
The RobotResults2DB tool

Tran Duy Ngoan (RBVH/ECM1)

Mar 08, 2022

CONTENTS:

1	RobotResults2DB tool's Documentation	1
1.1	Introduction:	1
1.2	RobotFramework Testcase Settings:	1
1.3	Sample RobotFramework Testcase:	2
1.4	Display on WebApp:	3
1.5	Notes:	4
2	RobotResults2DB package	7
2.1	Module contents	7
	Python Module Index	17

ROBOTRESULTS2DB TOOL'S DOCUMENTATION

1.1 Introduction:

RobotResults2DB tool helps to import robot *output.xml* result file(s) to WebApp's database for presenting an overview about the execution and detail of each test result.

In order to display the Robotframework test results on **TestResultWebApp** Dashboard properly, Robot testcase need to give some required information for management such as project/variant, software version, component, ...

Therefore, **Metadata** and **[Tags]** are used to provide that information to *output.xml* result which is used for importing data to WebApp.

1.2 RobotFramework Testcase Settings:

For the whole test execution:

- Project/Variant (can be overwritten by argument *-variant* or *-config* of **RobotResults2DB** tool when importing):

Metadata	project	\${Project_name}
----------	---------	------------------

- Versions (can be overwritten by argument *-versions* or *-config* of **RobotResults2DB** tool when importing):

Metadata	version_hw	\${Software_version}
Metadata	version_hw	\${Hardware_version}
Metadata	version_test	\${Test_version}

For the Suite/File information:

- Description/Documentation:

Documentation	\${Suite_description}
---------------	-----------------------

- Author:

Metadata	author	\${Author_name}
----------	--------	-----------------

- Component (can be overwritten by argument *-config* of **[RobotResults2DB]** tool when importing):

Metadata	component	\${Component_name}
----------	-----------	--------------------

- Test Tool - framework and python version, e.g **Robot Framework 3.2rc2 (Python 3.9.0 on win32)**:

```
Metadata    testtool    ${Test_tool}
```

- Test Machine:

```
Metadata    machine    %{COMPUTERNAME}
```

- Tester:

```
Metadata    tester     %{USER}
```

For test case information:

- Issue ID:

```
[Tags]      ISSUE-${ISSUE_ID}
```

- Testcase ID:

```
[Tags]      TCID-${TC_ID}
```

- Requirement ID:

```
[Tags]      FID-${REQ_ID}
```

1.3 Sample RobotFramework Testcase:

For test case management, we need some traceable 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 necessary information for importing to RQM:

```
*** Settings ***
# Test execution level
Metadata    project      ROFW           # Project/Variant
Metadata    version_sw   SW_VERSION_0.1   # Software version
Metadata    version_hw   HW_VERSION_0.1   # Hardware version
Metadata    version_test  TEST_VERSION_0.1 # Test version

# File/Suite level
Documentation  This is description for robot test file
Metadata      author      Tran Duy Ngoan (RBVH/ECM1)
Metadata      component    Import_Tools
Metadata      testtool     Robot Framework 3.2rc2 (Python 3.9.0 on win32)
Metadata      machine      %{COMPUTERNAME}
Metadata      tester       %{USER}

*** Test Cases ***
Testcase 01
  [Tags]    ISSUE-001    TCID-1001    FID-112    FID-111
  Log       This is Testcase 01
```

(continues on next page)

(continued from previous page)

