Specification of

RobotFramework AIO

v. 0.5.2

Implemented by:

Thomas Pollerspöck

Nguyen Huynh Tri Cuong
Tran Duy Ngoan
Mai Dinh Nam Son
Tran Hoang Nguyen
Holger Queckenstedt

09.2022

Contents

Chapter 1

RobotFramework AIO environment

During an installation of the RobotFramework AIO all content is placed at two different main locations:

- `RobotFramework`
  Contains the framework installation files itself
- `RobotTest`
  Contains the *playground* for the user, including default folder for testcases and logfiles

The root path to these folders can be set by the user during the installation process (called `<root>` in listings below).

The following overview contains a list of environment variables that are introduced by the installer. They can be used to refer to locations inside these folders.

- `RobotLogPath`
  Log files folder in which the RobotFramework AIO places the log files per default.
  Location: `<root>/RobotTest/logfiles`
- `RobotTestPath`
  Folder in which the user can place the testcases.
  Location: `<root>/RobotTest/testcases`
- `RobotTutorialPath`
  Folder containing a tutorial of the RobotFramework AIO (useful for self learning).
  Location: `<root>/RobotTest/tutorial`
- `RobotPythonPath`
  Folder containing the Python installation.
  Location: `<root>/RobotFramework/python39`
- `RobotScriptPath`
  Folder containing scripts of the Python installation.
  Location: `<root>/RobotFramework/python39/scripts`
- `RobotVsCode`
  Folder containing the installation of *Visual Studio Code* that is intended to be the main development environment for the RobotFramework AIO.
  Location: `<root>/RobotFramework/robotvscode`
- `RobotToolsPath`
  Tools which require and extend functionality of the Robot Framework.
  Location: `<root>/RobotFramework/tools`
- `RobotDevtools`
  Tools which are independent of Robot Framework, but are bundeled and tested with RobotFramework AIO.
  Location: `<root>/RobotFramework/devtools`

- `RobotAppium`
  Location of Appium tested and distributed with RobotFramework AIO. More information about Appium you find here: appium
  Location: `<root>/RobotFramework/devtools/Windows/Appium`

- `RobotNodeJS`
  Root folder of `nodjs` tested and distributed with RobotFramework AIO. More information about `nodjs` you you find here: nodejs.org
  Location: `<root>/RobotFramework/devtools/Windows/nodejs`

Chapter 2

Apertis Pro

2.1 How to use DLT

2.1.1 dlt-daemon on Apertis

Verify that **dlt-daemon** has installed on Apertis Pro target or not.
`systemctl status dlt-daemon`

In case **dlt-daemon** is not available, follow below steps to install and start dlt-daemon service:

- Install **dlt-daemon** package
  `sudo apt install dlt-daemon`
- Start **dlt-daemon** service
  `sudo systemctl start dlt-daemon`

2.1.2 Apertis Pro firewall configuration

In order to capture DLT log/trace from DLT client(**DLT Viewer**, **DLTConnector**), DLT client has to communicate with Apertis Pro (TCP/IP protocol) via port **3490** (as default).
So that, this connection should be allowed on Apertis Pro target.

Adopt settings of firewall at Apertis Pro:

- Add new rule to allow DLT service at port **3490** (as default)
  Edit `/etc/iptables/rules.v4` file to add below line

```
...
# Accept dlt for development
-A INPUT -p tcp -m state --state NEW -m tcp --dport 3490 -j ACCEPT
...
```

- Restart the firewall with changed parameters
  `sudo systemctl restart iptables.service`

2.1.3 DLTSelfTestApp

DLTSelfTestApp is an application which will be run on the Apertis Pro target for testing the DLT connection between RobotFramework AIO and target.
This package is a part of RobotFramework AIO selftest helpers.
To install **DLTSelfTestApp**, download its debian package on Apertis Pro target then execute the below command.
`sudo dpkg -i <path/to/dltselftestapp_1.0.0_amd64.deb>`

DLTSelfTestApp application will be installed in `/opt/bosch/robfw/dlt` directory and can be started with below command:

`/opt/bosch/robfw/dlt/DLTSelfTestApp`

Welcome log message `Welcome to RobotFramework AIO DLTSelfTestApp...` will be sent at application startup.
Then the ping log `ping message from RobotFramework AIO DLTSelfTestApp` every 5 seconds.

DLT command injection:
To perform the DLT command injection, use below information:

- App ID: **RBFW**
- Context ID: **TEST**
- Service ID: **0x1000**
- Data as Textdata

DLT log reponse of **DLTSelfTestApp** will bases on injected command:

- `welcome` : DLT reponse as above welcome message.
- `exit` : DLT reponse as `Bye...` then **DLTSelfTestApp** will be terminated.
- Other commands: DLT reponse as combination of data and string.
  e.g: `Data: 000000: 77 65 6c 63 6f 6d 65 31 32 31 32 00 xx xx xx xx welcome1212`

2.1.4 QConnectDLTLibrary

QConnectDLTLibrary is part of RobotFramework AIO. It provides the ability for handling connection to Diagnostic Log and Trace(DLT) Module. The library support for getting trace message and sending trace command

Sample RobotFramework AIO testcase which are using **QConnectDLTLibrary** to test DLTSelfTestApp on Apertis Pro target:

- Header of a RobotFramework AIO testcase containing common settings (`*** Settings ***`) like the setups and teardowns, the definition of variables (`*** Variables ***`) and the definition of keywords (`*** Keywords ***`):

```
*** Settings ***
Documentation  This is selftest for DLT connection with DLTSelfTestApp
Library     QConnectionLibrary.ConnectionManager
Suite Setup     Open Connection
Suite Teardown  Close Connection

*** Variables ***
${CONNECTION_NAME}  TEST_CONN_DLTSelfTestApp
${DLT_CONNECTION_CONFIG} =  SEPARATOR=
...  {
...      "gen3flex@DLTLSIMWFH": {
...            "target_ip": "127.0.0.1",
...            "target_port": 4490,
...            "mode": 0,
...            "ecu": "ECU1",
...            "com_port": "COM1",
...            "baudrate": 115200,
...            "server_ip": "localhost",
...            "server_port": 1234
...      }
...  }

*** Keywords ***
Close Connection
   disconnect  ${CONNECTION_NAME}
   Log to console    \nDLT connection has been closed!

Open Connection
```

```
    ${dlt_config} =    evaluate    json.loads('''${DLT_CONNECTION_CONFIG}''')    json

    connect             conn_name=${CONNECTION_NAME}
    ...                 conn_type=DLT
    ...                 conn_mode=dltconnector
    ...                 conn_conf=${dlt_config}

    Log to console    \nDLT connection has been opened successfully!
```

- Sample RobotFramework AIO testcase to verify the ping message from DLTSelfTestApp

```
*** Test Cases ***
Match log/trace from DLTSelfTestApp
    [Documentation]   Match log/trace from DLTSelfTestApp
    [Tags]   DLTSelfTestApp
    ${res}=    verify     conn_name=${CONNECTION_NAME}
    ...                   search_pattern=(DLT:0x01.*RBFW.*)
    ...                   timeout=6    # DLTSelfTestApp pings a message every 5 seconds

    # log to console     \n${res}[0]
    # verify that reponse message should contain "Ping" keyword
    Should Match Regexp     ${res}[0]    DLT:0x01.*RBFW.*Ping.*
```

- Sample RobotFramework AIO testcase to verify command injection with DLTSelfTestApp

```
Command injection with DLTSelfTestApp
    [Documentation]   Get log/trace from DLTSelfTestApp
    [Tags]   DLTSelfTestApp
    ${res}=    verify     conn_name=${CONNECTION_NAME}
    ...                   search_pattern=(DLT:0x01.*RBFW.*Welcome.*)
    ...                   send_cmd=DLT_CALL_SW_INJECTION_ECU ECU1 1000 RBFW TEST ↩
    ↪ 'welcome'

    # log to console     \n${res}[0]

    ${res}=    verify     conn_name=${CONNECTION_NAME}
    ...                   search_pattern=(DLT:0x01.*RBFW.*other_cmd.*)
    ...                   send_cmd=DLT_CALL_SW_INJECTION_ECU ECU1 1000 RBFW TEST ↩
    ↪ 'other_cmd'

    # log to console     \n${res}[0]
```

Please refer QConnectDLTLibrary repository for more details about usage and other example testcase for DLT connection.

Chapter 3

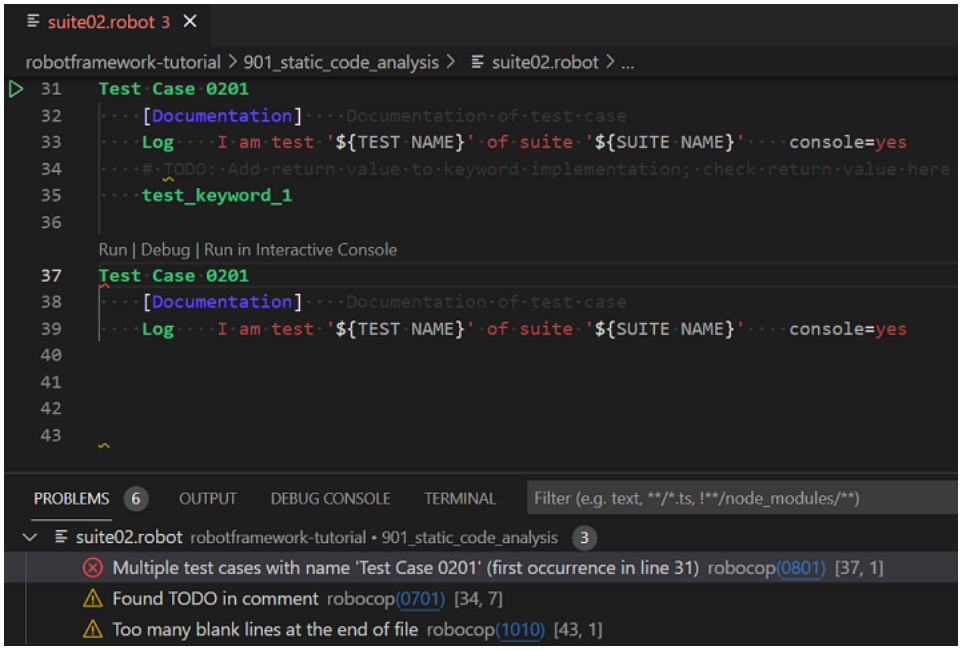
Code analysis

The RobotFramework AIO installation provides a static code analyser to detect potential errors and violations to coding conventions. The name of the analyser is **Robocop**.

Robocop is integrated in Visual Studio Code and and is triggered automatically in case of

1. a file is opened in editor,
2. a file is changed and saved.

The outcome will look like this:



Further examples can be found in the RobotFramework AIO tutorial, chapter `901_static_code_analysis` .

How to configure?

Robocop contains a set of rules that are used to check the source files of the RobotFramework AIO. Not all of them are really useful and some of them are also against our company internal development rules. Therefore we have the need to exclude rules from Robocop execution.

Some rules can be configured (e.g. the rule checking the maximum number of characters within a line of code). Also here we have the need for careful adaptions.

When Robocop is triggered automatically within Visual Studio Code, then the configuration is done with the help of a `.robocop` file. This file has to be placed within the projects folder (see tutorial). To make Robocop results comparable this configuration file should not be modified - or at least should be kept aligned with the version all other project team members use.

