RobotLog2DB

v. 1.5.2

Tran Duy Ngoan

28.06.2024

Contents

Chapter 1

Introduction

RobotLog2DB is a command-line tool that enables you to import Robot Framework XML result files into TestResultWebApp's database for presenting an overview about the whole test execution and details of each test result.

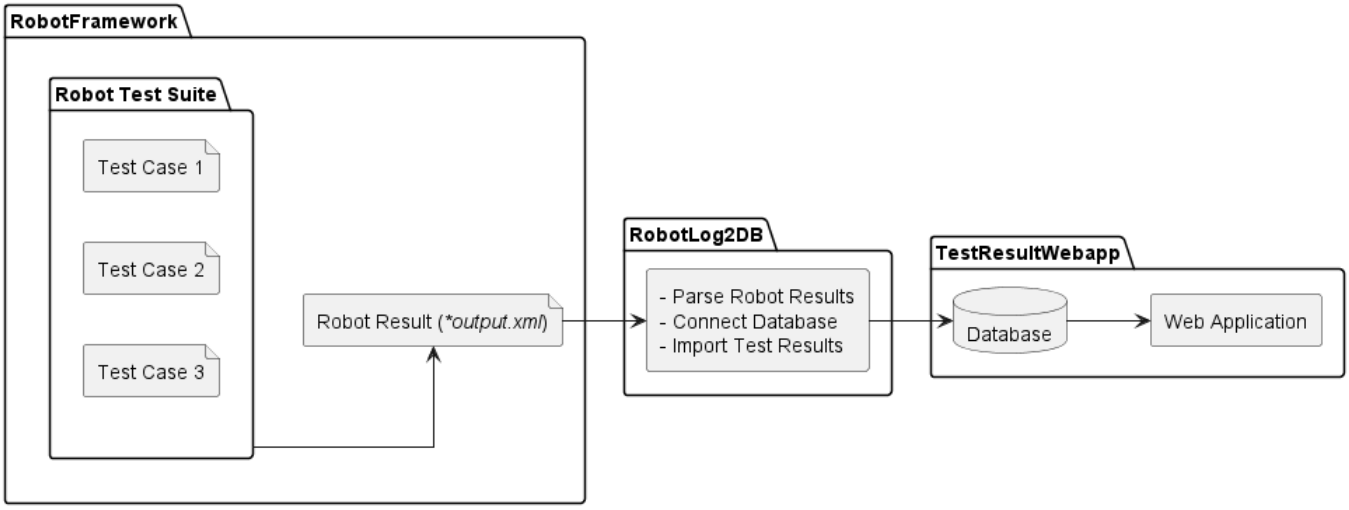

Figure 1.1: Tool data flow

RobotLog2DB tool requires serveral arguments, including the location of the Robot Framework XML result file(s) to parse all information of test execution result and TestResultWebApp's database credential for importing that result.

TestResultWebApp requires some mandatory information to manage and display the test result properly. Therefore, they should be provided in Robot Framework test case before execution, then they will be available in XML result file for importing.

However, you can use optional arguments of **RobotLog2DB** tool to provide those data if they are missing or you want to overwrite them with the expected values.

Finally, **RobotLog2DB** also allows you to append existing results in the database, which is helpful when you need to update previous test results or add the missing XML result file(s) from previous tool execution.

Chapter 2

Description

TestResultWebApp requires **project/variant**, **software version** as essential data to manage results in database and some optional information such as **component**, ... to visualize test data properly on web application.

These data are not the mandatory fields of Robot Framework test case or XML result file. So that, they should be defined as `Metadata` and `[Tags]` in Robot Framework test case, then the generated XML result file includes all necessary information of test execution for importing to TestResultWebApp's database.

The next section Get Robot Framework XML result will introduce you the Robot Framework test case settings to make necessary information available for displaying on TestResultWebApp.

In case you have already had the Robot Framework XML file(s) without those definitions, do not worry because **RobotLog2DB** will handle those data with default values.

Futhermore, you can specify them with your expected values by providing as command line arguments when executing **RobotLog2DB** tool. The detail of those command line arguments are described in Specify essential information with optional arguments section.

2.1 Get Robot Framework XML result

2.1.1 Robot Framework test case Settings

The below document is recommended Robot Framework test case `Settings` before execution. So that, the generated **.xml* result file will contain all necessary information.