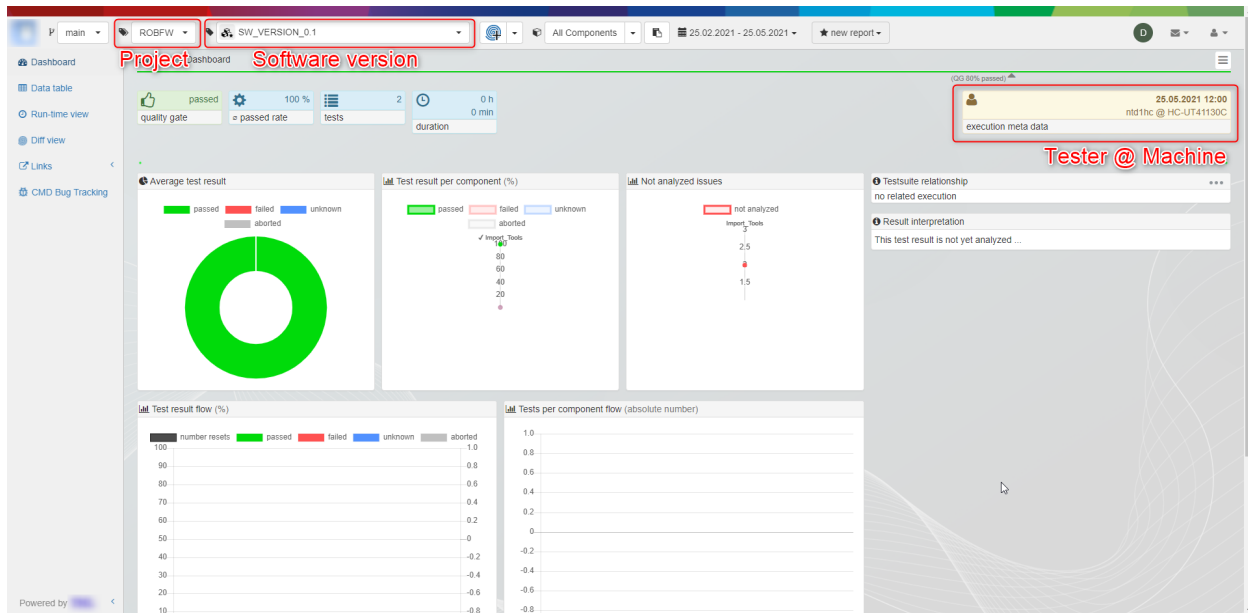
Testcase 02
[Tags] ISSUE-RTC-003 TCID-1002 FID-113
Log This is Testcase 01

1.4 Display on WebApp:

When the *output.xml* file(s) is importing successfully to database, the result for that execution will be available on [TestResultWebApp](#).

Above settings in robot testcase will be reflect on **Dashboard** (General view) and **Data table** (Detailed view) as below figures:

Execution result metadata:



Suite/File metadata and Testcase information:

The screenshot displays the RobotResults2DB tool interface. On the left is a sidebar with navigation links: Dashboard, Data table, Run-time view, Diff view, Links, and CMD Bug Tracking. The main area shows a list of test results with columns for Result, Name, Component, toid, fid, and issue. A search bar is at the top right. The test results are displayed in a table format with a search bar and filters. The test results are as follows:

Result	Name	Component	toid	fid	issue
[Test started: 25.05.2021 12:00:12]					
[Test tool configuration: {}]					
[Test file header]					
[Test execution]					
[Test requirements]					
[Test done: 25.05.2021 12:00:19]					
Testcase 01		Import_Tools	1001	111;112	001
Testcase 02		Import_Tools	1002	113	RTC-003

Annotations on the screenshot:

- Test tool information**: Points to the [Test tool configuration: {}] section.
- Suite/File Author & Description**: Points to the [Test file header] section.
- Tester and Machine**: Points to the [Test execution] section.
- Component**: Points to the Component column in the table.
- Testcase info**: Points to the toid, fid, and issue columns in the table.

1.5 Notes:

When above settings is missing, that leads to the missing information in the *output.xml*.

Some required fields for management will be set to default value when importing with **RobotResults2DB** tool:

- *Project*: will be set to default value **ROBFW** if not defined.
- *Software version*: will be set to execution time `%Y%m%d_%H%M%S` as default value.
- *Component*: will be set to default value **unknown** if not defined.

But, you can provide them as command arguments when executing the **RobotResults2DB** tool with below optional arguments (refer its [usage](#)):

- `--variant VARIANT`

To specify the *Project/Variant* information.

- `--versions VERSIONS`

To specify the *Software version* information.

- `--config CONFIG`

Provide a configuration json file *CONFIG* which helps:

- To configure the *Project/Variant*, *Software version* information (lower priority than above commandline arguments)
- To create a mapping between testcase folder and *Component* information which is display on **TestResultWebApp**.