Command line

In addition to the automated triggering within Visual Studio Code Robocop can also be called in command line. This is useful

1. in case of a complete folder has to be checked in one step (to avoid the need to open every single source file inside separately),
2. in case of an alternative configuration is wanted (temporarily; to avoid to manipulate the standard configuration file),
3. in case of Robocop shall be triggered by automats like Jenkins,
4. in case of the Robocop results are needed within a log file (output to a log file needs to be configured separately).

To give it a try make a copy of the configuration file `.robocop` available within the tutorial, and give this copy any other name (e.g. `robocop.arg`).
Now Robocop can be called with command lines like this:

1.) Simple check of single file with default configuration:

```
"%RobotPythonPath%/python.exe" -m robocop "⟨path⟩/mytestfile.robot"
```

2.) Check of single file with individual configuration taken from argument file:

```
"%RobotPythonPath%/python.exe" -m robocop --argumentfile "⟨path⟩/robocop.arg" ↩
    ↪ "⟨path⟩/mytestfile.robot"
```

3.) Check of entire folder with individual configuration taken from argument file:

```
"%RobotPythonPath%/python.exe" -m robocop --argumentfile "⟨path⟩/robocop.arg" "⟨folder⟩"
```

How to activate?

In Visual Studio Code the automated triggering of Robocop is activated per default within

```
RobotFramework\robotvscode\data\user-data\User\settings.json
```

with the following switches set to `true` :

```
"robot.lint.enabled": true,
"robot.lint.robocop.enabled": true,
```

Chapter 4

Library documentation

The following sections contain the documentation of additional libraries that are part of the RobotFramework AIO.

RobotFramework AIO bundle

RobotFramework AIO
Version 0.5.2 (from 09.2022)

Included libraries

GenPackageDoc
Version 0.38.0 (from 30.11.2022)
https://github.com/test-fullautomation/python-genpackagedoc
Documentation builder for Python packages

PythonExtensionsCollection
Version 0.11.3 (from 21.11.2022)
https://github.com/test-fullautomation/python-extensions-collection
Additional Python functions

RobotframeworkExtensions
Version 0.8.5 (from 21.11.2022)
https://github.com/test-fullautomation/robotframework-extensions-collection
Additional Robot Framework keywords

JsonPreprocessor
Version 0.1.4 (from 14.09.2022)
https://github.com/test-fullautomation/python-jsonpreprocessor
Preprocessor for json files

RobotFramework_Testsuites
Version 0.2.2 (from 18.07.2022)
https://github.com/test-fullautomation/robotframework-testsuitesmanagement
Functionality to manage RobotFramework testsuites

QConnectBase
Version 1.1.1 (from 12.09.2022)
https://github.com/test-fullautomation/robotframework-qconnect-base
Robot Framework test library for TCP, SSH, serial connection

RobotLog2RQM
Version 1.1.4 (from 10.11.2022)
https://github.com/test-fullautomation/robotframework-robotlog2rqm
Imports robot result(s) to IBM Rational Quality Manager (RQM)

TMLLog2RobotLog
Version 1.2.0 (from 29.08.2022)
https://sourcecode.socialcoding.bosch.com/projects/ROBFW/repos/robotframework-tmllog2robotlog/browse
Convert TML-Framework results to Robot Framework results

RobotLog2DB
Version 1.2.4 (from 18.11.2022)
https://github.com/test-fullautomation/robotframework-robotlog2db
Imports robot result(s) to TestResultWebApp database

PyTestLog2DB
Version 0.1.1 (from 22.11.2022)
https://github.com/test-fullautomation/python-pytestlog2db
Imports pytest result(s) to TestResultWebApp database

TestResultWebApp
Version 0.1.3 (from 18.10.2022)
https://github.com/test-fullautomation/testresultwebapp
Web based display of test results

4.1 GenPackageDoc

GenPackageDoc

v. 0.38.0

Holger Queckenstedt

30.11.2022

Contents

Chapter 1

Introduction

The Python package **GenPackageDoc** generates the documentation of Python modules. The content of this documentation is taken out of the docstrings of functions, classes and their methods. All docstrings have to be written in reStructuredText (RST) format, that is a certain markdown dialect.

It is possible to extend the documentation by the content of additional files either in reStructuredText format or in LaTeX format.

The documentation is generated in four steps:

1. Files in LaTeX format are taken over immediately.
2. Files in reStructuredText format are converted to LaTeX files.
3. All docstrings of all Python modules in the package are converted to LaTeX files.
4. All LaTeX files together are converted to a single PDF document. This requires a separately installed LaTeX distribution (recommended: TeX Live). A LaTeX distribution is **not** part of **GenPackageDoc** and has to be installed separately!

The sources of **GenPackageDoc** are available in the following GitHub repository:

python-genpackagedoc

The repository **python-genpackagedoc** uses it's own functionality to document itself and the contained Python package **GenPackageDoc**.

Therefore the complete repository can be used as an example about writing a package documentation.

It has to be considered, that the main goal of **GenPackageDoc** is to provide a toolchain to generate documentation out of Python sources that are stored within a repository, and therefore we have dependencies to the structure of the repository. For example: Configuration files with values that are specific for a repository, should not be installed. Such a specific configuration value is e.g. the name of the package or the name of the PDF document.

The impact is: There is a deep relationship between the repository containing the sources to be documented, and the sources and the configuration of **GenPackageDoc** itself. Therefore some manual preparations are necessary to use **GenPackageDoc** also in other repositories.

How to do this is explained in detail in the next chapters.

The outcome of all preparations of **GenPackageDoc** in your own repository is a PDF document like the one you are currently reading.

Chapter 2

Description

2.1 Repository content

What is the content of the repository **python-genpackagedoc**?

- Folder `GenPackageDoc`

  Contains the package code.

