

# **RobotLog2RQM**

**v. 1.5.0**

Tran Duy Ngoan

28.10.2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Description</b>	<b>2</b>
2.1	Get Robot Framework XML result	2
2.2	Tool features	2
2.2.1	Usage	3
2.2.2	Basic import feature	3
2.2.3	Verify the given arguments	4
2.2.4	Import multiple *.xml result files	4
2.2.5	Create missing Test Case on RQM	4
2.2.6	Update existing Test Case on RQM	5
2.2.7	Test suite feature	5
2.2.8	Project with Configuration Management (CM)	5
2.2.9	Naming convention of generated resources	6
2.3	Robot Framework Test Case Information on RQM:	7
2.4	Additional Tool: RQMTool	9
2.4.1	Purpose	9
2.4.2	Usage	9
2.4.3	Examples	9
2.4.4	Notes	10
<b>3</b>	<b>CRQM.py</b>	<b>11</b>
3.1	Function: get_xml_tree	11
3.2	Class: Identifier	11
3.3	Class: CRQMClient	11
3.3.1	Method: login	12
3.3.2	Method: verifyProjectName	12
3.3.3	Method: disconnect	12
3.3.4	Method: config	12
3.3.5	Method: userURL	13
3.3.6	Method: integrationURL	13
3.3.7	Method: webIDfromResponse	14
3.3.8	Method: webIDfromGeneratedID	14
3.3.9	Method: getResourceByID	15
3.3.10	Method: getAllByResource	15
3.3.11	Method: getAllBuildRecords	15
3.3.12	Method: getAllConfigurations	15

3.3.13	Method: <code>getAllTeamAreas</code>	16
3.3.14	Method: <code>getTestsFromTestplan</code>	16
3.3.15	Method: <code>addTeamAreaNode</code>	16
3.3.16	Method: <code>createTestcaseTemplate</code>	17
3.3.17	Method: <code>createTCERTemplate</code>	18
3.3.18	Method: <code>createExecutionResultTemplate</code>	18
3.3.19	Method: <code>createBuildRecordTemplate</code>	19
3.3.20	Method: <code>createConfigurationTemplate</code>	20
3.3.21	Method: <code>createTSERTemplate</code>	20
3.3.22	Method: <code>createTestsuiteResultTemplate</code>	20
3.3.23	Method: <code>createTestsuiteTemplate</code>	21
3.3.24	Method: <code>createResource</code>	22
3.3.25	Method: <code>createBuildRecord</code>	22
3.3.26	Method: <code>createConfiguration</code>	23
3.3.27	Method: <code>updateResourceByID</code>	23
3.3.28	Method: <code>linkListTestcase2Testplan</code>	24
3.3.29	Method: <code>linkListTestcase2Testsuite</code>	24
3.3.30	Method: <code>addTestsuite2Testplan</code>	25
<b>4</b>	<b><code>logger.py</code></b>	<b>26</b>
4.1	Class: <code>Logger</code>	26
4.1.1	Method: <code>config</code>	26
4.1.2	Method: <code>log</code>	26
4.1.3	Method: <code>log_warning</code>	27
4.1.4	Method: <code>log_error</code>	27
<b>5</b>	<b><code>robotlog2rqm.py</code></b>	<b>28</b>
5.1	Function: <code>get_from_tags</code>	28
5.2	Function: <code>convert_to_datetime</code>	28
5.3	Function: <code>process_config_file</code>	29
5.4	Function: <code>is_valid_config</code>	29
5.5	Function: <code>process_suite_metadata</code>	29
5.6	Function: <code>process_metadata</code>	30
5.7	Function: <code>process_suite</code>	30
5.8	Function: <code>process_test</code>	31
5.9	Function: <code>RobotLog2RQM</code>	31
<b>6</b>	<b><code>rqmtool.py</code></b>	<b>33</b>
6.1	Function: <code>write_json_file</code>	33
6.2	Function: <code>write_csv_file</code>	33
6.3	Function: <code>write_output_file</code>	34
6.4	Function: <code>RQMTool</code>	34
<b>7</b>	<b>Appendix</b>	<b>35</b>
<b>8</b>	<b>History</b>	<b>36</b>

# Chapter 1

## Introduction

**RobotLog2RQM** facilitates the import of Robot Framework result file(s) in **\*.xml** format into IBM® Rational® Quality Manager (RQM) resources.

It introduces the **CRQM Class**, offering the capability to interact with various RQM resources, including test plans, test cases, builds, and more, through the [RqmAPI](#) to:

- retrieve RQM resources: obtain resources by a given ID or retrieve all available entities of a specified resource type.
- update RQM resources: modify existing resources by providing the relevant ID.
- create new RQM resources: generate new resources using predefined templates located in the [RQM.templates](#) folder.

So that **RobotLog2RQM** tool can:

- create all required resources (*Test Case Excution Record*, *Test Case Execution Result*, ...) for new test cases on RQM.
- link all test cases to provided test plan.
- add new test results for existing test cases on RQM.
- update existing test cases on RQM.

## Chapter 2

# Description

### 2.1 Get Robot Framework XML result

In order to manage test cases and their results Rational Quality Manager (RQM), certain traceable information, such as version, test case ID, component, etc., is required.

This enables the **RobotLog2RQM** tool to associate the imported test results with specific elements (Test Case) or link them to other entities (Build Record, Test Environment) in RQM.

These information should be provided in **Metadata** (for the whole testsuite/execution info: version, build, ...) and **[Tags]** information (for specific test case info: component, test case ID, requirement ID, ...) of Robot Framework test case. Then when executing Robot Framework test case(s), the generated Robot Framework result file (default is *output.xml*) will contain all of them and ready for importing.