Sample configuration json file:

```
{
  "component": {
```

(continues on next page)

(continued from previous page)

```
"cli"      : "robot/cli",
"core"     : "robot/core",
"external" : "robot/external",
"keywords" : "robot/keywords",
"libdoc"   : "robot/libdoc",
"output"   : "robot/output",
"parsing"  : "robot/parsing",
"reboot"   : "robot/reboot",
"rpa"      : "robot/rpa",
"running"  : "robot/running",
"std_lib"  : "robot/standard_libraries",
"tags"     : "robot/tags",
"test_lib" : "robot/test_libraries",
"testdoc"  : "robot/testdoc",
"tidy"     : "robot/tidy",
"variables" : "robot/variables"
},
"version_sw" : "Atest",
"variant"    : "ROBFW"
}
```


ROBOTRESULTS2DB PACKAGE

2.1 Module contents

class RobotResults2DB.CDataBase.CDataBase(*args, **kwargs)

Bases: object

CDataBase class play a role as mysqlclient and provide methods to interact with TestResultWebApp's database.

arGetCategories()

Get existing categories.

Returns: arCategories : list of existing categories.

bExistingResultID(sResultID)

Verify the given test result UUID is existing in *tbl_result* table or not.

Args: sResultID : test result UUID to be verified.

Returns: bExisting : True if test result UUID is already existing.

cleanAllTables()

Delete all table data. Please be careful before calling this method.

connect(host=None, user=None, passwd=None, database=None, charset='utf8', use_unicode=True)

Connect to the database with provided authentication and db info.

Args: host: URL which is hosted the TestResultWebApp's database.

user : user name for database authentication.

passwd : user password for database authentication.

database : database name.

charset (optional): the connection character set.

use_unicode (optional): If True, CHAR and VARCHAR and TEXT columns are returned as
Unicode strings, using the configured character set.

Returns: None.

disconnect()

Disconnect from TestResultWebApp's database.

nCreateNewFile(_tbl_file_name, _tbl_file_tester_account, _tbl_file_tester_machine, _tbl_file_time_start,
_tbl_file_time_end, _tbl_test_result_id, _tbl_file_origin='ROBFW')

Create new file entry in *tbl_file* table.

Args: `_tbl_file_name` : file name information.

`_tbl_file_tester_account` : tester account information.

`_tbl_file_tester_machine` : test machine information.

`_tbl_file_time_start` : test file start time.

`_tbl_file_time_end` : test file end time.

`_tbl_test_result_id` : UUID of test result for linking to *tbl_result* table.

`_tbl_file_origin` [origin (test framework) of test file.] Default is “ROBFW”

Returns: `iInsertedID`: ID of new entry.

nCreateNewSingleTestCase(*_tbl_case_name, _tbl_case_issue, _tbl_case_tcid, _tbl_case_fid,*
_tbl_case_testnumber, _tbl_case_repeatcount, _tbl_case_component,
_tbl_case_time_start, _tbl_case_result_main, _tbl_case_result_state,
_tbl_case_result_return, _tbl_case_counter_resets, _tbl_case_lastlog,
_tbl_test_result_id, _tbl_file_id)

Create single testcase entry in *tbl_case* table immediately.

Args: `_tbl_case_name` : test case name.

`_tbl_case_issue` : test case issue ID.

`_tbl_case_tcid` : test case ID (used for testmanagement tool).

`_tbl_case_fid` : test case requirement (function) ID.

`_tbl_case_testnumber` : order of test case in file.

`_tbl_case_repeatcount` : test case repeatition count.

`_tbl_case_component` : component which test case is belong to.

`_tbl_case_time_start` : test case start time.

`_tbl_case_result_main` : test case main result.

`_tbl_case_result_state` : test case completion state.

`_tbl_case_result_return` : test case result code (as integer).

`_tbl_case_counter_resets` : counter of target reset within test case execution.

`_tbl_case_lastlog` : traceback information when test case is failed.

`_tbl_test_result_id` : UUID of test result for linking to file in *tbl_result* table.

`_tbl_file_id` : test file ID for linking to file in *tbl_file* table.

Returns: `iInsertedID`: ID of new entry.

nCreateNewTestCase(*_tbl_case_name, _tbl_case_issue, _tbl_case_tcid, _tbl_case_fid,*
_tbl_case_testnumber, _tbl_case_repeatcount, _tbl_case_component,
_tbl_case_time_start, _tbl_case_result_main, _tbl_case_result_state,
_tbl_case_result_return, _tbl_case_counter_resets, _tbl_case_lastlog,
_tbl_test_result_id, _tbl_file_id)

Create bulk of test case entries: new test case are buffered and inserted as bulk.

Once `__NUM_BUFFERD_ELEMENTS_FOR_EXECUTEMANY` is reached, the creation query is executed.

Args: `_tbl_case_name` : test case name.

`_tbl_case_issue` : test case issue ID.