For the whole test execution:

- Project/Variant

```
Metadata    project     ${Project_name}
```

- Versions

```
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

```
Metadata    component    ${Component_name}
```

- Test Tool - Test 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}
```

2.1.2 Sample Robot Framework Test Case

Sample Robot Framework test case with the neccessary information for importing to TestResultWebApp's database:

```
*** Settings ***
# Test execution level
Metadata    project         ROBFW               # 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 4.1.3 (Python 3.9.16 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

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

Listing 2.1: Sample Robot Framework testcase

Hint

Instead of setting `Metadata` in Robot Framework test case, you can also specify them as the `--metadata` command line argument when executing Robot Framework test cases. Furthermore, 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` .

2.1.3 Execute Robot Framework test case(s) to get result file

Now, execute your Robot Framework test case(s) with your IDE or using command line to get the Robot result file

```
robot your_testcases.robot
```

Or with python module

```
python -m robot your_testcases.robot
```

The default name of Robot result file is *output.xml*. Its filename can be changed by specifying other filename with argument `--output (-o)` when executing Robot Framework test case(s)

```
robot your_testcases.robot --output path/to/robot_result.xml
```

2.2 Tool features

After getting the Robot Framework **.xml* result file(s), you can use the **RobotLog2DB** tool to import the test results into TestResultWebApp's database.

Its usage and enhance features are described as below.

2.2.1 Usage

Use below command to get tools's usage:

```
RobotLog2DB -h
```

The tool's usage should be showed as below:

```
usage: RobotLog2DB (RobotXMLResult to TestResultWebApp importer) [-h] [-v]
                   [--recursive] [--dryrun] [--append] [--UUID UUID]
                   [--variant VARIANT] [--versions VERSIONS] [--config CONFIG]
                   resultxmlfile server user password database

RobotLog2DB imports XML result files (default: output.xml) generated by the
                    Robot Framework into a WebApp database.

positional arguments:
resultxmlfile        absolute or relative path to the result file or directory
                     of result 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 schema for database login.

optional arguments:
-h, --help           show this help message and exit
-v, --version        version of the RobotLog2DB importer.
--recursive          if set, then the path is searched recursively for output
                     files to be imported.
--dryrun             if set, then verify all input arguments (includes DB
                     connection) and show what would be done.
--append             is used in combination with --UUID UUID.If set, allow to
                     append new result(s) to existing execution result UUID in
                     --UUID argument.
--UUID UUID          UUID used to identify the import and version ID on webapp.
                     If not provided RobotLog2DB will generate an UUID for the
                     whole import.
--variant VARIANT    variant name to be set for this import.
--versions VERSIONS  metadata: Versions (Software;Hardware;Test) to be set for
                     this import (semicolon separated).
--config CONFIG      configuration json file for component mapping information.
--interface {db,rest}
                     database access interface.
--testrunurl TESTRUNURL
                     link to test execution job: Jenkins job, Gitlab CI/CD pipeline, ...
```

As above instruction, **RobotLog2DB** tool requires 5 positional arguments which contains all required information for inporting.
The below command is simple usage with all required arguments to import robot results into TestResultWebApp's database:

```
RobotLog2DB output.xml localhost db_user db_pw db_name
```

Besides the executable file **RobotLog2DB** you can also run tool as a Python module