Sample Robot Framework test case with the necessary information for importing to RQM:

```
*** Settings ***
Metadata    project      ROBFW          # Test Environment on RQM for linking
Metadata    version_sw   SW_VERSION_0.1 # Build Record on RQM for linking
Metadata    machine      ${COMPUTERNAME} # Hostname attribute in RQM Test Case Result
Metadata    component     Import_Tools   # Component attribute in RQM Test Case
Metadata    team-area     Internet Team RQM # team-area (case-sensitive) on RQM for linking

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

Listing 2.1: Sample Robot Framework test case

#### Hint



In case you are using RobotFramework AIO, above highlighted **Metadata** definitions are not required because they have been handled by **RobotFramework.TestsuitesManagement** library within **Suite Setup**.

### 2.2 Tool features

After getting the Robot Framework *\*.xml* result file(s), you can use the **RobotLog2RQM** tool to import them into RQM.

Its usage and features are described as following sections.

### 2.2.1 Usage

Use below command to get tools's usage:

```
RobotLog2DB -h
```

The tool's usage should be showed as below:

```
usage: RobotLog2RQM (RobotXMLResult to RQM importer) [-h] [-v] [--testsuite TESTSUITE] ↵
        ↵ [--recursive] [--createmissing] [--updatetestcase] [--dryrun] [--stream STREAM] ↵
        ↵ [--baseline BASELINE]
                                resultxmlfile host project user ↵
        ↵ password testplan

RobotLog2RQM imports XML result files (default: output.xml) generated by the Robot ↵
        ↵ Framework into an IBM Rational Quality Manager.

positional arguments:
  resultxmlfile          absolute or relative path to the xml result file or directory of ↵
        ↵ result files to be imported.
  host                   RQM host url.
  project                project on RQM.
  user                   user for RQM login.
  password               password for RQM login.
  testplan               testplan ID for this execution.

optional arguments:
  -h, --help             show this help message and exit
  -v, --version           Version of the RobotLog2RQM importer.
  --testsuite TESTSUITE  testsuite ID for this execution. If 'new', then create a new testsuite for this execution.
  --recursive            if set, then the path is searched recursively for log files to ↵
        ↵ be imported.
  --createmissing         if set, then all testcases without tcid are created when importing.
  --updatetestcase       if set, then testcase information on RQM will be updated bases ↵
        ↵ on robot testfile.
  --dryrun               if set, then verify all input arguments (includes RQM ↵
        ↵ authentication) and show what would be done.
  --stream STREAM        project stream. Note, requires Configuration Management (CM) to ↵
        ↵ be enabled for the project area.
  --baseline BASELINE    project baseline. Note, requires Configuration Management (CM), ↵
        ↵ or Baselines Only to be enabled for the project area.
```

As above instruction, **RobotLog2RQM** tool requires 5 positional arguments consists of:

- The Robot Framework result file/folder `resultxmlfile`
- The RQM authentication `host` , `project` , `user` , `password`
- The RQM `testplan` ID which will contains all importing test results

### 2.2.2 Basic import feature

Use the below command for the simple import the *output.xml* file to RQM project **ROBFW-AIO** which is hosted at <https://sample-rqm-host.com>

```
RobotLog2RQM output.xml https://sample-rqm-host.com ROBFW-AIO test_user test_pw 720
```

When command is executed, the tool will process with following steps:

- Login the RQM server with the provided credential, then verify the existences of given `project` , `testplan` on RQM
- Create RQM **Build Record** and **Test Environment** (if already provided in Robot Framework test case and not existing on RQM)

- Create new RQM **Test Case Execution Record - TCER** (if it is not existing) bases on test case ID (defined `TCID=xxx` in `[Tags]` of Robot Framework Test Cases) and `testplan` ID
- Create new RQM **Test Case Execution Result** which contents the detail and result state of Robot Framework test case
- Link all test case(s) to provided `testplan`

### 2.2.3 Verify the given arguments

In case you just want to verify whether the given `*.xml` file/folder and the RQM authentication in arguments are corrected or not, the optional argument `--dryrun` will help to do it.

In the dryrun mode, **RobotLog2RQM** will not create any resources on RQM, it just verify:

- The given Robot Framework result file/folder is valid or not
- The given RQM authentication is correct or not
- The given RQM project and testplan are existing or not

### 2.2.4 Import multiple \*.xml result files

**RobotLog2RQM** accepts the first argument `resultxmlfile` can be a single file or the folder that contains multiple Robot Framework result files.

When the folder is used, **RobotLog2RQM** will only search for `*.xml` file under given directory and exclude any file within subdirectories as default.

In case you have result file(s) under the subdirectory of given folder and want these result files will also be imported, the optional argument `--recursive` should be used when executing **RobotLog2RQM** command.

When `--recursive` argument is set, **RobotLog2RQM** will walk through the given directory and its subdirectories to discover and collect all available `*.xml` for importing.

For example: your result folder has a structure as below:

```
logFolder
|_____ result_1.xml
|_____ result_2.xml
|_____ subFolder_1
|       |_____ result_sub_1.xml
|       |_____ subSubFolder
|           |_____ result_sub_sub_1.xml
|_____ subFolder_2
|           |_____ result_sub_2.xml
```

- Without `--recursive` : only `result_1.xml` and `result_2.xml` are found for importing.
- With `--recursive` : all `result_1.xml`, `result_2.xml`, `result_sub_1.xml`, `result_sub_2.xml` and `result_sub_sub_1.xml` will be imported.

### 2.2.5 Create missing Test Case on RQM

By default, **RobotLog2RQM** tool will not touch (create, update) any RQM Test Case.

If the `TCID=xxx` information is missing in `[Tags]` section of Robot Framework Test Case, an error message will be raised for that specific importing Test Case, and the tool will proceed with the next Test Cases accordingly

```
ERROR: There is no 'tcid' information for importing test 'Testcase 01'.
```

So that, in order to import those missing `TCID=xxx` Test Cases, the optional arguments `--createmissing` should be provided in the **RobotLog2RQM** arguments.

When `--createmissing` is used, **RobotLog2RQM** will help to create RQM Test Cases bases on the defined information in Robot Framework Test Cases. It obtains the new Test Case ID and uses it for linking to related RQM resources **TCER**, **Test Case Execution Result** as basic feature.

The new IDs for the created Test Cases are also displayed in the execution log. You can copy these IDs and update the `TCID-xxx` information in Robot Framework Test Cases for the next execution. This information will then be available in the generated `*.xml` result file for importing.

Please refer [Robot Framework Test Case Information on RQM](#) section for details on how the defined information in Robot Framework Test Cases is reflected in RQM.

## 2.2.6 Update existing Test Case on RQM

In case the Test Case is existing on RQM, but you want to update its attribute(s) such as **Component**, **Description**, ... the optional argument `--updatetestcase` should be used.

**RobotLog2RQM** will update RQM Test Case resource bases on the defined information in Robot Framework Test Case before creating its result.

Please refer [Robot Framework Test Case Information on RQM](#) section for details on how the defined information in Robot Framework Test Cases is reflected in RQM.

## 2.2.7 Test suite feature

In addition to managing the imported test results by testplan, the **RobotLog2RQM** tool also supports the use of testsuite with the optional argument `--testsuite TESTSUITE`.

This argument provides two main features:

- **Importing to an existing testsuite:** If you specify the ID of an existing testsuite using the `--testsuite TESTSUITE` argument, the tool will import the test results directly into that testsuite.

Example: testsuite with id `1234` is already existing