  This folder is specific for the package.

- Folder `config`

  Contains the repository configuration (e.g. the name of the package, the name of the repository, the author, and more ...).

  This folder is specific for the repository.

- Folder `additions`

  Contains additionally needed sources like setup related class definitions and sources, that are imported from other repositories - to make this repository stand alone

- Folder `packagedoc`

  Contains all package documentation related files, e.g. the **GenPackageDoc** configuration, additional input files and the generated documentation itself.

  This folder is specific for the documentation.

- Repository root folder

  - `genpackagedoc.py`
    Python script to start the documentation build
  - `setup.py`
    Python script to install the package sources. This includes the execution of `genpackagedoc.py`. Therefore building the documentation is part of the installation process.
  - `dump_repository_config.py`
    Little helper to dump the repository configuration to console
  - `readme.rst2md.py`
    Little helper to convert the RST version of the README file to MD format separately (`setup.py` also does this).

2.2 Documentation build process

How do the files and folders listed above, belong together? What is the way, the information flows when the documentation is generated?

- The process starts with the execution of `genpackagedoc.py` within the repository root folder.

  `genpackagedoc.py` can be used stand alone - but this script is also called by `setup.py`. The impact is that every installation includes an update of the documentation.

- `genpackagedoc.py` creates a repository configuration object

  `config/CRepositoryConfig.py`

- The repository configuration object reads the static repository configuration values out of a separate json file

  `config/repository_config.json`

- The repository configuration object adds dynamic values (like operating system specific settings and paths) to the repository configuration. Not all of them are required for the documentation build process, but the repository configuration also supports the setup process (`setup.py`).

  There is one certain setting in the repository configuration file

  `config/repository_config.json`,

  that is essential for the documentation build process:

  `"PACKAGEDOC" : "./packagedoc"`

  This is the path to a folder, in which all further documentation related files are placed. In case of the path is relative, the reference is the position of `genpackagedoc.py`. It is required that within this folder the configuration file for the documentation build process

  `packagedoc_config.json`

  can be found. The name of this json file is fix!

- The configuration file `packagedoc_config.json` contains settings like

  - Paths to Python packages to be documented
  - Paths and names of additional RST files
  - Path and name of output folder (LaTeX files and output PDF file)
  - User defined parameter (that can be defined here as global runtime variables and can be used in any RST code)
  - Basic settings related to the output PDF file (like document name, name of author, ...)
  - Path to LaTeX compiler
    (*a LaTeX distribution is not part of* **GenPackageDoc**)

  Be aware of that the within `packagedoc_config.json` specified output folder

  `"OUTPUT" : "./build"`

  will be deleted at the beginning of the documentation build process! Make sure that you do not have any files inside this folder opened when you start the process. In case of the path is relative, the reference is the position of `genpackagedoc.py`. The complete path is created recursively.

  Further details are explained within the json file itself.

- `genpackagedoc.py` also creates an own configuration object

  `GenPackageDoc/CPackageDocConfig.py`

  `CPackageDocConfig.py` takes over all repository configuration values, reads in the static **GenPackageDoc** configuration (`packagedoc_config.json`) and adds dynamically computed values like the full absolute paths belonging to the documentation build process. Also all command line parameters are resolved and checked.

  The reference for all relative paths is the position of `genpackagedoc.py` (that is the repository root folder).

After the execution of `genpackagedoc.py` the resulting PDF document can be found under the specified name within the specified output folder (`"OUTPUT"`). This folder also contains all temporary files generated during the documentation build process.

Because the output folder is a temporary one, the PDF document is copied to the folder containing the package sources and therefore is included in the package installation. This is defined in the **GenPackageDoc** configuration, section `"PDFDEST"`.

Command line

Some configuration parameter predefined within `packagedoc_config.json`, can be overwritten in command line.

`--output`

Path and name of folder containing all output files.

`--pdfdest`

Path and name of folder in which the generated PDF file will be copied to (after this file has been created within the output folder).

Caution: The generated PDF file will per default be copied to the package folder within the repository. This is defined in `packagedoc_config.json`. The version of the PDF file within the package folder will be part of the installation (when using `setup.py`). When you change the PDF destination, then you get this file at another location - but this file will not be part of the installation any more. Installed will be the version, that is still present within the package folder of the repository. Please try to get the bottom of your motivation when you change this setting.

`--configdest`

Path and name of folder in which a dump of the current configuration will be copied to.

The configuration dump is part of the build output (section 'OUTPUT') and available in txt and in json format. It might be useful for further processes to have access to all details regarding the current documentation build.

--strict

If `True`, a missing LaTeX compiler aborts the process, otherwise the process continues.

--simulateonly

If `True`, the LaTeX compiler is switched off. No new PDF output will be generated. Already existing PDF output will not be updated. This is not handled as error and also not handled as warning. Only the source files will be parsed. This switch is useful to do a pre check for possible syntax issues within the source files without spending time for rendering PDF files.

Example

```
genpackagedoc.py --output="../any/other/location" --pdfdest="../any/other/location"
↪  --configdest="../any/other/location" --strict=True
```

All listed parameters are optional. **GenPackageDoc** creates the complete output path (`--output`) recursively. Other destination folder (`--pdfdest` and `--configdest`) have to exist already.

2.3 PDF document structure

How is the resulting PDF document structured? What causes an entry within the table of content of the PDF document?

In the following we use terms taken over from the LaTeX world: *chapter*, *section* and *subsection*.

A *chapter* is the top level within the PDF document; a *section* is the level below *chapter*, a *subsection* is the level below *section*.

