PyTestLog2DB

v. 0.1.9

Tran Duy Ngoan

20.04.2023

Contents

Chapter 1

Introduction

PyTestLog2DB is a command-line tool that enables you to import pytest XML result files into TestResultWebApp's database for presenting an overview about the whole test execution and detail of each test result.
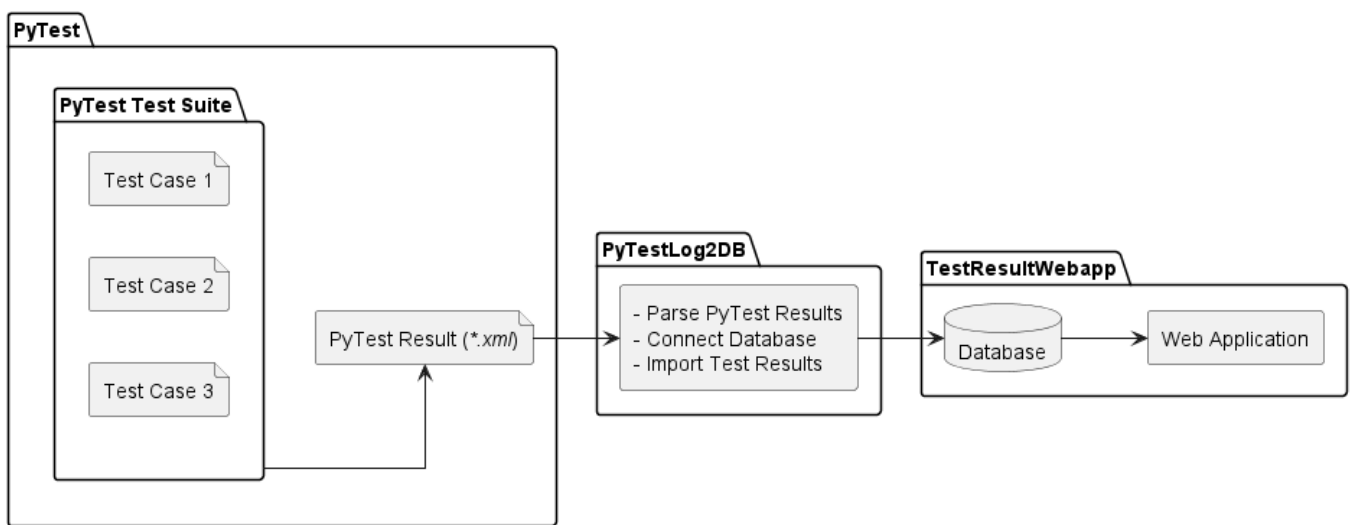
Figure 1.1: Tool data flow

PyTestLog2DB tool requires serveral arguments, including the location of the pytest 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, but the generated **xml* file contains only the basic test result information. So that, **PyTestLog2DB** tool will set those required information with default values.

Besides, you can also use optional arguments of **PyTestLog2DB** tool to provide missing information or you want to overwrite them with the expected values.

Finally, **PyTestLog2DB** 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

2.1 Execute pytest test case(s) to get result file

When executing pytest test case(s), the test result is only displayed on console log without generating any result file by default.

In order to get the result **.xml* (JUnit XML format) files, the optional argument `--junit-xml=<log>` needs to be specified when executing the pytest test case(s).

The example pytest command to get the ***.xml** result file:

```
pytest --junit-xml=path/to/result.xml pytest/folder
```

2.2 Tool features

2.2.1 Usage

Use below command to get tools's usage:

```
PyTestLog2DB -h
```

The tool's usage should be showed as below:

```
usage: PyTestLog2DB (PyTestXMLReport to TestResultWebApp importer) [-h] [-v]
                    [--recursive] [--dryrun] [--append] [--UUID UUID]
                    [--variant VARIANT] [--versions VERSIONS] [--config CONFIG]
                    resultxmlfile server user password database

PyTestLog2DB imports pytest JUnit XML report file(s)generated by pytest into a WebApp ←
    ↪ database.

positional arguments:
resultxmlfile         absolute or relative path to the pytest JUnit XML report
                      file or directory of report 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 PyTestLog2DB 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 PyTestLog2DB 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.
```

The below command is simple usage with all required arguments to import pytest results into TestResultWebApp's database:

```
PyTestLog2DB resultxmlfile server user password database
```

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

```
python -m PyTestLog2DB resultxmlfile server user password database
```

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, **PyTestLog2DB** will:

- Verify the provided pytest *.xml 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 **PyTestLog2DB** 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 **PyTestLog2DB** can be a single file or the folder that contains multiple result files.