```
RobotLog2RQM output.xml https://sample-rqm-host.com ROBFW-AIO test_user test_pw 720 ↵
    ↪ --testsuite 1234
```

- **Creating a new testsuite and then importing:** If you set the value to `new` using the `--testsuite new` argument, the tool will create a new testsuite and then import the test results into this newly created testsuite.

Example:

```
RobotLog2RQM output.xml https://sample-rqm-host.com ROBFW-AIO test_user test_pw 720 ↵
    ↪ --testsuite new
```

When using the `--testsuite` argument, the tool will follow these additional steps besides the basic steps which is described in above [Basic import feature](#) section.

- Verify the existence of given `testsuite`. Or create the new **Test Suite** if `new` value is used.
- Create a **Test Suite Execution Record** for given (or newly created) `testsuite`.
- Create a **Test Suite Result** which contains all **Test Case Execution Result**.
- Link all test cases to the given (or newly created) `testsuite`.
- Add that `testsuite` to the given `testplan`.

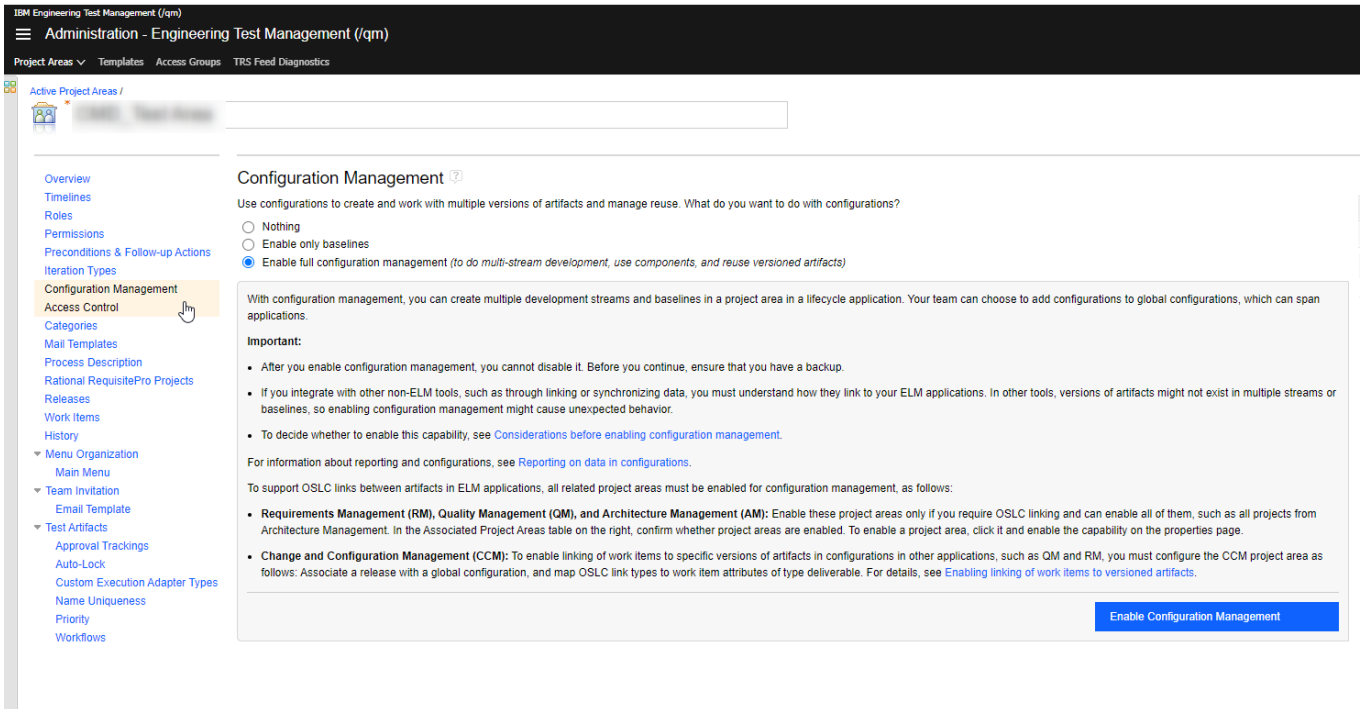
## 2.2.8 Project with Configuration Management (CM)

When Configuration Management (CM) is enabled for the project area, it is necessary to specify the project stream or baseline when executing import tool to interact with the proper stream/baseline resources.

Otherwise, the tool will not be able to correctly associate the imported test results.

Configuration Management (CM) is only enabled by Project Administrator as below figure:





The following arguments are used to provide this information:

- `--stream STREAM` : specifies the project `STREAM` .

A project stream represents a branch of work within a project area. When CM is enabled, it is crucial to provide this option to correctly associate the imported test results with the appropriate stream.

Example:

```
RobotLog2RQM output.xml https://sample-rqm-host.com ROBFW-AIO test_user test_pw ↔
↪ 720 --stream "Develop_stream"
```

- `--baseline BASELINE` : specifies the project `BASELINE` .

A baseline captures the state of project artifacts at a particular point in time. When CM or Baselines Only is enabled, this option is required to ensure that the test results are linked to the correct baseline.

Example:

```
RobotLog2RQM output.xml https://sample-rqm-host.com ROBFW-AIO test_user test_pw ↔
↪ 720 --baseline "Baseline_v1"
```

#### Note



Only one option ( `--stream` or `--baseline` ) should be used at a time. If both options are provided, the `--stream` option will take precedence.

### 2.2.9 Naming convention of generated resources

This feature provides flexibility in defining how new resources such as test execution results, test cases, and test suite are named, allowing users to customize the naming of resources generated during the import of test results to RQM.

The optional argument `--naming_config` is used to provide the user's definitions. It takes a path to a JSON file as value. The JSON file specifies the format and structure of the names for different types of resources using predefined placeholders `{resource_type}` .

Supported placeholders:

- `{testplan}`: Name of the test plan.
- `{build}`: Name of the build record.
- `{environment}`: Name of the test environment.
- `{testsuite}`: Name of the test suite (if `--testsuite` is used).

- **{testcase}**: Name of the test case, only available for test case relevant resources: **testcase**, **tcer** and **testresult**

Example JSON file ( `naming_convention.json` ):

```
{
  "testcase": "{testcase} (imported by RobotLog2RQM tool)", // customize test case name
  "tcer": "{testcase}_{environment}", // Name TCER with environment as suffix
  "testresult": "{testcase}_{build}", // Name Test result with build as suffix
  "testsuite": "{testplan}_{testsuite}", // Name testsuite with test plan as prefix
  "tser": "{testsuite}_{environment}",
  "suiteresult": "{testsuite}_{environment}_{build}"
}
```

Example Usage:

```
RobotLog2RQM output.xml https://sample-rqm-host.com ROBFW-AIO test_user test_pw 720 ↵
↵ --naming_config path/to/naming_convention.json
```

#### Default Behavior:

If the `--naming_config` argument is not used, the tool will use default naming convention as defined by the **RobotLog2RQM** tool:

```
{
  "testcase"      : "{testcase}",
  "tcer"          : "TCER: {testcase}",
  "testresult"    : "Execution result: {testcase}",
  "testsuite"     : "{testsuite}",
  "tser"          : "TSER: {testsuite}",
  "suiteresult"   : "Testsuite result: {testsuite}"
}
```

## 2.3 Robot Framework Test Case Information on RQM:

For more detail about the mapping between the defined information from Robot Framework Test Case to Robot Framework result (*output.xml*) file and their reflections on RQM WebApp, please refer below mapping table:

**Table: Mapping RQM - Robot**

RQM data		Robot Framework	
Resource	Attribute/ Field	Testsuite/Testcase	Output.xml
Build Record	Title	<code>Metadata version_sw Build</code>	<code>//suite/metadata/item[@name="version_sw"]</code>
Test Environment	Title	<code>Metadata project Environment</code>	<code>//suite/metadata/item[@name="project"]</code>
Test Case	ID	<code>[Tags] tcid-xxx</code>	<code>//suite/test/tags/tag[@text="tcid-xxx"]</code>
	Name	tesname	<code>//suite/test/@name</code>
	Team Area	<code>Metadata team-area Team_Area</code>	<code>//suite/metadata/item[@name="team-area"]</code>
	Description	test doc - <code>[Documentation]</code>	<code>//suite/test/doc/@text</code>
	Owner	provided <code>user</code> in cli	
	Component/ Categories	<code>Metadata component Component</code>	<code>//suite/metadata/item[@name="component"]</code>
	Requirement ID	<code>[Tags] fid-yyy</code>	<code>//suite/test/tags/tag[@text="fid-yyy"]</code>
	Robot File	<code>[Tags] robotfile-zzz</code>	<code>//suite/test/tags/tag[@text="robotfile-zzz"]</code>
Test Case Execution Record (TCER)	Owner	provided <code>user</code> in cli	
	Team Area	<code>Metadata team-area Team_Area</code>	<code>//suite/metadata/item[@name="team-area"]</code>
	Test Plan	Interaction URL to provided <code>testplan</code> in cli	
	Test Case	Interaction URL to provided test case ID: provided tcid in <code>[Tags]: tcid-xxx</code> or generated tcid when using <code>-createmissing</code>	<code>//suite/test/tags/tag[@text="tcid-xxx"]</code>
	Test Environment	<code>Metadata project Environment</code>	<code>//suite/metadata/item[@name="project"]</code>
Test Result	Owner	provided <code>user</code> in cli	
	Tested By	provided <code>user</code> in cli - userid must be used	
	Team Area	<code>Metadata team-area Team_Area</code>	<code>//suite/metadata/item[@name="team-area"]</code>
	Actual Result	Test case result (PASSED, FAILED, UNKNOWN)	<code>//suite/test/status/@status</code>
	Host Name	<code>Metadata machine ↔ ↔ %{COMPUTERNAME}</code>	<code>//suite/metadata/item[@name="machine"]</code>
	Test Plan	Interaction URL to provided <code>testplan</code> in cli	
	Test Case	Interaction URL to provided test case ID: provided tcid in <code>[Tags]: tcid-xxx</code> or generated tcid when using <code>-createmissing</code>	<code>//suite/test/tags/tag[@text="tcid-xxx"]</code>
	Test Case Execution Record	Interaction URL to TCER ID	
	Build	<code>Metadata version_sw Build</code>	<code>//suite/metadata/item[@name="version_sw"]</code>
	Start Time	Test case start time	<code>//suite/test/status/@starttime</code>
	End Time	Test case end time	<code>//suite/test/status/@endtime</code>
	Total Run Time	Calculated from start and end time	
	Result Details	Test case message log	<code>//suite/test/status/@text</code>

## 2.4 Additional Tool: RQMTTool

The **RobotLog2RQM** package now includes the **\*\*RQMTTool\*\*** submodule, which provides a standalone CLI for fetching test cases and test suites from IBM Rational Quality Manager (RQM) test plans.

### 2.4.1 Purpose

RQMTTool enables:

- Fetching test cases and/or test suites from a given RQM test plan.
- Exporting fetched artifacts as CSV or JSON.
- Performing dry-run validation of inputs and RQM connectivity without fetching.

### 2.4.2 Usage

The RQMTTool can be executed as a Python module:

```
python -m RobotLog2RQM.rqmtool --host <RQM_SERVER_URL> --project <PROJECT_AREA> \
    --user <USERNAME> --password <PASSWORD> \
    --testplan <TESTPLAN_ID> [--types <artifact_types>] \
    [--format <csv|json>] [--output-dir <DIR>] \
    [--basename <BASENAME>] [--dryrun]
```

Arguments description:

- **--host (required)**: Base URL of the RQM server.
- **--project (required)**: Name of the RQM project area.
- **--user (required)**: RQM username.
- **--password (required)**: RQM password.
- **--testplan (required)**: ID of the test plan to fetch.
- **--types (optional)**: Comma-separated list of artifact types to fetch ( `testcase` , `testsuite` ). Default: both.
- **--format (optional)**: Output format: `csv` or `json` . Default: `csv` .
- **--output-dir (optional)**: Directory to save exported files. Default: current directory.
- **--basename (optional)**: Base name for output files. Default: `testplan_export` .
- **--dryrun (optional)**: Validate inputs and RQM connection without performing fetch.

### 2.4.3 Examples

Fetch all test cases and test suites from test plan ID 720 and export to CSV:

```
python -m RobotLog2RQM.rqmtool --host https://sample-rqm-host.com \
    --project ROBFW-AIO \
    --user test_user \
    --password test_pw \
    --testplan 720 \
    --types testcase,testsuite \
    --format csv
```

Perform a dry-run to check connection and arguments:

```
python -m RobotLog2RQM.rqmtool --host https://sample-rqm-host.com \
    --project ROBFW-AIO \
    --user test_user \
    --password test_pw \
    --testplan 720 \
    --dryrun
```

#### 2.4.4 Notes

- CSV export automatically generates filenames including test plan ID.
- JSON export contains only selected artifact types.
- Dry-run mode does not fetch or save any data, only validates arguments and RQM access.

##### Tip



RQMTTool can be used independently to fetch test cases from a testplan and even execute them selectively, or it can be combined with the main **RobotLog2RQM** import tool to retrieve test case information before importing Robot Framework results into RQM.

## Chapter 3

# CRQM.py

### 3.1 Function: get\_xml\_tree

Parse xml object from file.

**Arguments:**

- `file_name`  
/ *Condition*: required / *Type*: str /  
Path to file or file-like object.
- `bdttd.validation`  
/ *Condition*: optional / *Type*: bool /  
If True, validate against a DTD referenced by the document.

**Returns:**

- `oTree`  
/ *Type*: `lxml.etree.ElementTree` object /  
The xml etree object.

### 3.2 Class: Identifier

*Imported by:*

```
from RobotLog2RQM.CRQM import Identifier
```

Identifier class used to identify RQM resource with name and id

### 3.3 Class: CRQMClient

*Imported by:*

```
from RobotLog2RQM.CRQM import CRQMClient
```

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

Resource 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

### 3.3.1 Method: login

Log in RQM by provided user & password.

**Arguments:**

*(no arguments)*

**Returns:**

- bSuccess  
/ *Type*: bool /  
Indicates if the computation of the method login was successful or not.

### 3.3.2 Method: verifyProjectName

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

**Arguments:**

*(no arguments)*

**Returns:**

- bSuccess  
/ *Type*: bool /  
Indicates if the computation of the method verifyProjectName was successful or not.

### 3.3.3 Method: disconnect

Disconnect from RQM.

**Arguments:**

*(no arguments)*

**Returns:**

*(no returns)*

### 3.3.4 Method: config

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.

**Arguments:**

- plan\_id  
/ *Condition*: required / *Type*: str /  
Testplan ID of RQM project for importing result(s).

- `build_name`  
*/ Condition: optional / Type: str / Default: None /*  
 The Build Record for linking result(s). Set it to None if not be used, the empty name "" will lead to error.
- `config_name`  
*/ Condition: optional / Type: str / Default: None /*  
 The Test Environment for linking result(s). Set it to None if not be used, the empty name "" may lead to error.
- `createmissing`  
*/ Condition: optional / Type: bool / Default: False /*  
 If True, the testcase without tcid information will be created on RQM.
- `updatetestcase`  
*/ Condition: optional / Type: bool / Default: False /*  
 If True, the information of testcase on RQM will be updated bases on robot testfile.
- `suite_id (optional)`  
*/ Condition: optional / Type: str / Default: None /*  
 Testsuite ID of RQM project for importing result(s).

**Returns:***(no returns)***3.3.5 Method: userURL**

Return interaction URL of provided userID

**Arguments:**

- `userID`  
*/ Condition: required / Type: str /*  
 The user ID.

**Returns:**

- `userURL`  
*/ Type: str /*  
 The interaction URL of provided userID.

**3.3.6 Method: integrationURL**

Return interaction URL of provided resource and ID. The provided ID can be internalID (contains only digits) or externalID.

**Arguments:**

- `resourceType`  
*/ Condition: required / Type: str /*  
 The RQM resource type (e.g: "testplan", "testcase", ...).
- `id`  
*/ Condition: optional / Type: str / Default: None /*  
 The 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`  
/ *Condition*: optional / *Type*: bool / *Default*: False /  
If True, force to return the url of resource as internal ID.

**Returns:**

- `integrationURL`  
/ *Type*: str /  
The interaction URL of provided resource and ID.

**3.3.7 Method: webIDfromResponse**

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

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

**Arguments:**

- `response`  
/ *Condition*: required / *Type*: str /  
The xml response from POST method for parsing ID information.
- `tagID`  
/ *Condition*: optional / *Type*: str / *Default*: 'rqm:resultId' /  
Tag name which contains ID information.

**Returns:**

- `resultId`  
/ *Type*: str /  
The internal ID (as number).

**3.3.8 Method: webIDfromGeneratedID**

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.

**Arguments:**

- `resourceType`  
/ *Condition*: required / *Type*: str /  
The RQM resource type.
- `generateID`  
/ *Condition*: required / *Type*: str /  
The Slug ID which is returned in Content-Location from POST response.

**Returns:**

- `webID`  
/ *Type*: str /  
The web ID (as number).

### 3.3.9 Method: getResourceByID

Return data of provided resource and ID by GET method

**Arguments:**

- resourceType  
/ Condition: required / Type: str /  
The RQM resource type.
- id  
/ Condition: required / Type: str /  
ID of resource.

**Returns:**

- res  
/ Type: Response object /  
Response data of GET request.

### 3.3.10 Method: getAllByResource

Return all entries (in all pages) of provided resource by GET method.

**Arguments:**

- resourceType  
/ Condition: required / Type: str /  
The RQM resource type.

**Returns:**

- dReturn  
/ Type: dict /  
A dictionary which contains response status, message and data.  
Example:

```
{
  'success' : False,
  'message' : '',
  'data'    : {}
}
```

### 3.3.11 Method: getAllBuildRecords

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

**Arguments:**

(no arguments)

**Returns:**

(no returns)

### 3.3.12 Method: getAllConfigurations

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

**Arguments:**

(no arguments)

**Returns:**

(no returns)

### 3.3.13 Method: 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}'
}
```

#### Arguments:

(no arguments)

#### Returns:

(no returns)

### 3.3.14 Method: getTestsFromTestplan

Get all test cases and test suites associated with a given test plan.

#### Arguments:

- testplan\_id  
/ Condition: required / Type: str /  
The RQM test plan to get test artifact(s).
- artifact\_types  
/ Condition: required / Type: list /  
List of artifact types (testcase, testsuite) for fetching.

#### Returns:

- / Type: dict /  
A dictionary containing fetched artifacts:

```
{
  'testcase': [{'id': ..., 'name': ..., 'url': ...}, ...],
  'testsuite': [{'id': ..., 'name': ..., 'url': ...}, ...]
}
```

### 3.3.15 Method: addTeamAreaNode

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

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

#### Arguments:

- root  
/ Condition: required / Type: Element object /  
The xml root object.
- sTeam  
/ Condition: required / Type: str /  
Team name to be added.

#### Returns:

- root  
/ Type: str /  
The xml root object with addition team-area node.

### 3.3.16 Method: createTestcaseTemplate

Return testcase template from provided information.