The following assignments happen during the generation of a PDF document:

- The content of every additionally included separate RST file is a *chapter*.
  - In case of you want to add another chapter to your documentation, you have to include another RST file.
  - The headline of the chapter is the name of the RST file (automatically).
    Therefore the heading within an RST file has to start at section level!
- The content of every included Python module is also a *chapter*.
  - The headline of the chapter is the name of the Python module (automatically).
    This means also that within the PDF document structure every Python module is at the same level as additionally included RST files.
- Within additionally included separate RST files sections and subsections can be defined by the following RST syntax elements for headings:
  - A line underlined with "=" characters is a section
  - A line underlined with "-" characters is a subsection
- Within the docstrings of Python modules the headings are added automatically (for functions, classes and methods)
  - Classes and functions are listed at section level (both classes and functions are assumed to be at the same level).
  - Class methods are listed at subesction level.

  Further nestings of headings are not supported (because we do not want to overload the table of content).

2.4 Examples

2.4.1 Example 1: RST file

The text of this chapter is taken over from an RST file named `Description.rst`.

This RST file contains the following headlines:

```
Repository content
==================

Documentation build process
===========================

PDF document structure
======================

Examples
========

Example 1: RST file
-------------------

Example 2: Python module
------------------------
```

Because `Description.rst` is the second imported RST file, the chapter number is 2. The chapter headline is "Description" (the name of the RST file). The top level headlines *within* the RST file are at *section* level. The fourth section (Examples) contains two subsections.

The outcome is the following part of the table of content:

2.4.2 Example 2: Python module

Part of this documentation is a Python module with name `CDocBuilder.py` (listed in table of content at *chapter* level). This module contains a class with name `CDocBuilder` (listed in table of content at *section* level). The class `CDocBuilder` contains a method with name `Build` (listed in table of content at *subsection* level).

This causes the following entry within the table of contents:

2.5 Interface and module descriptions

How to describe an interface of a function or a method? How to describe a Python module?

To have a unique look and feel of all interface descriptions, the following style is recommended:

Example

```
    """
Description of function or method.

**Arguments:**

* ``input_param_1``

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

  Description of input_param_1.

* ``input_param_2``

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

  Description of input_param_2.

**Returns:**

* ``return_param``

  / *Type*: str /

  Description of return_param.
    """
```

Some of the special characters used within the interface description, are part of the RST syntax. They will be explained in one of the next sections.

The docstrings containing the description, have to be placed directly in the next line after the `def` or `class` statement.

It is also possible to place a docstring at the top of a Python module. The exact position doesn't matter - but it has to be the first constant expression within the code. Within the documentation the content of this docstring is placed before the interface description and should contain general information belonging to the entire module.

The usage of such a docstring is an option.

2.6 Runtime variables

What are "runtime variables" and how to use them in RST text?

All configuration parameters of **GenPackageDoc** are taken out of four sources:

1. the static repository configuration
   `config/repository_config.json`
2. the dynamic repository configuration
   `config/CRepositoryConfig.py`
3. the static **GenPackageDoc** configuration
   `packagedoc/packagedoc_config.json`
4. the dynamic **GenPackageDoc** configuration
   `GenPackageDoc/CPackageDocConfig.py`

Some of them are runtime variables and can be accessed within RST text (within docstrings of Python modules and also within separate RST files).

This means it is possible to add configuration values automatically to the documentation.

This happens by encapsulating the runtime variable name in triple hashes. This "triple hash" syntax is introduced to make it easier to distinguish between the json syntax (mostly based on curly brackets) and additional syntax elements used within values of json keys.

The name of the repository e.g. can be added to the documentation with the following RST text:

```
The name of the repository is ###REPOSITORYNAME###.
```

This document contains a chapter "Appendix" at the end. This chapter is used to make the repository configuration a part of this documentation and can be used as example.

Additionally to the predefined runtime variables a user can add own ones.
See `"PARAMS"` within `packagedoc_config.json`.

All predefined runtime variables are written in capital letters. To make it easier for a developer to distinguish between predefined and user defined runtime variables, all user defined runtime variables have to be written in small letters completely.

Also the `"DOCUMENT"` keys within `packagedoc_config.json` are runtime variables.

Also within `packagedoc_config.json` the triple hash syntax can be used to access repository configuration values.

With this mechanism it is e.g. possible to give the output PDF document automatically the name of the package:

```
"DOCUMENT" : {
                "OUTPUTFILENAME" : "###PACKAGENAME###.tex",
```

Within parts of the documentation that are written in LaTeX directly, two auto generated LaTeX commands can be used to insert the name of the repository and the name of the package. Both values are taken out of the repository configuration.

1. `\repo` : name of the repository
2. `\pkg` : name of the package

Example:

```
The repository \repo\ contains the package \pkg.
```

Consider the trailing backslash at the end of the command (that together with the following blank indicates a masked blank). This is necessary when you use the command in the middle of a text.

2.7 Syntax aspects

2.7.1 Common rules

Important to know about the syntax of Python and RST is:

- In both Python and RST the indentation of text is part of the syntax!
- The indentation of the triple quotes indicating the beginning and the end of a docstring has to follow the Python syntax rules.
- The indentation of the content of the docstring (= the interface description in RST format) has to follow the RST syntax rules. To avoid a needless indentation of the text within the resulting PDF document and to avoid further unwanted side effects caused by improper indentations, it is strongly required to start at least the first line of a docstring text within the first column! And this first line is the reference for the indentation of further lines of the current docstring. The indentation of these further lines depends on the RST syntax element that is used here.
- In RST also blank lines are part of the syntax!

Why is a proper indentation of the docstrings so much important?

The contents of all doctrings of a Python module will be merged to one single RST document (internally by **GenPackageDoc**). In this single RST document we do not have separated docstring lines any more. We have one text! And we have a relationship between previous lines and following lines in this text. The indentation of these previous and following lines must fit together – accordingly to the RST syntax rules. Otherwise we either get syntax issues during computation or we get text with a layout that does not fit to our expectation.