When the folder is used, **PyTestLog2DB** 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 **PyTestLog2DB** command.

When `--recursive` argument is set, **PyTestLog2DB** 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.

But the ***.xml** report file which is generated by pytest contains only the testcase result(s) and no data for the information which is required by TestResultWebApp.

So that, the missing information will be set to default values when importing with **PyTestLog2DB** tool:

- **Project**: `PyTest`

- **Software version**: the execution time `%Y%m%d_%H%M%S` as default value. E.g `20221128_143547`

- **Hardware version**: empty string

- **Test version**: empty string

- **Component**: `unknown`

- **Test tool**: the combination of current Python and pytest version. E.g `PyTest 6.2.5 (Python 3.9.0)`

- **Tester**: The current user

Specify essential information with optional arguments

You can also provide essential information in command line when executing **PyTestLog2DB** tool with below optional arguments:

- `--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 test case `classname` (list of `classname`) (**.robot* file) as value.

```json
{
  "components": {
    "Testsuite1": "test-data.test-tsclass.TestSuite1",
    "Testsuite2": "test-data.test-tsclass.TestSuite2",
    "Others"    : [
          "test-data.test-ts1",
          "test-data.test-ts2"
    ]
  },
  ...
}
```

As above sample configuration, the `"components"` key contains the mappings for individual components, such as **Testsuite1** and **Testsuite2**, where the value is the `classname` (part of `classname`) of pytest test case(s).

Additionally, the **Others** key is an example of a mapping where the value is a list of test case `classname` , indicating that the **Others** component is composed of all pytest test case(s) which its `classname` contains **test-data.test-ts1** and **test-data.test-ts2**.

In other words, the component mapping can be explained as below:

· Test case(s) which its `classname` contains **test-data.test-tsclass.TestSuite1** is belong **Testsuite1** component

· Test case(s) which its `classname` contains **test-data.test-tsclass.TestSuite2** is belong **TestSuite2** component

· **Others** component consists of all test case(s) which its `classname` contains **test-data.test-ts1** and **test-data.test-ts2**.

> **! Hint**
>
> The `classname` of test case in the generated pytest result **.xml* file is the combination of directory, test file and class of defined pytest test case.
>
> Therefore, you can use directory information for component mapping to group all test cases in a folder as the same component.
>
> In addition, you can use the optional argument `--junit-prefix=str` when executing pytest to set the prefix to the `classname` of the test case in the result **.xml* file. After that, you can use that prefix information for component mapping when importing the result **.xml* file with the **PyTestLog2DB** tool.

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

PyTestLog2DB 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 **PyTestLog2DB** 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 PyTestLog2DB 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.
> ```