`_tbl_case_tcid` : test case ID (used for testmanagement tool).

`_tbl_case_fid` : test case requirement (function) ID.

`_tbl_case_testnumber` : order of test case in file.

`_tbl_case_repeatcount` : test case repetition count.

`_tbl_case_component` : component which test case is belong to.

`_tbl_case_time_start` : test case start time.

`_tbl_case_result_main` : test case main result.

`_tbl_case_result_state` : test case completion state.

`_tbl_case_result_return` : test case result code (as integer).

`_tbl_case_counter_resets` : counter of target reset within test case execution.

`_tbl_case_lastlog` : traceback information when test case is failed.

`_tbl_test_result_id` : UUID of test result for linking to file in *tbl_result* table.

`_tbl_file_id` : test file ID for linking to file in *tbl_file* table.

Returns: None.

sCreateNewTestResult(*_tbl_prj_project*, *_tbl_prj_variant*, *_tbl_prj_branch*, *_tbl_test_result_id*,
_tbl_result_interpretation, *_tbl_result_time_start*, *_tbl_result_time_end*,
_tbl_result_version_sw_target, *_tbl_result_version_sw_test*,
_tbl_result_version_target, *_tbl_result_jenkinsurl*,
_tbl_result_reporting_qualitygate)

Creates a new test result in *tbl_result*. This is the main table which is linked to all other data by means of *test_result_id*.

Args: `_tbl_prj_project` : project information.

`_tbl_prj_variant` : variant information.

`_tbl_prj_branch` : branch information.

`_tbl_test_result_id` : UUID of test result.

`_tbl_result_interpretation` : result interpretation.

`_tbl_result_time_start` : test result start time.

`_tbl_result_time_end` : test result end time.

`_tbl_result_version_sw_target` : software version information.

`_tbl_result_version_sw_test` : test version information.

`_tbl_result_version_target` : hardware version information.

`_tbl_result_jenkinsurl` : jenkinsurl in case test result is executed by jenkins.

`_tbl_result_reporting_qualitygate` : qualitygate information for reporting.

Returns: `_tbl_test_result_id`: *test_result_id*.

sGetLatestFileID(*sResultID*)

Get latest file ID from *tbl_file* table.

Args: sResultID : UUID of test result to get the latest file ID.

Returns: sFileID : file ID.

vCreateAbortReason(*_tbl_test_result_id, _tbl_abort_reason, _tbl_abort_message*)

Create abort reason entry.

Args: *_tbl_test_result_id* : UUID of test result.

_tbl_abort_reason : abort reason.

_tbl_abort_message : detail message of abort.

Returns: None.

vCreateCCRdata(*_tbl_test_case_id, ICCRdata*)

Create CCR data per test case.

Args: *_tbl_test_case_id* : test case ID.

ICCRdata : list of CCR data.

Returns: None.

vCreateNewHeader(*_tbl_file_id, _tbl_header_testtoolconfiguration_testtoolname, _tbl_header_testtoolconfiguration_testtoolversionstring, _tbl_header_testtoolconfiguration_projectname, _tbl_header_testtoolconfiguration_logfileencoding, _tbl_header_testtoolconfiguration_pythonversion, _tbl_header_testtoolconfiguration_testfile, _tbl_header_testtoolconfiguration_logfilepath, _tbl_header_testtoolconfiguration_logfilemode, _tbl_header_testtoolconfiguration_ctrlfilepath, _tbl_header_testtoolconfiguration_configfile, _tbl_header_testtoolconfiguration_confname, _tbl_header_testfileheader_author, _tbl_header_testfileheader_project, _tbl_header_testfileheader_testfiledate, _tbl_header_testfileheader_version_major, _tbl_header_testfileheader_version_minor, _tbl_header_testfileheader_version_patch, _tbl_header_testfileheader_keyword, _tbl_header_testfileheader_shortdescription, _tbl_header_testexecution_useraccount, _tbl_header_testexecution_computername, _tbl_header_testrequirements_documentmanagement, _tbl_header_testrequirements_testenvironment, _tbl_header_testbenchconfig_name, _tbl_header_testbenchconfig_data, _tbl_header_preprocessor_filter, _tbl_header_preprocessor_parameters*)

Create a new header entry in *tbl_file_header* table which is linked with the file.

Args: *_tbl_file_id* : file ID information.

_tbl_header_testtoolconfiguration_testtoolname : test tool name.