Please be attentive while typing your documentation in RST format!

2.7.2 Syntax extensions

GenPackageDoc extends the RST syntax by the following topics:

- *newline*

  A newline (line break) is realized by a slash ('/') at the end of a line containing any other RST text (this means: the slash must **not** be the only character in line).
  Internally this slash is mapped to the LaTeX command `\newline`.

- *vspace*

  An additional vertical space (size: the height of the 'x' character - depending on the current type and size of font) is realized by a single slash ('/'). This slash must be the only character in line!
  Internally this slash is mapped to the LaTeX command `\vspace{1ex}`.

- *newpage*

  A newpage (page break) is realized by a double slash ('//'). These two slashes must be the only characters in line!
  Internally this double slash is mapped to the LaTeX command `\newpage`.

These syntax extensions can currently be used in separate RST files only and are not available within docstrings of Python modules.

Chapter 3

CDocBuilder.py

Python module containing all methods to generate tex sources.

3.1 Class: CDocBuilder

Imported by:

```
from GenPackageDoc.CDocBuilder import CDocBuilder
```

Main class to build tex sources out of docstrings of Python modules and separate text files in rst format.

Depends on a json configuration file, provided by a `oPackageDocConfig` object (this includes the Repository configuration).

Method to execute: `Build()`

3.1.1 Method: Build

Arguments:

(*no arguments*)

Returns:

- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method `sMethod` was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method `sMethod`.

Chapter 4

CInterface.py

Python module containing an interface for **GenPackageDoc**. This interface can be used to get access to the LaTeX stylesheets that are part of the **GenPackageDoc** installation.

4.1 Class: CInterface

Imported by:

```
from GenPackageDoc.CInterface import CInterface
```

4.1.1 Method: GetLaTeXStyles

The LaTeX stylesheets are part of the installation of **GenPackageDoc**. In case of anyone else than **GenPackageDoc** needs these stylesheets, this method can be used to copy them to any other folder.

Arguments:

- `sDestination`
  / *Condition*: required / *Type*: str /
  Path and name of a folder in which the styles folder from **GenPackageDoc** will be copied.

Returns:

- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method `sMethod` was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method `sMethod`.

Chapter 5

CPackageDocConfig.py

Python module containing the configuration for **GenPackageDoc**. This includes the repository configurantion and command line values.

5.1 Class: CPackageDocConfig

Imported by:

```
from GenPackageDoc.CPackageDocConfig import CPackageDocConfig
```

5.1.1 Method: PrintConfig

Prints all cofiguration values to console.

5.1.2 Method: PrintConfigKeys

Prints all cofiguration key names to console.

5.1.3 Method: Get

Returns the configuration value belonging to a key name.

5.1.4 Method: GetConfig

Returns the complete configuration dictionary.

Chapter 6

CPatterns.py

Python module containing source patterns used to generate the tex file output.

6.1 Class: CPatterns

Imported by:

```
from GenPackageDoc.CPatterns import CPatterns
```

The `CPatterns` class provides a set of LaTeX source patterns used to generate the tex file output.

All source patterns are accessible by corresponding `Get` methods. Some source patterns contain placeholder that will be replaced by input parameter of the `Get` method.

6.1.1 Method: GetHeader

Defines the header of the main tex file.

Arguments:

- `sTitle`
  / *Condition*: required / *Type*: str /
  The title of the output document (name of the described package)
- `sVersion`
  / *Condition*: required / *Type*: str /
  The version of the output document (version of the described package)
- `sAuthor`
  / *Condition*: required / *Type*: str /
  The author of the output document (author of the described package)
- `sDate`
  / *Condition*: required / *Type*: str /
  The date of the output document (date of the described package)

Returns:

- `sHeader`
  / *Type*: str /
  LaTeX code containing the header of main tex file.

6.1.2 Method: GetChapter

Defines single chapter of the main tex file.

A single chapter is equivalent to an additionally imported text file in rst format or equivalent to a single Python module within a Python package.

Arguments:

- `sHeadline`
  / *Condition*: required / *Type*: str /
  The chapter headline (that is either the name of an additional rst file or the name of a Python module).
- `sLabel`
  / *Condition*: required / *Type*: str /
  The chapter label (to enable linking to this chapter)
- `sDocumentName`
  / *Condition*: required / *Type*: str /
  The name of a single tex file containing the chapter content. This file is imported in the main text file after the chapter headline that is set by `sHeadline`.

Returns:

- `sHeader`
  / *Type*: str /
  LaTeX code containing the headline and the input of a single tex file.

6.1.3 Method: GetFooter

Defines the footer of the main tex file.

Arguments:

(*no arguments*)

Returns:

- `sFooter`
  / *Type*: str /
  LaTeX code containing the footer of the main tex file.

6.1.4 Method: GetAutodefinedHeader

Defines the header of the autodefined LaTeX sty file.

Arguments:

(*no arguments*)

Returns:

- `sAutodefinedHeader`
  / *Type*: str /
  LaTeX code containing the header of the autodefined LaTeX sty file.

Chapter 7

CSourceParser.py

Python module containing all methods to parse the documentation content of Python source files.

7.1 Class: CSourceParser

Imported by:

```
from GenPackageDoc.CSourceParser import CSourceParser
```

The `CSourceParser` class provides a method to parse the functions, classes and their methods together with the corresponding docstrings out of Python modules. The docstrings have to be written in rst syntax.

7.1.1 Method: ParseSourceFile

The method `ParseSourceFile` parses the content of a Python module.

Arguments:

- `sFile`
  / *Condition*: required / *Type*: str /
  Path and name of a single Python module.
- `bIncludePrivate` (currently not active, is `False`)
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `False`: private methods are skipped, otherwise they are included in documentation.
- `bIncludeUndocumented`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If `True`: also classes and methods without docstring are listed in the documentation (together with a hint that information is not available), otherwise they are skipped.