2.3 Display on TestResultWebApp

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

Dashboard view:
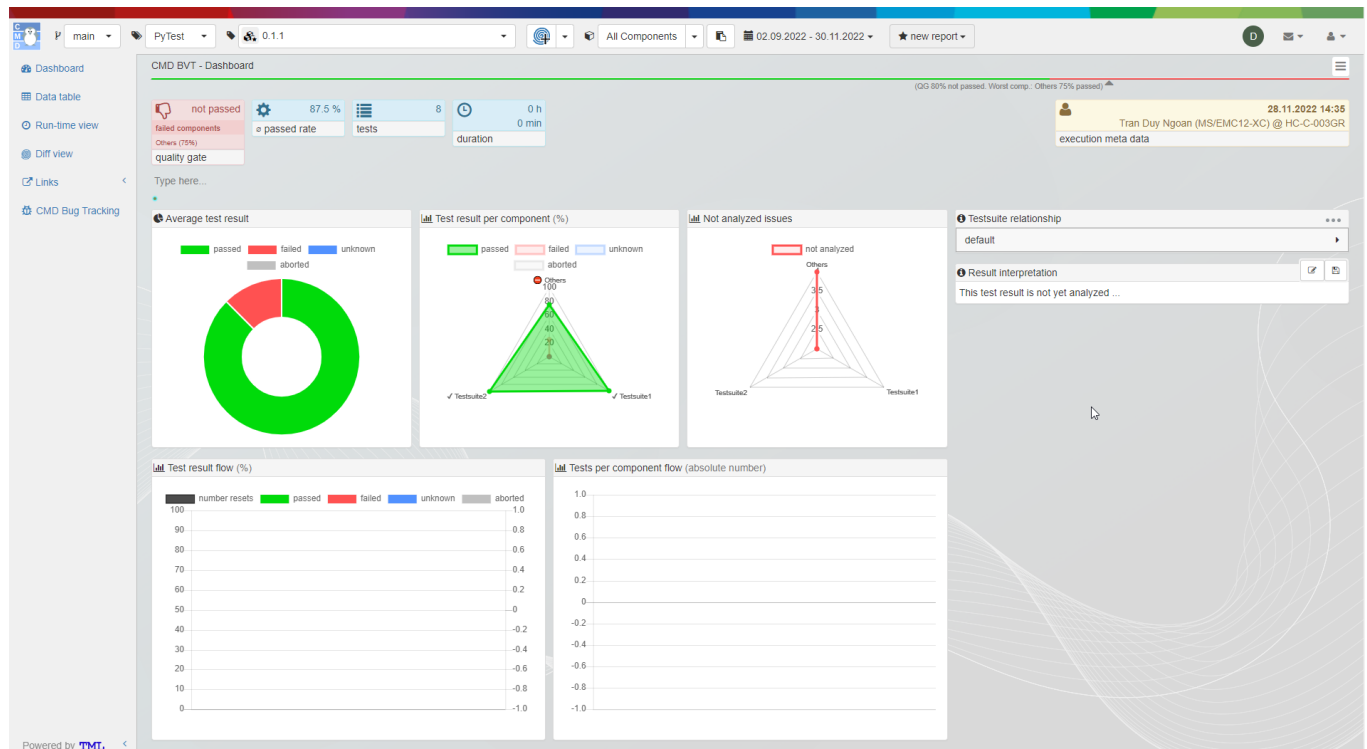
Figure 2.1: Dashboard view

Datatable view:
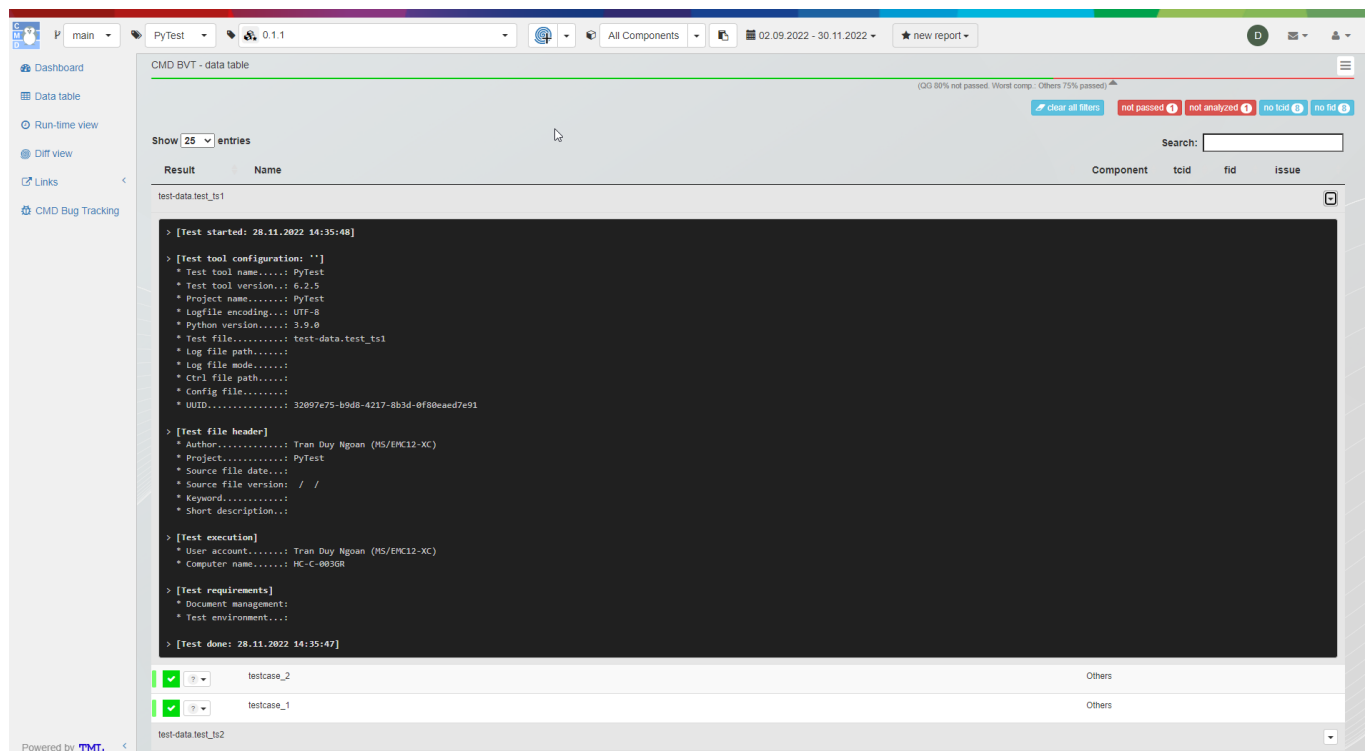
Figure 2.2: Datatable view