**Arguments:**

- `testcaseName`  
/ *Condition*: required / *Type*: str /  
Testcase name.
- `sDescription`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
Testcase description.
- `sComponent`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
Component which testcase is belong to.
- `sFID`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
Function ID (requirement ID) for linking.
- `sTeam`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
Team name for linking.
- `sRobotFile`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
Link to robot file on source control.
- `sTestType`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
Test type information.
- `sASIL`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
ASIL information.
- `sOwnerID`  
/ *Condition*: optional / *Type*: str / *Default*: "" /  
User ID of testcase owner.
- `sTCtemplate`  
/ *Condition*: optional / *Type*: str / *Default*: None /  
Existing testcase template as xml string.  
If not provided, template file under RQM\_templates is used as default.

**Returns:**

- `sTCxml`  
/ *Type*: str /  
The xml testcase template as string.

### 3.3.17 Method: createTCERTemplate

Return testcase execution record template from provided information.

**Arguments:**

- `testcaseID`  
/ *Condition*: required / *Type*: str /  
Testcase ID for linking.
- `testcaseName`  
/ *Condition*: required / *Type*: str /  
Testcase name.
- `testplanID`  
/ *Condition*: required / *Type*: str /  
Testplan ID for linking.
- `confID`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Configuration - Test Environment for linking.
- `sTeam`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Team name for linking.
- `sOwnerID`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
User ID of testcase owner.

**Returns:**

- `sTCERxml`  
/ *Type*: str /  
The xml testcase execution record template as string.

### 3.3.18 Method: createExecutionResultTemplate

Return testcase execution result template from provided information.

**Arguments:**

- `testcaseID`  
/ *Condition*: required / *Type*: str /  
Testcase ID for linking.
- `testcaseName`  
/ *Condition*: required / *Type*: str /  
Testcase name.
- `testplanID`  
/ *Condition*: required / *Type*: str /  
Testplan ID for linking.
- `TCERID`  
/ *Condition*: required / *Type*: str /  
Testcase execution record (TCER) ID for linking.

- `resultState`  
/ *Condition*: required / *Type*: str /  
Testcase result status.
- `startTime`  
/ *Condition*: required / *Type*: str /  
Testcase start time.
- `endTime`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Testcase end time.
- `duration`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Testcase duration.
- `testPC`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Test PC which executed testcase.
- `testBy`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
User ID who executed testcase.
- `lastlog`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Traceback information (for Failed testcase).
- `buildrecordID`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Build Record ID for linking.
- `sTeam`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Team name for linking.
- `sOwnerID`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
User ID of testcase owner.

**Returns:**

- `sTCResultxml`  
/ *Type*: str /  
The xml testcase result template as string.

**3.3.19 Method: createBuildRecordTemplate**

Return build record template from provided build name.

**Arguments:**

- `buildName`  
/ *Condition*: required / *Type*: str /  
Build Record name.

**Returns:**

- `sBuildxml`  
/ *Type*: str /  
The xml build template as string.

### 3.3.20 Method: createConfigurationTemplate

Return configuration - Test Environment template from provided configuration name.

**Arguments:**

- buildName  
/ *Condition*: required / *Type*: str /  
Configuration - Test Environment name.

**Returns:**

- sEnvironmentxml  
/ *Type*: str /  
The xml test environment template as string.

### 3.3.21 Method: createTSERTemplate

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

**Arguments:**

- testsuiteID  
/ *Condition*: required / *Type*: str /  
Testsuite ID.
- testsuiteName  
/ *Condition*: required / *Type*: str /  
Testsuite name.
- testplanID  
/ *Condition*: required / *Type*: str /  
Testplan ID for linking.
- confID  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Configuration - Test Environment ID for linking.
- sOwnerID  
/ *Condition*: optional / *Type*: str / *Default*: " /  
User ID of testsuite owner.

**Returns:**

- sTSxml  
/ *Type*: str /  
The xml testsuite template as string.

### 3.3.22 Method: createTestsuiteResultTemplate

Return testsuite execution result template from provided configuration name.

**Arguments:**

- testsuiteID  
/ *Condition*: required / *Type*: str /  
Testsuite ID.

- `testsuiteName`  
/ *Condition*: required / *Type*: str /  
Testsuite name.
- `TSERID`  
/ *Condition*: required / *Type*: str /  
Testsuite execution record (TSER) ID for linking.
- `lTCER`  
/ *Condition*: required / *Type*: str /  
List of testcase execution records (TCER) for linking.
- `lTCResults`  
/ *Condition*: required / *Type*: str /  
List of testcase results for linking.
- `startTime`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Testsuite start time.
- `endTime`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Testsuite end time.
- `duration`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Testsuite duration.
- `sOwnerID`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
User ID of testsuite owner.

**Returns:**

- `sTSResultxml`  
/ *Type*: str /  
The xml testsuite result template as string.

**3.3.23 Method: createTestsuiteTemplate**

Return testcase template from provided information.

**Arguments:**

- `testsuiteName`  
/ *Condition*: required / *Type*: str /  
Testsuite name.
- `sDescription`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Testsuite description.
- `sOwnerID`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
User ID of testsuite owner.



- `sTStemplate`  
/ *Condition*: optional / *Type*: str / *Default*: None /  
Existing testsuite template as xml string.  
If not provided, template file under `RQM_templates` is used as default.

**Returns:**

- `STSxml`  
/ *Type*: str /  
The xml testsuite template as string.

**3.3.24 Method: createResource**

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

**Arguments:**

- `resourceType`  
/ *Condition*: required / *Type*: str /  
Resource type.
- `content`  
/ *Condition*: required / *Type*: str /  
The xml template as string.

**Returns:**

- `returnObj`  
/ *Type*: dict /  
A dictionary reponse which contains status, ID, status\_code and error message.  
Example:

```
{
  'success' : False,
  'id': None,
  'message': '',
  'status_code': ''
}
```

**3.3.25 Method: createBuildRecord**

Create new build record.

**Arguments:**

- `sBuildSWVersion`  
/ *Condition*: required / *Type*: str /  
Build version - Build Record name.
- `forceCreate`  
/ *Condition*: optional / *Type*: bool / *Default*: False /  
If True, force to create new build record without existing verification.

**Returns:**

- returnObj  
/ *Type*: dict /  
A dictionary reponse which contains status, ID, status\_code and error message.  
Example:

```
{
  'success' : False,
  'id': None,
  'message': '',
  'status_code': ''
}
```

### 3.3.26 Method: createConfiguration

Create new configuration - test environment.

#### Arguments:

- sConfigurationName  
/ *Condition*: required / *Type*: str /  
Configuration - Test Environment name.
- forceCreate  
/ *Condition*: optional / *Type*: str / *Default*: False /  
If True, force to create new Test Environment without existing verification.