```
python -m RobotLog2DB output.xml localhost db_user db_pw db_name
```

Beside the required arguments, there are optional arguments which provides some enhance features as the following sections.

2.2.2 Verify provided arguments

Sometimes, we just want to validate the **.xml* and database connection without changing anything in the database, the optional argument `--dryrun` can be used in this case.

When executing in dryrun mode, **RobotLog2DB** will:

- Verify the provided Robot Framework **.xml* result file is valid or not.
- Verify the database connection with provided credential.
- Verify other information which given in optional arguments.
- Just print all test cases will be imported without touching database.

This feature will helps you to ensure that there is no error when executing **RobotLog2DB** tool (normal mode) to create new record(s) and update TestResultWebApp's database.

2.2.3 Searching *.xml result file(s)

The first argument `resultxmlfile` of **RobotLog2DB** can be a single file or the folder that contains multiple result files.

When the folder is used, **RobotLog2DB** will only search for **.xml* file(s) 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 **RobotLog2DB** command.

When `--recursive` argument is set, **RobotLog2DB** 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.

2.2.4 Handle essential information for TestResultWebApp

Default values

TestResultWebApp requires **Project**, **Software version** to manage the execution results and **Component** to group test cases in the displayed charts.

In case above information is missing in test case settings during the test case execution, that leads to the missing information in the *output.xml* result file. So, this missing information will be set to default values when importing with **RobotLog2DB** 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.

Specify essential information with optional arguments

You can also provide essential information in command line when executing the **RobotLog2DB** tool with below optional arguments:

> **Warning**
>
> These below settings may overwrite the existing information of `project` , `version_sw` and `component` which have been defined in Robot Framework test case in above section.
> So, **only** use these arguments in case you want to define the missing information or overwrite the existing values with your expected values for the importing result.

- `--variant VARIANT` To specify the **Project/Variant** information.
- `--versions VERSIONS` To specify the **Software, Hardware** and **Test** versions information.
- `--config CONFIG` To provide a configuration **.json* file as `CONFIG` argument. Currently, the configuration **.json* supports below settings:
  - `"variant"` to specify the **Project/Variant** as `string` value.
  - `"version_sw"` to specify the **Software version** information as `string` value.
  - `"version_hw"` to specify the **Hardware under-test version** as `string` value.
  - `"version_test"` to specify the **Test version** as `string` value.

    > **Notice**
    >
    > These above settings with `--config CONFIG` will have lower priority than the commandline arguments `--variant VARIANT` and `--versions VERSIONS`

  - `"testtool"` to specify the **Test toolchain** as `string` value.
  - `"tester"` to specify the **Test user** as `string` value.
  - `"components"` to specify the **Component** information which will be displayed on TestResultWebApp. Value can be:
    * `string` : to specify the same **Component** for all test casea within this execution.

      ```
      {
        "components" : "atest",
        ...
      }
      ```

    * `object` : to define the mapping json object between **Component** info as key and a directory path (or list of directory paths) to Robot Framework test file (**.robot* file) as value.

      ```
      {
        "components" : {
          "cli"        : "robot/cli",
          "core"       : "robot/core",
          "external"   : "robot/external",
          "keywords"   : "robot/keywords",
          "libdoc"     : "robot/libdoc",
          "connectivity" : [
              "selftest/serial",
              "selftest/ssh",
              "selftest/tcpip"
          ]
        },
        ...
      }
      ```

      As above sample configuration, the `"components"` key contains the mappings for individual components, such as **cli**, **core**, **external**, **keywords** and **libdoc**, where the value is the directory path for the corresponding Robot Framework files.

Additionally, the **connectivity** key is an example of a mapping where the value is a list of directory paths, indicating that the **connectivity** component is composed of all Robot Framework files located in multiple directories, namely **selftest/serial**, **selftest/ssh** and **selftest/tcpip**.
In other words, the component mapping can be explained as below:

- Test case(s) under **robot/cli** folder is belong **cli** component
- Test case(s) under **robot/core** folder is belong **core** component
- ...
- **connectivity** component contains all test case(s) which is located under **selftest/serial**, **selftest/ssh** and **selftest/tcpip** folders.

In case the given configuration **.json* is not valid or unsupported key is used, the corresponding error will be raised.

2.2.5 Append to existing execution result

RobotLog2DB also allows you to append new test result(s) (missing from previous import, on other test setup, ...) into the existing execution result (identified by the **UUID**) in TestResultWebApp's database. The combination of optional arguments `--UUID <UUID>` and `--append` should be used in this case.

The `--append` makes **RobotLog2DB** run in append mode and the `--UUID <UUID>` is used to specify the existing UUID of execution result to be appended.

For example, the result with UUID **c7991c07-4de2-4d65-8568-00c5556c82aa** is already existing in TestResultWebApp's database and you want to append result(s) in **output.xml** into that execution result.

The command will be used as below:

```
python -m RobotLog2DB output.xml localhost testuser testpw testdb --UUID ↩
    ↪ c7991c07-4de2-4d65-8568-00c5556c82aa --append
```

If the argument `--UUID <UUID>` is used without `--append` :

- An error will be thrown and the import job is terminated immediately if the provided **UUID** is already existing