_tbl_header_testtoolconfiguration_testtoolversionstring : test tool version.

_tbl_header_testtoolconfiguration_projectname : project name.

_tbl_header_testtoolconfiguration_logfileencoding : encoding of logfile.

_tbl_header_testtoolconfiguration_pythonversion : Python version info.

_tbl_header_testtoolconfiguration_testfile : test file name.

_tbl_header_testtoolconfiguration_logfilepath : path to log file.

_tbl_header_testtoolconfiguration_logfilemode : mode of log file.

_tbl_header_testtoolconfiguration_ctrlfilepath : path to control file.
 _tbl_header_testtoolconfiguration_configfile : path to configuration file.
 _tbl_header_testtoolconfiguration_confname : configuration name.
 _tbl_header_testfileheader_author : file author.
 _tbl_header_testfileheader_project : project information.
 _tbl_header_testfileheader_testfiledate : file creation date.
 _tbl_header_testfileheader_version_major : file major version.
 _tbl_header_testfileheader_version_minor : file minor version.
 _tbl_header_testfileheader_version_patch : file patch version.
 _tbl_header_testfileheader_keyword : file keyword.
 _tbl_header_testfileheader_shortdescription : file short description.
 _tbl_header_testexecution_useraccount : tester account who run the execution.
 _tbl_header_testexecution_computername : machine name which is executed on.
 _tbl_header_testrequirements_documentmanagement : requirement management information.
 _tbl_header_testrequirements_testenvironment : requirement environment information.
 _tbl_header_testbenchconfig_name : testbench configuration name.
 _tbl_header_testbenchconfig_data : testbench configuration data.
 _tbl_header_preprocessor_filter : preprocessor filter information.
 _tbl_header_preprocessor_parameters : preprocessor parameters definition.

Returns: None.

vCreateReanimation(*_tbl_test_result_id*, *_tbl_num_of_reanimation*)

Create reanimation entry.

Args: *_tbl_test_result_id* : UUID of test result.

_tbl_num_of_reanimation : counter of target reanimation during execution.

Returns: None.

vCreateTags(*_tbl_test_result_id*, *_tbl_usr_result_tags*)

Create tag entries.

Args: *_tbl_test_result_id* : UUID of test result.

_tbl_usr_result_tags : user tags information.

Returns: None.

vEnableForeignKeyCheck(*enable=True*)

Switch *foreign_key_checks* flag.

vFinishTestResult(*_tbl_test_result_id*)

Finish upload:

- First do bulk insert of rest of test cases if buffer is not empty.
- Then set state to “new report”.

vSetCategory(*_tbl_test_result_id, tbl_result_category_main*)

Create category entry.

Args: *_tbl_test_result_id* : UUID of test result.

tbl_result_category_main : category information.

Returns: None.

vUpdateEvtbl(*sTestResultID*)

Call *update_evtbl* stored procedure to update provided *test_result_id*.

vUpdateEvtbls()

Call *update_evtbls* stored procedure.

vUpdateFileEndTime(*sFileID, sEndtime*)

Update test file end time.

Args: *sFileID* : file ID to be updated.

sEndtime : end time information.

Returns: None.

vUpdateResultEndTime(*sResultID, sEndtime*)

Update test result end time.

Args: *sResultID* : test result UUID to be updated.

sEndtime : end time information.

Returns: None.

vUpdateStartEndTime(*_tbl_test_result_id, _tbl_result_time_start, _tbl_result_time_end*)

Create start-end time entry.

Args: *_tbl_test_result_id* : UUID of test result.

_tbl_result_time_start : result start time.

_tbl_result_time_end : result end time.

Returns: None.

class RobotResults2DB.robot2db.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: '

RobotResults2DB.robot2db.RobotResults2DB(*args=None*)

Import robot results from *output.xml* to TestResultWebApp's database

Flow to import Robot results to database:

1. Process provided arguments from command line
2. Connect to database
3. Parse Robot results
4. Import results into database
5. Disconnect from database

Args:

args [Argument parser object:]

- *outputfile* : path to the output file or directory with output files to be imported.
- *server* : server which hosts the database (IP or URL).
- *user* : user for database login.
- *password* : password for database login.
- *database* : database name.

- *recursive* : if True, then the path is searched recursively for log files to be imported.
- *dryrun* : if True, then just check the RQM authentication and show what would be done.
- *UUID* : UUID used to identify the import and version ID on TestResultWebApp.
- *variant* : variant name to be set for this import.
- *versions* : metadata: Versions (Software;Hardware;Test) to be set for this import.
- *config* : configuration json file for component mapping information.