Chapter 3

CDataBase.py

3.1   Class: CDataBase

Imported by:

```
from PyTestLog2DB.CDataBase import CDataBase
```

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

3.1.1   Method: connect

Connect to the database with provided authentication and db info.
Arguments:

- host

  / *Condition*: required / *Type*: str /

  URL which is hosted the TestResultWebApp's database.

- user

  / *Condition*: required / *Type*: str /

  User name for database authentication.

- passwd

  / *Condition*: required / *Type*: str /

  User's password for database authentication.

- database

  / *Condition*: required / *Type*: str /

  Database name.

- charset

  / *Condition*: optional / *Type*: str / *Default*: 'utf8' /

  The connection character set.

- use_unicode

  / *Condition*: optional / *Type*: bool / *Default*: True /

  If True, CHAR and VARCHAR and TEXT columns are returned as Unicode strings, using the configured character set.

Returns:

(*no returns*)

3.1.2 Method: disconnect

Disconnect from TestResultWebApp's database.

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.1.3 Method: cleanAllTables

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

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.1.4 Method: sCreateNewTestResult

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`.

Arguments:

- `_tbl_prj_project`
  / *Condition*: required / *Type*: str /
  Project information.

- `_tbl_prj_variant`
  / *Condition*: required / *Type*: str /
  Variant information.

- `_tbl_prj_branch`
  / *Condition*: required / *Type*: str /
  Branch information.

- `_tbl_test_result_id`
  / *Condition*: required / *Type*: str /
  UUID of test result.

- `_tbl_result_interpretation`
  / *Condition*: required / *Type*: str /
  Result interpretation.

- `_tbl_result_time_start`
  / *Condition*: required / *Type*: str /
  Test result start time as format `%Y-%m-%d %H:%M:%S`.

- `_tbl_result_time_end`
  / *Condition*: required / *Type*: str /
  Test result end time as format `%Y-%m-%d %H:%M:%S`.

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

- ␣tbl␣result␣version␣sw␣test
  / *Condition*: required / *Type*: str /
  Test version information.

- ␣tbl␣result␣version␣target
  / *Condition*: required / *Type*: str /
  Hardware version information.

- ␣tbl␣result␣jenkinsurl
  / *Condition*: required / *Type*: str /
  Jenkinsurl in case test result is executed by jenkins.

- ␣tbl␣result␣reporting␣qualitygate
  / *Condition*: required / *Type*: str /
  Qualitygate information for reporting.