```
FATAL ERROR: Execution result with UUID 'c7991c07-4de2-4d65-8568-00c5556c82aa' is ↩
    ↪ already existing.
              Please use other UUID (or remove '--UUID' argument from your command) ↩
    ↪ for new execution result.
              Or add '--append' argument in your command to append new result(s) to ↩
    ↪ this existing UUID.
```

- The importing execution result will have an identifier as the provided **UUID** if that **UUID** is not existing

If the argument `--append` is used and:

- given UUID in `--UUID <UUID>` argument is existing: the new result(s) will be appended to that UUID
- given UUID in `--UUID <UUID>` argument is not existing: tool will be terminated immediately with below error

```
FATAL ERROR: Execution result with UUID 'c7991c07-4de2-4d65-8568-00c5556c82aa' is ↩
    ↪ not existing for appending.
              Please use an existing UUID to append new result(s) to that UUID.
              Or remove '--append' argument in your command to create new execution ↩
    ↪ result with given UUID.
```

- `--UUID <UUID>` is not provided: tool will be terminated immediately with below error

```
FATAL ERROR: '--append' argument should be used in combination with '--UUID UUID` ↩
    ↪ argument
```

Notice

When using append mode and `project` / `variant` , `version_sw` are provided within `--variant VARIANT` , `--versions VERSIONS` or `--config CONFIG` arguments, they will be validated with the existing values in database.
An error will be raised in case the given value is not matched with the existing ones. E.g:

```
FATAL ERROR: Given version software 'my_version' is different with existing value 'SW01' in database.
```

2.2.6 Switch Database Access Interface

In addition to direct database connections — which can sometimes be restricted or prohibited due to security concerns — the **RobotLog2DB** tool also provides users with an alternative way (interface) to import test data into the database, namely the REST API.

`--interface INTERFACE` : An optional argument to specify the database backend interface. The default value is `db` (direct connection to MySQL database). It can also be set to `rest` to import data into the database via a REST API.

Direct Connection Backend (interface="db")

The default mode of operation for the import tool is the direct connection to the MySQL database. This means that when you run the tool without specifying the `--interface` option, it automatically uses the `db` interface, establishing a direct connection to the MySQL database.

You can also explicitly specify the `db` interface by using the `--interface` option with the value `db` .

```
RobotLog2DB output.xml db_host db_user db_password db_name --interface db
```

This method provides flexibility, allowing users to choose between interfaces explicitly or rely on the default behavior.

REST API Backend (interface="rest")

The 'rest' interface allows users to import data into the database through a REST API. This mode is particularly useful in scenarios where direct database connections are restricted or not feasible, such as when the database is secured and direct connections are not allowed.

