- This method is only used for generated *testcase*, *executionworkitem* and *executionresult*.
- *buildrecord* and *configuration* does not have *ns2:webId* in response data.

Args: resourrceType : the RQM resource type.

generateID : the Slug ID which is returned in *Content-Location* from POST response.

Returns: webID : web ID (number).

webIDfromResponse(*response*, *tagID='rqm:resultId'*)

Get internal ID (number) from response of POST method.

Note: Only *executionresult* has response text. Other resources has only response header.

Args: response : the response from POST method.

tagID : tag name which contains ID information.

Returns: resultId : internal ID (as number).

RobotResults2RQM.CRQM.**get_xml_tree**(*file_name*, *bdtd_validation=True*)

Parse xml object from file.

Args: file_name : path to file or file-like object. bdtd_validation : if True, validate against a DTD referenced by the document.

Returns: oTree : xml etree object

class RobotResults2RQM.robot2rqm.**Logger**

Bases: object

Logger class for logging message.

color_error = '\x1b[31m\x1b[1m'

color_normal = '\x1b[37m\x1b[22m'

color_reset = '\x1b[0m\x1b[39m\x1b[49m'

color_warn = '\x1b[33m\x1b[1m'

classmethod config(*output_console=True*, *output_logfile=None*, *indent=0*, *dryrun=False*)

Configure Logger class.

Args: output_console : write message to console output.

output_logfile : path to log file output.

indent : offset indent.

dryrun : if set, a prefix as 'dryrun' is added for all messages.

Returns: None.

dryrun = False

classmethod log(*msg=''*, *color=None*, *indent=0*)

Write log message to console/file output.

Args: msg : message to write to output.

color : color style for the message.

indent : offset indent.

Returns: None.

classmethod log_error(*msg*, *fatal_error=False*)

Write error message to console/file output.

Args: msg : message to write to output.

fatal_error : if set, tool will terminate after logging error message.

Returns: None.

classmethod log_warning(*msg*)

Write warning message to console/file output.

Args: msg : message to write to output.

Returns: None.

output_console = True

output_logfile = None

prefix_all = ''

prefix_error = 'ERROR: '

prefix_fatalerror = 'FATAL ERROR: '

prefix_warn = 'WARN: '

RobotResults2RQM.robot2rqm.**RobotResults2RQM**(*args=None*)

Import robot results from output.xml to RQM - IBM Rational Quality Manager.

Flow to import Robot results to RQM:

1. Process provided arguments from command line
2. Login Rational Quality Management (RQM)
3. Parse Robot results
4. Import results into RQM
5. Link all executed testcases to provided testplan/testsuite ID

Args:

args [Argument parser object:]

- *outputfile* : path to the output file or directory with output files to be imported.
- *host* : RQM host url.
- *project* : RQM project name.
- *user* : user for RQM login.
- *password* : user password for RQM login.
- *testplan* : RQM testplan ID.
- *recursive* : if True, then the path is searched recursively for log files to be imported.
- *createmissing* : if True, then all testcases without fcid are created when importing.
- *updatetestcase* : if True, then testcases information on RQM will be updated bases on robot testfile.
- *dryrun* : if True, then just check the RQM authentication and show what would be done.

Returns: None.

RobotResults2RQM.robot2rqm.**convert_to_datetime**(*time*)

Convert time string to datetime.

Args: time : string of time.

Returns: dt : datetime object

RobotResults2RQM.robot2rqm.**get_from_tags**(*lTags*, *reInfo*)

Extract testcase information from tags.

Example: TCID-xxxx, FID-xxxx, ...

Args: lTags : list of tag information.

reInfo : regex to get the expectated info (ID) from tag info.

Returns: lInfo : list of expected information (ID)

RobotResults2RQM.robot2rqm.**process_metadata**(*metadata*, *default_metadata={'author': '', 'category': '', 'component': 'unknown', 'configfile': '', 'description': '', 'machine': '', 'project': 'ROBFW', 'tags': '', 'team-area': '', 'tester': '', 'testtool': '', 'version_hw': '', 'version_sw': '', 'version_test': ''}*)

Extract metadata from suite result bases on DEFAULT_METADATA

Args: metadata : Robot metadata object.

default_metadata: initial Metadata information for updating.

Returns: dMetadata : dictionary of Metadata information.

RobotResults2RQM.robot2rqm.**process_suite**(*RQMClient*, *suite*)

process robot suite for importing to RQM.

Args: RQMClient : RQMClient object.

suite : Robot suite object.

Returns: None.

RobotResults2RQM.robot2rqm.**process_suite_metadata**(*suite*, *default_metadata={'author': '', 'category': '', 'component': 'unknown', 'configfile': '', 'description': '', 'machine': '', 'project': 'ROBFW', 'tags': '', 'team-area': '', 'tester': '', 'testtool': '', 'version_hw': '', 'version_sw': '', 'version_test': ''}*)

Try to find metadata information from all suite levels.

Note: Metadata at top suite level has a highest priority.

Args: suite : Robot suite object.

default_metadata: initial Metadata information for updating.

Returns: dMetadata : dictionary of Metadata information.

RobotResults2RQM.robot2rqm.**process_test**(*RQMClient*, *test*)

process robot test for importing to RQM.

Args: RQMClient : RQMClient object.

test : Robot test object.

Returns: None.

PYTHON MODULE INDEX

r

INDEX

L

M

N

O

P

R

U

V

W