Returns:

- `dictContent`
  / *Type*: dict /
  A dictionary containing all the information parsed out of `sFile`.
- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method `sMethod` was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method `sMethod`.

Chapter 8

Appendix

About this package:

Table 8.1: Package setup

Setup parameter	Value
Name	GenPackageDoc
Version	0.38.0
Date	30.11.2022
Description	Documentation builder for Python packages
Package URL	python-genpackagedoc
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.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 :: 3 - Alpha
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 9

History

History of **GenPackageDoc** (hosted in repository **`python-genpackagedoc`**).

0.1.0	04/2022
Initial version	
0.2.0	05.05.2022
Python syntax highlighting within code blocks added	
0.3.0	06.05.2022
Automated headings for functions, classes and methods	
0.4.0	06.05.2022
Possibility to describe complete Python modules added	
0.5.0	09.05.2022
Parameter `INCLUDEPRIVATE` added	
0.6.0	09.05.2022
Parameter `INCLUDEUNDOCUMENTED` added	
0.7.0	10.05.2022
Setup process introduced and README.rst added; code maintenance	
0.8.0	10.05.2022
Bugfixes and code maintenance; history added	
0.9.0	10.05.2022
Layout maintenance and syntax extensions for `newline`, `newpage` and `vspace` reworked	
0.9.1	11.05.2022
Documentation maintenance	
0.9.2	16.05.2022
Fix: automated line breaks within code blocks	
0.10.0	17.05.2022
Postprocessing for rst and tex sources added; 'multiply-defined labels' fix	
0.11.0	18.05.2022
Import of tex files enabled	
0.12.0	19.05.2022
- Admonitions added, based on LaTeX environment `tcolorbox` *- Layout adaptions in titlepage* *- Page numbering fix in TOC*	
0.13.0	24.05.2022

LaTeX style definitions moved to separate folder	
0.14.0	27.05.2022
- LaTeX compiler check added *- Control parameter `STRICT` added to `packagedoc_config`*	
0.15.0	31.05.2022
- Command line added *- Separate **GenPackageDoc** configuration class added*	
0.16.0	01.06.2022
Path computation reworked	
0.17.0	17.06.2022
- Configuration dump added *- Code maintenance* *- Error handling extended*	
0.18.0	20.06.2022
Added parameter to define an output folder for a dump of final configuration	
0.19.0	28.06.2022
- Method `GetLaTeXStyles` added *- PythonExtensionsCollection updated to version 0.8.0*	
0.20.0	29.06.2022
Document title bugfix: Added missing masking of underlines (required for LaTeX)	
0.21.0	12.07.2022
Separated file `preamble.tex`	
0.22.0	13.07.2022
- Maintenance of `preamble.tex` and `styles` folder *- `setup.py` fix (install tex files also)*	
0.23.0	13.07.2022
Maintenance of `preamble.tex`	
0.24.0	25.07.2022
Maintenance of `robotframeworkaio.sty` (line breaks in listings)	
0.25.0	27.07.2022
Layout maintenance of RobotFramework AIO syntax highlighting (`.sty` files)	
0.26.0	27.07.2022
History reworked; common.sty introduced	
0.27.0	17.08.2022
Added LaTeX style definition for Python syntax highlighting	
0.28.0	23.08.2022
- Introduced new LaTeX environment variable `GENDOC_LATEXPATH` *- `robotframeworkaio.sty` aligned to version in `GenMainDoc`*	
0.29.0	24.08.2022
Changed the way the import path of a module is printed out to PDF file	
0.30.0	31.08.2022
Introduced `simulateonly` mode (command line switch to skip the PDF generation)	
0.31.0	12.09.2022
Fix of import path of a module in PDF file	
0.32.0	16.09.2022

- Added labels at chapter level *- partial rework of label mechanisms*	
0.33.0	19.09.2022
Rework of label mechanism (to enable unique links to functions, classes and methods with names that are not unique over all Python modules within a package)	
0.34.0	07.11.2022
Introduced auto defined LaTeX style file containing mnemotechnical commands to type the repository name and the package name	
0.35.0	16.11.2022
Layout settings of some LaTeX commands adapted: *- Repository name and package name in bold* *- Inline code and inline listings in clearer colors*	
0.36.0	17.11.2022
Brightness of all listings colors reduced to 45% (both text boxes and inline)	
0.37.0	21.11.2022
- LaTeX style adaptions and bugfixes *- Introduced LaTeX commands* `Python_log` *and* `pylog` *- Added* `keyword` *decorator detection* *- Feature 'INCLUDEPRIVATE' temporarily switched off (requires bugfixes)*	
0.38.0	30.11.2022
Removed harming ligatures that were added by Pandoc automatically to LaTeX code in case of multiple minus characters in names	

GenPackageDoc.pdf

Created at 13.12.2022 - 16:28:15

by GenPackageDoc v. 0.38.0

4.2 PythonExtensionsCollection

PythonExtensionsCollection

v. 0.11.3

Holger Queckenstedt

21.11.2022

Contents

Chapter 1

Introduction

The **PythonExtensionsCollection** extends the functionality of Python by some useful functions that are not available in Python immediately.

This covers for example file and folder operations, string operations like normalizing a path and a pretty print method.

The **PythonExtensionsCollection** contains several Python modules and every module has to be imported separately in case of the functions inside are needed.

The sources of the **PythonExtensionsCollection** are available in GitHub.

Informations about how to install the **PythonExtensionsCollection** can be found in the README.