#### Returns:

- returnObj  
/ *Type*: dict /  
A dictionary reponse which contains status, ID, status\_code and error message.  
Example:

```
{
  'success' : False,
  'id': None,
  'message': '',
  'status_code': ''
}
```

### 3.3.27 Method: updateResourceByID

Update data of provided resource and ID by PUT method.

#### Arguments:

- resourceType  
/ *Condition*: required / *Type*: str /  
Resource type.
- id  
/ *Condition*: required / *Type*: str /  
Resource id.
- content  
/ *Condition*: required / *Type*: str /  
The xml template as string.

**Returns:**

- `res`  
/ *Type*: Response object /  
Response object from PUT request.

**3.3.28 Method: linkListTestcase2Testplan**

Link list of test cases to provided testplan ID.

**Arguments:**

- `testplanID`  
/ *Condition*: required / *Type*: str /  
Testplan ID to link given testcase(s).
- `lTestcases`  
/ *Condition*: optional / *Type*: list / *Default*: None /  
List of testcase(s) to be linked with given testplan.  
If not provide, `lTestcaseIDs` property will be used as list of testcase.

**Returns:**

- `returnObj`  
/ *Type*: dict /  
Response dictionary which contains status and error message.  
Example:

```
{
  'success' : False,
  'message': ''
}
```

**3.3.29 Method: linkListTestcase2Testsuite**

Link list of test cases to provided testsuite ID

**Arguments:**

- `testsuiteID`  
/ *Condition*: required / *Type*: str /  
Testsuite ID to link given testcase(s).
- `lTestcases`  
/ *Condition*: optional / *Type*: list / *Default*: None /  
List of testcase(s) to be linked with given testplan.  
If not provide, `lTestcaseIDs` property will be used as list of testcase.

**Returns:**

- `returnObj`  
/ *Type*: dict /  
Response dictionary which contains status and error message.  
Example:

```
{
  'success' : False,
  'message': ''
}
```

### 3.3.30 Method: addTestsuite2Testplan

Add testsuite ID to provided testplan ID

#### Arguments:

- testplanID  
/ *Condition*: required / *Type*: str /  
Testplan ID to link given testsuite ID.
- testsuiteID  
/ *Condition*: optional / *Type*: str / *Default*: None /  
Testsuite to be linked with given testplan.  
If not provide, testsuite.id value will be used as id of testsuite.

#### Returns:

- returnObj  
/ *Type*: dict /  
Response dictionary which contains status and error message.  
Example:

```
{
  'success' : False,
  'message': ''
}
```

# Chapter 4

## logger.py

### 4.1 Class: Logger

*Imported by:*

```
from RobotLog2RQM.logger import Logger
```

Logger class for logging message.

#### 4.1.1 Method: config

Configure Logger class.

**Arguments:**

- `output_console`  
/ *Condition*: optional / *Type*: bool / *Default*: True /  
Write message to console output.
- `output_logfile`  
/ *Condition*: optional / *Type*: str / *Default*: None /  
Path to log file output.
- `dryrun`  
/ *Condition*: optional / *Type*: bool / *Default*: True /  
If set, a prefix as 'dryrun' is added for all messages.

**Returns:**

(no returns)

#### 4.1.2 Method: log

Write log message to console/file output.

**Arguments:**

- `msg`  
/ *Condition*: optional / *Type*: str / *Default*: '' /  
Message which is written to output.
- `color`  
/ *Condition*: optional / *Type*: str / *Default*: None /  
Color style for the message.

- `indent`  
/ *Condition*: optional / *Type*: int / *Default*: 0 /  
Offset indent.

**Returns:**

(no returns)

### 4.1.3 Method: `log_warning`

Write warning message to console/file output.

**Arguments:**

- `msg`  
/ *Condition*: required / *Type*: str /  
Warning message which is written to output.
- `indent`  
/ *Condition*: optional / *Type*: int / *Default*: 0 /  
Offset indent.

**Returns:**

(no returns)

### 4.1.4 Method: `log_error`

Write error message to console/file output.

- `msg`  
/ *Condition*: required / *Type*: str /  
Error message which is written to output.
- `fatal_error`  
/ *Condition*: optional / *Type*: bool / *Default*: False /  
If set, tool will terminate after logging error message.
- `indent`  
/ *Condition*: optional / *Type*: int / *Default*: 0 /  
Offset indent.

**Returns:**

(no returns)

## Chapter 5

# robotlog2rqm.py

### 5.1 Function: get\_from\_tags

Extract testcase information from tags.

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

**Arguments:**

- lTags  
/ *Condition:* required / *Type:* list /  
List of tag information.
- reInfo  
/ *Condition:* required / *Type:* str /  
Regex to get the expected info (ID) from tag info.

**Returns:**

- lInfo  
/ *Type:* list /  
List of expected information (ID)

### 5.2 Function: convert\_to\_datetime

Convert time string to datetime.

**Arguments:**

- time  
/ *Condition:* required / *Type:* str /  
String of time.

**Returns:**

- dt  
/ *Type:* datetime object/  
Datetime object.

## 5.3 Function: process\_config\_file

Parse and validate content of configuration file

### Arguments:

- path\_file  
/ Condition: required / Type: str /  
Path to the configuration json file.

### Returns:

- dConfig  
/ Type: dict /  
Content of json file.

## 5.4 Function: is\_valid\_config

Validate the json configuration base on given schema.

Default schema supports below information:

```
NAMING_CONVENTION_SCHEMA = {
    "testcase"      : str,
    "tcer"          : str,
    "testresult"    : str,
    "testsuite"     : str,
    "tser"          : str,
    "suiteresult"   : str
}
```

### Arguments:

- dConfig  
/ Condition: required / Type: dict /  
Json configuration object to be verified.
- dSchema  
/ Condition: optional / Type: dict / Default: CONFIG\_SCHEMA /  
Schema for the validation.
- bExitOnFail  
/ Condition: optional / Type: bool / Default: True /  
If True, exit tool in case the validation is fail.

### Returns:

- bValid  
/ Type: bool /  
True if the given json configuration data is valid.

## 5.5 Function: process\_suite\_metadata

Try to find metadata information from all suite levels.

Metadata at top suite level has a highest priority.

### Arguments:



- `suite`  
/ *Condition*: required / *Type*: TestSuite object /  
Robot suite object.
- `default_metadata`  
/ *Condition*: optional / *Type*: dict / *Default*: DEFAULT\_METADATA /  
Initial Metadata information for updating.