Returns: None.

`RobotResults2DB.robot2db.format_time(stime)`

Format the given time string to TestResultWebApp's format for importing to db.

Args: *stime* : string of time.

Returns: TestResultWebApp's time format.

`RobotResults2DB.robot2db.get_branch_from_swversion(sw_version)`

Get branch name from software version information.

Convention of branch information in suffix of software version:

- All software version with .0F is the main/feature branch. The leading number is the current year. E.g. 17.0F03
- All software version with .1S, .2S, ... is a stabi branch. The leading number is the year of branching out for stabilization. The number before "S" is the order of branching out in the year.

Args: *sw_version* : software version.

Returns: *branch_name* : branch name.

`RobotResults2DB.robot2db.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 expected info (ID) from tag info.

Returns: *lInfo* : list of expected information (ID)

`RobotResults2DB.robot2db.is_valid_uuid(uuid_to_test, version=4)`

Verify the given UUID is valid or not.

Args: *uuid_to_test* : UUID to be verified.

version (optional): UUID version.

Returns: True if the given UUID is valid.

`RobotResults2DB.robot2db.normalize_path(sPath)`

Normalize path file.

Args: *sPath* : string of path file to be normalized.

Returns: *sNPath* : string of normalized path file.

`RobotResults2DB.robot2db.process_config_file(config_file)`

Parse information from configuration file:

- *component*:

```
{
  "component" : {
    "componentA" : "componentA/path/to/testcase",
    "componentB" : "componentB/path/to/testcase",
    "componentC" : [
      "componentC1/path/to/testcase",
      "componentC2/path/to/testcase"
    ]
  }
}
```

Then **all** testcase which its path contain componentA/path/to/testcase will be belong to componentA, ...

- *variant, version_sw*: configuration file has low priority than command line

Args: config_file : path to configuration file.

Returns: dConfig : configuration object.

RobotResults2DB.robot2db.**process_metadata**(metadata, default_metadata={'author': '', 'category': '', 'component': '', 'configfile': '', 'machine': '', 'project': 'ROBFW', 'tags': '', '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.

RobotResults2DB.robot2db.**process_suite**(db, suite, _tbl_test_result_id, root_metadata, dConfig=None)

Process to the lowest suite level (test file):

- Create new file and its header information
- Then, process all child test cases

Args: suite : Robot suite object.

_tbl_test_result_id : UUID of test result for importing.

root_metadata : metadata information.

dConfig: configuration data which is parsed from given json configuration file.

Returns: None.

RobotResults2DB.robot2db.**process_suite_metadata**(suite, default_metadata={'author': '', 'category': '', 'component': '', 'configfile': '', 'machine': '', 'project': 'ROBFW', 'tags': '', '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.

`RobotResults2DB.robot2db.process_test(db, test, file_id, test_result_id, metadata_info, test_number)`
Process test case data and create new test case record.

Args: db : database object.

test : Robot test object.

file_id : file ID for mapping.

test_result_id : test result ID for mapping.

metadata_info : metadata information.

test_number : order of test case in file.

Returns: None.

`RobotResults2DB.robot2db.truncate_string(sString, iMaxLength, sEndChars='...')`
Truncate input string before importing to database.

Args: sString : input string for truncation.

iMaxLength : max length of string to be allowed.

sEndChars (optional): end characters which are added to end of truncated string.

Returns: content : string after truncation.

`RobotResults2DB.robot2db.validate_config(dConfig, sSchema={'component': [<class 'str'>, <class 'dict'>], 'variant': <class 'str'>, 'version_sw': <class 'str'>}, bExitOnFail=True)`

Validate the json configuration base on given schema.

Default schema just supports “component”, “variant” and “version_sw”

```
CONFIG_SCHEMA = {  
    "component" : [str, dict],  
    "variant"   : str,  
    "version_sw": str,  
}
```

Args: dConfig : json configuration object to be verified.

sSchema (optional): schema for the validation. *CONFIG_SCHEMA* is used as default.

bExitOnFail (optional): If True, exit tool in case the validation is fail.

Returns: bValid : True if the given json configuration data is valid.

PYTHON MODULE INDEX

r

RobotResults2DB.CDataBase, [7](#)

RobotResults2DB.robot2db, [12](#)