Returns:

- ␣tbl␣test␣result␣id
  / *Type*: str /
  test␣result␣id of new test result.

3.1.5   Method: nCreateNewFile

Create new file entry in tbl␣file table.

Arguments:

- ␣tbl␣file␣name
  / *Condition*: required / *Type*: str /
  File name information.

- ␣tbl␣file␣tester␣account
  / *Condition*: required / *Type*: str /
  Tester account information.

- ␣tbl␣file␣tester␣machine
  / *Condition*: required / *Type*: str /
  Test machine information.

- ␣tbl␣file␣time␣start
  / *Condition*: required / *Type*: str /
  Test file start time as format %Y-%m-%d %H:%M:%S.

- ␣tbl␣file␣time␣end
  / *Condition*: required / *Type*: str /
  Test file end time as format %Y-%m-%d %H:%M:%S.

- ␣tbl␣test␣result␣id
  / *Condition*: required / *Type*: str /
  UUID of test result for linking to tbl␣result table.

- ␣tbl␣file␣origin
  / *Condition*: required / *Type*: str /
  Origin (test framework) of test file. Deafult is "ROBFW"

Returns:

- iInsertedID
  / *Type*: int /
  ID of new entry.

3.1.6 Method: vCreateNewHeader

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

Arguments:

- `_tbl_file_id`
  / *Condition*: required / *Type*: int /
  File ID information.

- `_tbl_header_testtoolconfiguration_testtoolname`
  / *Condition*: required / *Type*: str /
  Test tool name.

- `_tbl_header_testtoolconfiguration_testtoolversionstring`
  / *Condition*: required / *Type*: str /
  Test tool version.

- `_tbl_header_testtoolconfiguration_projectname`
  / *Condition*: required / *Type*: str /
  Project name.

- `_tbl_header_testtoolconfiguration_logfileencoding`
  / *Condition*: required / *Type*: str /
  Encoding of logfile.

- `_tbl_header_testtoolconfiguration_pythonversion`
  / *Condition*: required / *Type*: str /
  Python version info.

- `_tbl_header_testtoolconfiguration_testfile`
  / *Condition*: required / *Type*: str /
  Test file name.

- `_tbl_header_testtoolconfiguration_logfilepath`
  / *Condition*: required / *Type*: str /
  Path to log file.

- `_tbl_header_testtoolconfiguration_logfilemode`
  / *Condition*: required / *Type*: str /
  Mode of log file.

- `_tbl_header_testtoolconfiguration_ctrlfilepath`
  / *Condition*: required / *Type*: str /
  Path to control file.

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

- `_tbl_header_testtoolconfiguration_confname`
  / *Condition*: required / *Type*: str /
  Configuration name.

- `_tbl_header_testfileheader_author`
  / *Condition*: required / *Type*: str /
  File author.

- `_tbl_header_testfileheader_project`
  / *Condition*: required / *Type*: str /
  Project information.

- `_tbl_header_testfileheader_testfiledate`
  / *Condition*: required / *Type*: str /
  File creation date.

- `_tbl_header_testfileheader_version_major`
  / *Condition*: required / *Type*: str /
  File major version.

- `_tbl_header_testfileheader_version_minor`
  / *Condition*: required / *Type*: str /
  File minor version.

- `_tbl_header_testfileheader_version_patch`
  / *Condition*: required / *Type*: str /
  File patch version.

- `_tbl_header_testfileheader_keyword`
  / *Condition*: required / *Type*: str /
  File keyword.

- `_tbl_header_testfileheader_shortdescription`
  / *Condition*: required / *Type*: str /
  File short description.

- `_tbl_header_testexecution_useraccount`
  / *Condition*: required / *Type*: str /
  Tester account who run the execution.

- `_tbl_header_testexecution_computername`
  / *Condition*: required / *Type*: str /
  Machine name which is executed on.

- `_tbl_header_testrequirements_documentmanagement`
  / *Condition*: required / *Type*: str /
  Requirement management information.

- `_tbl_header_testrequirements_testenvironment`
  / *Condition*: required / *Type*: str /
  Requirement environment information.