```
RobotLog2DB output.xml api_endpoint api_user api_password db_name --interface rest
```

Additional Notes

This feature offers easy switching between database backends, making it transparent and convenient for end-users.
When using the REST API interface, ensure that you have the necessary credentials and permissions to access the REST API endpoints.

2.3 Display on WebApp

As soon as the *output.xml* file(s) is imported sucessfully to database, the result for that execution will be available on TestResultWebApp.

Above test case settings in Robot Framework test case will be reflected on **Dashboard** (General view) and **Data table** (Detailed view) as below figures:

Execution result metadata:

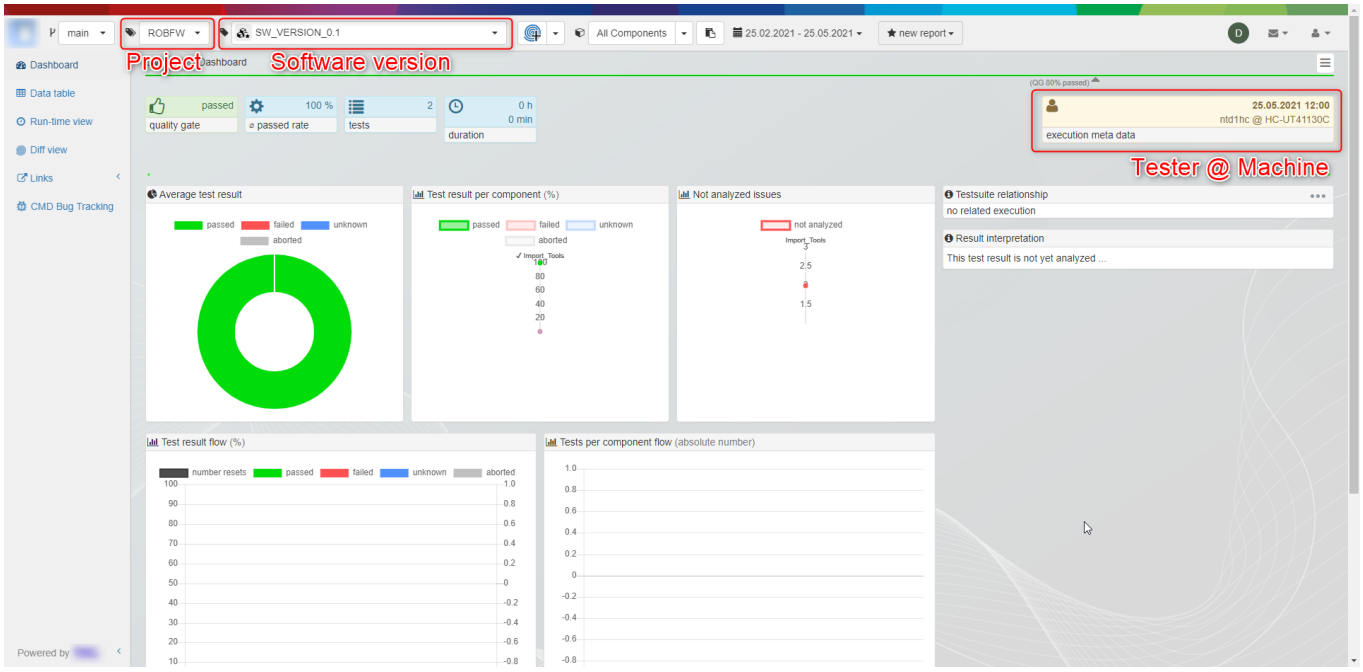

Figure 2.1: Dashboard view

Suite/File metadata and test case information:

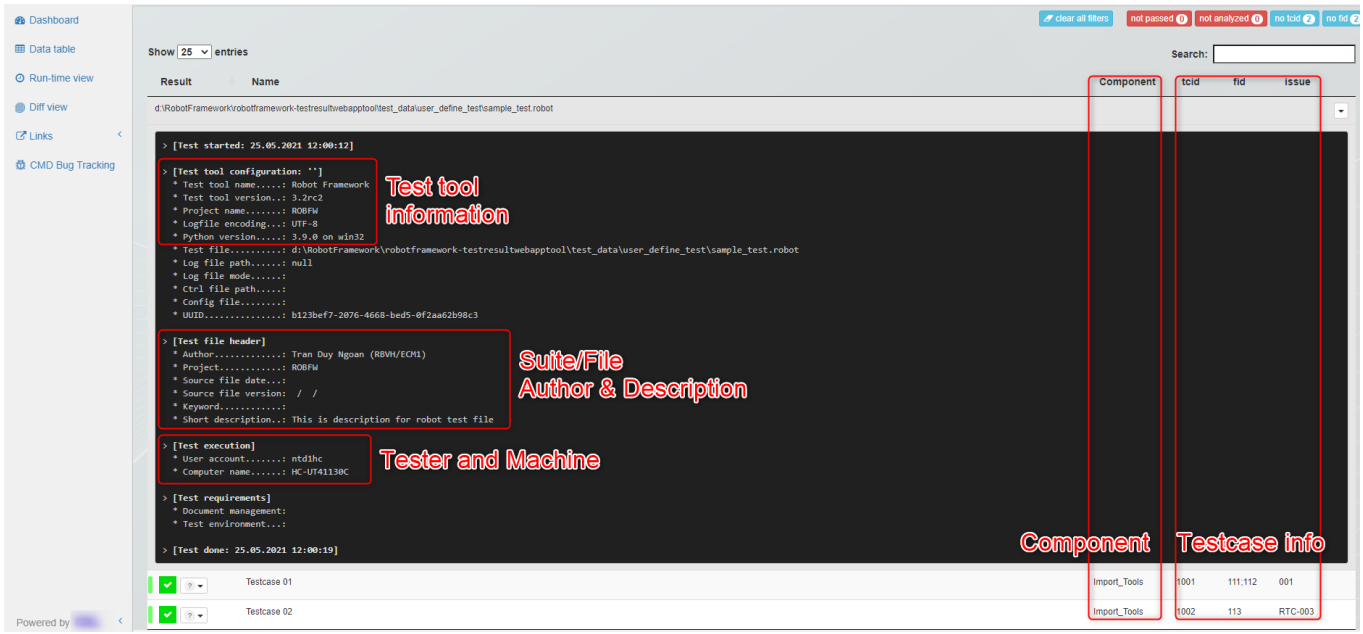

Figure 2.2: Datatable view

Chapter 3

robotlog2db.py

3.1 Function: collect_xml_result_files

Collect all valid Robot xml result file in given path.

Arguments:

- `path`
  / *Condition*: required / *Type*: str /
  Path to Robot result folder or file to be searched.
- `search_recursive`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If set, the given path is searched recursively for xml result files.

Returns:

- `lFoundFiles`
  / *Type*: list /
  List of valid xml result file(s) in given path.

3.2 Function: validate_xml_result

Verify the given xml result file is valid or not.

Arguments:

- `xml_result`
  / *Condition*: required / *Type*: str /
  Path to Robot result file.
- `xsd_schema`
  / *Condition*: optional / *Type*: str / *Default*: <installed_folder>/xsd/robot.xsd /
  Path to Robot schema *.xsd file.
- `exit_on_failure`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If set, exit with fatal error if the schema validation of given xml file failed.

Returns:

- / *Type*: bool /
  True if the given xml result is valid with the provided schema *.xsd.

3.3 Function: is_valid_uuid

Verify the given UUID is valid or not.

Arguments:

- `uuid_to_test`
  / *Condition*: required / *Type*: str /
  UUID to be verified.
- `version`
  / *Condition*: optional / *Type*: int / *Default*: 4 /
  UUID version.

Returns:

- `bValid`
  / *Type*: bool /
  True if the given UUID is valid.

3.4 Function: is_valid_config

Validate the json configuration base on given schema.
Default schema supports below information:

```
CONFIG_SCHEMA = {
    "components": [str, dict],
    "variant"   : str,
    "version_sw": str,
    "version_hw": str,
    "version_test": str,
    "testtool"  :  str,
    "tester"    :  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.

3.5 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 expectated info (ID) from tag info.

Returns:

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

3.6 Function: get_branch_from_swversion

Get branch name from software version information.

Convention of branch information in suffix of software version:

- All software version with .0F is the main/freature 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.

Arguments:

- `sw_version`
  / *Condition*: required / *Type*: str /
  Software version.

Returns:

- `branch_name`
  / *Type*: str /
  Branch name.

3.7 Function: format_time

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

Arguments:

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

Returns:

- `sFormatedTime`
  / *Type*: str /
  TestResultWebApp's time as format `%Y-%m-%d %H:%M:%S`.

3.8 Function: retrieve_result_starttime

Retrieve starttime infomration from given result object (TestSuite or TestCase). In case the starttime in given suite is 'N/A', it will try to get this information from its children suite/test.

Arguments:

- `stime`
  / *Condition*: required / *Type*: TestSuite or TestCase object /
  Result object to retrieve starttime.

Returns:

- / *Type*: str /
  Start time of given result.

3.9 Function: retrieve_result_endtime

Retrieve endtime infomration from given result object (TestSuite or TestCase). In case the endtime in given suite is 'N/A', it will try to get this information from its children suite/test.

Arguments:

- `stime`
  / *Condition*: required / *Type*: TestSuite or TestCase object /
  Result object to retrieve endtime.

Returns:

- / *Type*: str /
  End time of given result.

3.10 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.

3.11 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.

3.12 Function: process_suite

Process to the lowest suite level (test file):

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

Arguments:

- `db`
  / *Condition*: required / *Type*: CDataBase object /
  CDataBase object.
- `suite`
  / *Condition*: required / *Type*: TestSuite object /
  Robot suite object.
- `_tbl_test_result_id`
  / *Condition*: required / *Type*: str /
  UUID of test result for importing.
- `root_metadata`
  / *Condition*: required / *Type*: dict /
  Metadata information from root level.
- `dConfig`
  / *Condition*: required / *Type*: dict / *Default*: None /
  Configuration data which is parsed from given json configuration file.

Returns:

(*no returns*)

3.13 Function: process_test

Process test case data and create new test case record.

Arguments:

- `db`
  / *Condition*: required / *Type*: CDataBase object /
  CDataBase object.
- `test`
  / *Condition*: required / *Type*: TestCase object /
  Robot test object.
- `file_id`
  / *Condition*: required / *Type*: int /
  File ID for mapping.
- `test_result_id`
  / *Condition*: required / *Type*: str /
  Test result ID for mapping.
- `metadata_info`
  / *Condition*: required / *Type*: dict /
  Metadata information.
- `test_number`
  / *Condition*: required / *Type*: int /
  Order of test case in file.