**Returns:**

- `dMetadata`  
/ *Type*: dict /  
Dictionary of Metadata information.

## 5.6 Function: process\_metadata

Extract metadata from suite result bases on DEFAULT\_METADATA.

**Arguments:**

- `metadata`  
/ *Condition*: required / *Type*: dict /  
Robot metadata object.
- `default_metadata`  
/ *Condition*: optional / *Type*: dict / *Default*: DEFAULT\_METADATA /  
Initial Metadata information for updating.

**Returns:**

- `dMetadata`  
/ *Type*: dict /  
Dictionary of Metadata information.

## 5.7 Function: process\_suite

Process robot suite for importing to RQM.

**Arguments:**

- `RQMClient`  
/ *Condition*: required / *Type*: RQMClient object/  
RQMClient object.
- `suite`  
/ *Condition*: required / *Type*: TestSuite object/  
Robot suite object.
- `log_indent`  
/ *Condition*: optional / *Type*: int / *Default*: 0 /  
Indent for logging message.

**Returns:**

(no returns)

## 5.8 Function: process\_test

Process robot test for importing to RQM.

### Arguments:

- `RQMClient`  
/ *Condition*: required / *Type*: RQMClient object/  
RQMClient object.
- `test`  
/ *Condition*: required / *Type*: TestCase object/  
Robot test object.
- `log_indent`  
/ *Condition*: optional / *Type*: int / *Default*: 0 /  
Indent for logging message.

### Returns:

(no returns)

## 5.9 Function: RobotLog2RQM

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

### Arguments:

\* args

/ *Condition*: required / *Type*: ArgumentParser object /

Argument parser object which contains:

- `resultxmlfile` : path to the xml result file or directory of result 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.
- `testsuite` : testsuite ID for this execution. If 'new', then create a new testsuite for this execution.
- `recursive` : if True, then the path is searched recursively for log files to be imported.
- `createmissing` : if True, then all testcases without tcid are created when importing.
- `updatetestcase` : if True, then testcases information on RQM will be updated bases on robot testfile.
- `naming_config` : configuration json file for naming conventions when creating RQM resources.
- `dryrun` : if True, then verify all input arguments (includes RQM authentication) and show what would be done.

- stream : project stream. Note, requires Configuration Management (CM) to be enabled for the project area.
- baseline : project baseline. Note, requires Configuration Management (CM), or Baselines Only to be enabled for the project area.

**Returns:**

*(no returns)*

## Chapter 6

# rqmtool.py

### 6.1 Function: write\_json\_file

Write data to a JSON file.

**Arguments:**

- `file_name`  
/ *Condition*: required / *Type*: str /  
Path of the JSON file to write.
- `data`  
/ *Condition*: required / *Type*: dict /  
Data to export.

**Returns:**

(no returns)

### 6.2 Function: write\_csv\_file

Write data to a CSV file for a specific artifact type.

**Arguments:**

- `file_name`  
/ *Condition*: required / *Type*: str /  
Path of the CSV file to write.
- `data`  
/ *Condition*: required / *Type*: dict /  
Data dictionary containing artifacts.
- `artifact_type`  
/ *Condition*: required / *Type*: str /  
Artifact type (testcase or testsuite) to export.

**Returns:**

(no returns)

## 6.3 Function: write\_output\_file

Write data to output files (JSON or CSV) according to specified options.

### Arguments:

- data  
/ *Condition*: required / *Type*: dict /  
Data dictionary containing artifacts.
- output\_dir  
/ *Condition*: optional / *Type*: str /  
Directory to save output files. Default is current directory.
- basename  
/ *Condition*: optional / *Type*: str /  
Base name for output files. Default is "testplan\_export".
- extension  
/ *Condition*: optional / *Type*: str /  
Output format: json or csv. Default is csv.
- artifact\_types  
/ *Condition*: optional / *Type*: list /  
Artifact types to export. Default is all supported types.

### Returns:

(no returns)

## 6.4 Function: RQMTool

Main entry point for RQMTool CLI.

### Arguments:

(no arguments)

### Returns:

(no returns)

## Chapter 7

# Appendix

About this package:

Table 7.1: Package setup

Setup parameter	Value
Name	RobotLog2RQM
Version	1.5.0
Date	28.10.2025
Description	Imports robot result(s) to IBM Rational Quality Manager (RQM)
Package URL	<a href="#">robotframework-robotlog2rqm</a>
Author	Tran Duy Ngoan
Email	<a href="mailto:Ngoan.TranDuy@vn.bosch.com">Ngoan.TranDuy@vn.bosch.com</a>
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

## Chapter 8

# History

<b>0.1.0</b>	07/2022
<i>Initial version</i>	
<b>1.1.1</b>	01.08.2022
<i>Rework repository's document bases on GenPackageDoc</i>	
<b>1.1.2</b>	25.08.2022
<ul style="list-style-type: none"><li>- Correct indent of sourcode's docstring.</li><li>- Update new style for history.</li></ul>	
<b>1.1.3</b>	13.10.2022
<ul style="list-style-type: none"><li>- Fix findings and enhance README and document files</li><li>- Change argument name 'outputfile' to 'resultxmlfile'</li></ul>	
<b>1.1.4</b>	10.11.2022
<i>Rename package to RobotLog2RQM</i>	
<b>1.2.0</b>	09.01.2023
<ul style="list-style-type: none"><li>- Rework optional arguments and improve logging messages</li><li>- Update README and document for publishing pypi</li></ul>	
<b>1.2.1</b>	14.06.2023
<i>Update README: fix links issue and update installation section</i>	
<b>1.2.2</b>	06.03.2024
<i>Fix findings in documentation</i>	
<b>1.2.3</b>	14.03.2024
<i>Add support for basic authentication as an alternative to SSO system</i>	
<b>1.2.4</b>	11.06.2024
<i>Fix issue when test result has SKIP status</i>	
<b>1.3.0</b>	19.06.2024
<i>Add new feature of test suite with new optional argument <code>-testsuite TEST-SUITE</code></i>	
<b>1.3.1</b>	03.07.2024
<i>Add link to Test Environment for TSER when using testsuite feature</i>	
<b>1.4.0</b>	24.07.2024
<i>Add support project with enabled Configuration Management (multiple streams, baselines)</i>	
<b>1.4.1</b>	20.09.2024
<i>Add new feature of defining the naming convention of generated resources</i>	
<b>1.4.2</b>	18.02.2025

<i>Add missing build information when creating the testsuite result</i>	
<b>1.5.0</b>	28.10.2025
<i>Added RQMTool: additional support for fetching RQM test cases and test suites from a given test plan</i>	