- `_tbl_header_testbenchconfig_name`
  / *Condition*: required / *Type*: str /
  Testbench configuration name.

- `_tbl_header_testbenchconfig_data`
  / *Condition*: required / *Type*: str /
  Testbench configuration data.

- `_tbl_header_preprocessor_filter`
  / *Condition*: required / *Type*: str /
  Preprocessor filter information.

- `_tbl_header_preprocessor_parameters`
  / *Condition*: required / *Type*: str /
  Preprocessor parameters definition.

Returns:

(*no returns*)

3.1.7 Method: nCreateNewSingleTestCase

Create single testcase entry in tbl_case table immediately.

Arguments:

- _tbl_case_name
  / *Condition*: required / *Type*: str /
  Test case name.

- _tbl_case_issue
  / *Condition*: required / *Type*: str /
  Test case issue ID.

- _tbl_case_tcid
  / *Condition*: required / *Type*: str /
  Test case ID (used for testmanagement tool).

- _tbl_case_fid
  / *Condition*: required / *Type*: str /
  Test case requirement (function) ID.

- _tbl_case_testnumber
  / *Condition*: required / *Type*: int /
  Order of test case in file.

- _tbl_case_repeatcount
  / *Condition*: required / *Type*: int /
  Test case repeatition count.

- _tbl_case_component
  / *Condition*: required / *Type*: str /
  Component which test case is belong to.

- _tbl_case_time_start
  / *Condition*: required / *Type*: str /
  Test case start time as format %Y-%m-%d %H:%M:%S.

- _tbl_case_result_main
  / *Condition*: required / *Type*: str /
  Test case main result.

- _tbl_case_result_state
  / *Condition*: required / *Type*: str /
  Test case completion state.

- _tbl_case_result_return
  / *Condition*: required / *Type*: int /
  Test case result code (as integer).

- _tbl_case_counter_resets
  / *Condition*: required / *Type*: int /
  Counter of target reset within test case execution.

- _tbl_case_lastlog
  / *Condition*: required / *Type*: str /
  Traceback information when test case is failed.

- **_tbl_test_result_id**
  / *Condition*: required / *Type*: str /
  UUID of test result for linking to file in `tbl_result` table.

- **_tbl_file_id**
  / *Condition*: required / *Type*: int /
  Test file ID for linking to file in `tbl_file` table.

Returns:

- **iInsertedID**
  / *Type*: int /
  ID of new entry.

3.1.8 Method: nCreateNewTestCase

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

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

Arguments:

- **_tbl_case_name**
  / *Condition*: required / *Type*: str /
  Test case name.

- **_tbl_case_issue**
  / *Condition*: required / *Type*: str /
  Test case issue ID.

- **_tbl_case_tcid**
  / *Condition*: required / *Type*: str /
  Test case ID (used for testmanagement tool).

- **_tbl_case_fid**
  / *Condition*: required / *Type*: str /
  Test case requirement (function) ID.

- **_tbl_case_testnumber**
  / *Condition*: required / *Type*: int /
  Order of test case in file.

- **_tbl_case_repeatcount**
  / *Condition*: required / *Type*: int /
  Test case repeatition count.

- **_tbl_case_component**
  / *Condition*: required / *Type*: str /
  Component which test case is belong to.

- **_tbl_case_time_start**
  / *Condition*: required / *Type*: str /
  Test case start time as format `%Y-%m-%d %H:%M:%S`.

- **_tbl_case_result_main**
  / *Condition*: required / *Type*: str /
  Test case main result.

- _tbl_case_result_state
  / *Condition*: required / *Type*: str /
  Test case completion state.

- _tbl_case_result_return
  / *Condition*: required / *Type*: int /
  Test case result code (as integer).

- _tbl_case_counter_resets
  / *Condition*: required / *Type*: int /
  Counter of target reset within test case execution.

- _tbl_case_lastlog
  / *Condition*: required / *Type*: str /
  Traceback information when test case is failed.

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result for linking to file in tbl_result table.

- _tbl_file_id
  / *Condition*: required / *Type*: int /
  Test file ID for linking to file in `tbl_file` table.

Returns:

(*no returns*)

3.1.9 Method: vCreateTags

Create tag entries.