Returns:

(*no returns*)

3.14 Function: process_config_file

Parse information from configuration file:

- `component`:

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

  Then all testcases which their paths contain `componentA/path/to/testcase` will be belong to `componentA`, ...

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

Arguments:

- `config_file`
  / *Condition*: required / *Type*: str /
  Path to configuration file.

Returns:

- `dConfig`
  / *Type*: dict /
  Configuration object.

3.15 Function: normalize_path

Normalize path file.

Arguments:

- `sPath`
  / *Condition*: required / *Type*: str /
  Path file to be normalized.
- `sNPath`
  / *Type*: str /
  Normalized path file.

3.16 Function: RobotLog2DB

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. Parse Robot results.
3. Connect to database.
4. Import results into database.
5. Disconnect from database.

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.
  - 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 verify all input arguments (includes DB connection) and show what would be done.
  - append : if True, then allow to append new result(s) to existing execution result UUID which is provided by --UUID argument.
  - 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.
- 'testrunurl': link to test execution job: Jenkins job, Gitlab CI/CD pipeline, ...

Returns:

(*no returns*)

3.17 Class: Logger

Imported by:

```
from RobotLog2DB.robotlog2db import Logger
```

Logger class for logging message.

3.17.1 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*)

3.17.2 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*)

3.17.3 Method: log_warning

Write warning message to console/file output.

Arguments:

- `msg`
  / *Condition*: required / *Type*: str /
  Warning message which is written to output.

Returns:

(*no returns*)

3.17.4 Method: log_error

Write error message to console/file output.

Arguments:

- `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.

Returns:

(*no returns*)

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	RobotLog2DB
Version	1.5.2
Date	28.06.2024
Description	Imports robot result(s) to TestResultWebApp database
Package URL	robotframework-robotlog2db
Author	Tran Duy Ngoan
Email	Ngoan.TranDuy@vn.bosch.com
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 5

History

0.1.0	07/2022
Initial version	
1.2.1	22.08.2022
Rework repository's document bases on GenPackageDoc	
1.2.2	13.10.2022
- Fix findings and enhance README and document files *- Change argument name 'outputfile' to 'resultxmlfile'*	
1.2.3	10.11.2022
Rename package to RobotLog2DB	
1.2.4	18.11.2022
Add -append argument which allow to append into existing UUID	
1.3.0	06.12.2022
- Rework optional arguments in command line *- Improve coding: logging messages, shorten variable names, ...*	
1.3.1	09.01.2023
Improve messages when verify configuration json file	
1.3.2	28.02.2023
- Rename key to 'components' in configuration json file *- Change implementation of append mode to raise errors without proper given UUID* *- Enhance console log with component and append mode information* *- Add more supported keys in configuration json file*	
1.3.3	01.03.2023
Add a validation for xmlresultfile with robot schema	
1.3.4	13.03.2023
- Add a validation for existing project/variant and version_sw in db when using append mode *- Remove db maxlength handlers: truncations and validation*	
1.3.5	21.03.2023
Support 4 byte characters for importing to db	
1.3.6	05.04.2023
- Enhance console log with test case counter and component statistics *- Add try/except for database access*	
1.3.7	14.06.2023
Update README with new instruction for installation and image's links	

1.3.8	21.06.2023
Fix issue with suite.starttime at root suite is N/A	
1.3.9	23.06.2023
- Retrieve start/end time for suite *- Update result schema to support Robotframework Version 6.1*	
1.4.0	19.09.2023
Adaption for Robotframework 6.1 with change of `TestSuite.source` *datatype*	
1.4.1	15.03.2024
Update `robot.xsd` *schema to support the new log level* `USER` *of Robotframework*	
1.5.0	22.04.2024
Add optional argument `--interface` *which allow to switch between* `db` *and* `rest` *interfaces*	
1.5.1	14.06.2024
Fix issue when test result has SKIP status	
1.5.2	28.06.2024
Add optional argument `--testrunurl` *to specify link to test job such as Jenkins or Gitlab CI/CD*	

RobotLog2DB.pdf

Created at 28.06.2024 - 12:56:11

by GenPackageDoc v. 0.41.1