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result.

- _tbl_usr_result_tags
  / *Condition*: required / *Type*: str /
  User tags information.

Returns:

(*no returns*)

3.1.10 Method: vSetCategory

Create category entry.

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result.

- tbl_result_category_main
  / *Condition*: required / *Type*: str /
  Category information.

Returns:

(*no returns*)

3.1.11   Method: vUpdateStartEndTime

Create start-end time entry.

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result.

- _tbl_result_time_start
  / *Condition*: required / *Type*: str /
  Result start time as format `%Y-%m-%d %H:%M:%S`.

- _tbl_result_time_end
  / *Condition*: required / *Type*: str /
  Result end time as format `%Y-%m-%d %H:%M:%S`.

Returns:

(*no returns*)

3.1.12   Method: arGetCategories

Get existing categories.

Arguments:

(*no arguments*)

Returns:

- arCategories
  / *Type*: list /
  List of exsiting categories.

3.1.13   Method: vCreateAbortReason

Create abort reason entry.

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result.

- _tbl_abort_reason
  / *Condition*: required / *Type*: str /
  Abort reason.

- _tbl_abort_message
  / *Condition*: required / *Type*: str /
  Detail message of abort.

Returns:

(*no returns*)

3.1.14   Method: vCreateReanimation

Create reanimation entry.

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result.

- _tbl_num_of_reanimation
  / *Condition*: required / *Type*: int /
  Counter of target reanimation during execution.

Returns:

(*no returns*)

3.1.15   Method: vCreateCCRdata

Create CCR data per test case.

Arguments:

- _tbl_test_case_id
  / *Condition*: required / *Type*: int /
  test case ID.

- lCCRdata
  / *Condition*: required / *Type*: list /
  list of CCR data.

Returns:

(*no returns*)

3.1.16   Method: vFinishTestResult

Finish upload:

- First do bulk insert of rest of test cases if buffer is not empty.

- Then set state to "new report".

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  UUID of test result.

Returns:

(*no returns*)

3.1.17   Method: vUpdateEvtbls

Call update_evtbls stored procedure.

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.1.18 Method: vUpdateEvtbl

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

Arguments:

- `_tbl_test_result_id`
  / *Condition*: required / *Type*: str /
  UUID of test result.

Returns:

(*no returns*)

3.1.19 Method: vEnableForeignKeyCheck

Switch `foreign_key_checks` flag.

Arguments:

- `enable`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If True, enable foreign key constraint.

Returns:

(*no returns*)

3.1.20 Method: sGetLatestFileID

Get latest file ID from `tbl_file` table.

Arguments:

- `_tbl_test_result_id`
  / *Condition*: required / *Type*: str /
  UUID of test result.

Returns:

- `_tbl_file_id`
  / *Type*: int /
  File ID.

3.1.21 Method: vUpdateFileEndTime

Update test file end time.

Arguments:

- `_tbl_file_id`
  / *Condition*: required / *Type*: int /
  File ID to be updated.
- `_tbl_file_time_end`
  / *Condition*: required / *Type*: str /
  File end time as format `%Y-%m-%d %H:%M:%S`.

Returns:

(*no returns*)

3.1.22   Method: vUpdateResultEndTime

Update test result end time.

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  Result UUID to be updated.

- _tbl_result_time_end
  / *Condition*: required / *Type*: str /
  Result end time as format `%Y-%m-%d %H:%M:%S`.

Returns:

(*no returns*)

3.1.23   Method: bExistingResultID

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

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  Result UUID to be verified.

Returns:

- bExisting
  / *Type*: bool /
  True if test result UUID is already existing.

3.1.24   Method: arGetProjectVersionSWByID

Get the project and version_sw information of given test_result_id

Arguments:

- _tbl_test_result_id
  / *Condition*: required / *Type*: str /
  Result UUID to be get the information.

Returns:

- / *Type*: tuple /
  None if test result UUID is not existing, else the tuple which contains project and version_sw: (project, variant)
  is returned.

Chapter 4

pytestlog2db.py

4.1   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.

4.2   Function: validate_xml_result

Verify the given xml result file is valid or not.
Arguments:

- `xml_result`
  / *Condition*: required / *Type*: str /
  Path to PyTest result file.

- `xsd_schema`
  / *Condition*: optional / *Type*: str / *Default*: <installed_folder>/xsd/junit.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.

4.3   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.

4.4   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.

4.5 Function: parse_pytest_xml

Parse and merge all given pytest *.xml result files into one result file.Besides, starttime and endtime are also calculated and added in the merged result.

Arguments:

- `xmlfiles`

  / *Condition*: required / *Type*: str /

  Path to pytest *.xml result file(s).

Returns:

- `oMergedTree`

  / *Type*: etree._Element object /

  The result object which is parsed from provided pytest *.xml result file(s).

4.6 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.

4.7 Function: get_test_result

Get test result from provided Testcase object.

Arguments:

- `oTest`

  / *Condition*: required / *Type*: etree._Element object /

  Testcase object.

Returns:

  / *Type*: typle /

  Testcase result which contains result_main, lastlog and result_return.

4.8   Function: process_component_info

Return the component name bases on provided testcase's classname and component mapping.

Arguments:

- `dConfig`

  / *Condition*: required / *Type*: dict /

  Configuration which contains the mapping between component and testcase's classname.

- `sTestClassname`

  / *Condition*: required / *Type*: str /

  Testcase's classname to get the component info.

Returns:

- `sComponent`

  / *Type*: typle /

  Component name maps with given testcase's classname. Otherwise, "unknown" will be return as component name.

4.9   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`, ...

Arguments:

- `config_file`

  / *Condition*: required / *Type*: str /

  Path to configuration file.

Returns:

- `dConfig`

  / *Type*: dict /

  Configuration object.

4.10 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*: etree._Element 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.

- component_name
  / *Condition*: required / *Type*: str /
  Component name which this test case is belong to.

- test_number
  / *Condition*: required / *Type*: int /
  Order of test case in file.

- start_time
  / *Condition*: required / *Type*: datetime object /
  Start time of testcase.

Returns:

/ *Type*: float /
Duration (in second) of test execution.

4.11 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*: etree._Element object /
  Robot suite object.

- ˍtblˍtestˍresultˍid

  / *Condition*: required / *Type*: str /

  UUID of test result for importing.

- dConfig

  / *Condition*: required / *Type*: dict / *Default*: None /

  Configuration data which is parsed from given json configuration file.

Returns:

(*no returns*)

4.12   Function: PyTestLog2DB

Import pytest results from `*.xml` file(s) to TestResultWebApp's database.

Flow to import PyTest results to database:

1. Process provided arguments from command line.

2. Parse PyTest 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.

Returns:

(*no returns*)

4.13   Class: Logger

Imported by:

```
from PyTestLog2DB.pytestlog2db import Logger
```

Logger class for logging message.

4.13.1   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.

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

- `dryrun`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If set, a prefix as 'dryrun' is added for all messages.

Returns:

(*no returns*)

4.13.2   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*)

4.13.3   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*)

4.13.4   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 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	PyTestLog2DB
Version	0.1.9
Date	20.04.2023
Description	Imports pytest result(s) to TestResultWebApp database
Package URL	python-pytestlog2db
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 6

History

0.1.0	11/2022
Initial version	
0.1.1	22.11.2022
Initial implementation of PyTestLog2DB tool	
0.1.2	01.12.2022
Add tool's document	
0.1.4	24.02.2023
- Rename renamed key "components" in configuration json *- Change behaviour of append mode without UUID to raise a fatal error* *- Improve console log for append mode and matched component for testcase* *- Add command line aruguments –versions and –variant*	
0.1.5	27.02.2023
Change behaviour of append mode with not existing UUID to raise a fatal error	
0.1.6	01.03.2023
Add a validation for xmlresultfile with junitxml schema	
0.1.7	10.03.2023
- Add a validation for existing project/variant and version_sw in db when using append mode *- Remove db maxlength handlers: truncations and validation*	
0.1.8	21.03.2023
Support 4 byte characters for importing to db	
0.1.9	20.04.2023
- Enhance console log with test case counter and component statistics *- Add try/except for database access*	