Chapter 2

Description

2.1 Modules

The **PythonExtensionsCollection** contains the following modules:

1. `CComparison`
   Compare two text files based on regular expressions.
   Import: `from PythonExtensionsCollection.Comparison.CComparison import CComparison`
2. `CFile`
   File operations like read, write, copy, move, ...
   Import: `from PythonExtensionsCollection.File.CFile import CFile`
3. `CFolder`
   Folder operations like create, delete, copy, ...
   Import: `from PythonExtensionsCollection.Folder.CFolder import CFolder`
4. `CString`
   String operations like normalize a path, string filter, format results, ...
   Import: `from PythonExtensionsCollection.String.CString import CString`
5. `CUtils`
   Pretty print of Python data types
   Import: `from PythonExtensionsCollection.Utils.CUtils import CTypePrint`

2.2 Methods

Additionally to the interface descriptions in the second part of this document this section contains a more detailed description of some assorted methods together with examples how to use.

2.2.1 String operations with CString

NormalizePath

It's not easy to handle paths - and especially the path separators - independent from the operating system.

Under Linux it is obvious that single slashes are used as separator within paths. Whereas the Windows explorer uses single backslashes. In both operating systems web addresses contains single slashes as separator when displayed in web browsers.

Using single backslashes within code - as content of string variables - is dangerous because the combination of a backslash and a letter can be interpreted as escape sequence - and this is maybe not the effect a user wants to have.

To avoid unwanted escape sequences backslashes have to be masked (by the usage of two of them: `"\\"`). But also this could not be the best solution because there are also applications (like the Windows explorer) that are not able to handle masked backslashes. They expect to get single backslashes within a path.

Preparing a path for best usage within code also includes collapsing redundant separators and up-level references. Python already provides functions to do this, but the outcome (path contains slashes or backslashes) depends on the

operating system. And like already mentioned above also under Windows backslashes might not be the preferred choice.

It also has to be considered that redundant separators at the beginning of an address of a local network resource (like `\\server.com`) and or inside an internet address (like `https:\\server.com`) must **not** be collapsed! Unfortunately the Python function `normpath` does not consider this context.

To give the user full control about the format of a path, independent from the operating system and independent if it's a local path, a path to a local network resource or an internet address, the method `NormalizePath()` provides lot's of parameters to influence the result.

Example 1 (*file system path*)

```
path = r"C:\\subfolder1///../subfolder2\\\\../subfolder3\\"
path = CString.NormalizePath(path)
print(path)
```

Result (*output contains slashes*)

```
C:/subfolder3
```

Example 2 (*file system path*)

```
path = r"C:\\subfolder1///../subfolder2\\\\../subfolder3\\"
path = CString.NormalizePath(path, bWin=True)
print(path)
```

Result (*output contains masked backslashes*)

```
C:\\subfolder3
```

Example 3 (*path of a local network resource*)

```
path = r"\\anyserver.com\\part1//part2\\\\part3/part4"
path = CString.NormalizePath(path)
print(path)
```

Result

```
//anyserver.com/part1/part2/part3/part4
```

Example 4 (*internet address*)

```
path = r"http:\\anyserver.com\\part1//part2\\\\part3/part4"
path = CString.NormalizePath(path)
print(path)
```

Result

```
http://anyserver.com/part1/part2/part3/part4
```

DetectParentPath

The method `DetectParentPath` computes the path to any parent folder inside a given path. Optionally `DetectParentPath` is able to search for files inside the identified parent folder.

In the following examples we assume to have the following file system structure:

```
D:\pathtest\ABC\DEF\GHI\FILE.txt
D:\pathtest\ABC\FILE.txt
D:\pathtest\RST\UVW\XYZ\FILE.txt
```

We are inside folder `GHI` and want to know the path to folder `ABC` .

Python code

```
sStartPath  = r"D:\pathtest\ABC\DEF\GHI"
sFolderName = "ABC"
sDestPath, listDestPaths, sDestFile, listDestFiles, sDestPathParent = ↩
    ↪ CString.DetectParentPath(sStartPath, sFolderName)
print(f"sDestPath       = {sDestPath}")
print(f"listDestPaths   = {listDestPaths}")
print(f"sDestFile       = {sDestFile}")
print(f"listDestFiles   = {listDestFiles}")
print(f"sDestPathParent = {sDestPathParent}")
```

Outcome

```
sDestPath       = D:/pathtest/ABC
listDestPaths   = ['D:/pathtest/ABC']
sDestFile       = None
listDestFiles   = None
sDestPathParent = D:/pathtest
```

`sDestPath` contains the path to `sFolderName` within `sStartPath` . `sDestPathParent` contains the path to the parent folder of `sDestPath` . The meaning of the remaining return parameter are handled in the next examples.

It is possible to search for several folders.

We are inside folder `GHI` and want to know the path to folders `ABC` and `DEF` .

```
sStartPath  = r"D:\pathtest\ABC\DEF\GHI"
sFolderName = "ABC;DEF"
sDestPath, listDestPaths, sDestFile, listDestFiles, sDestPathParent = ↩
    ↪ CString.DetectParentPath(sStartPath, sFolderName)
```

Outcome

```
sDestPath       = D:/pathtest/ABC/DEF
listDestPaths   = ['D:/pathtest/ABC/DEF', 'D:/pathtest/ABC']
sDestFile       = None
listDestFiles   = None
sDestPathParent = D:/pathtest/ABC
```

`sFolderName` is a semicolon separated list of folder names. Accordingly to this list of folder names `listDestPaths` contains the paths to these folders. `DetectParentPath` searches in the path from right to left (from bottom level up to top level). Therefore the folder `DEF` is the first one who is found. The order of elements in `listDestPaths` is not synchronized with the order of folder names in `sFolderName` ! In every case `sDestPath` contains the first element in the list. `sDestPathParent` contains the path to the parent folder of `sDestPath` .

It is possible to search for a file.

We are inside folder `GHI` and want to know the path to folder `DEF` and within this folder we want to know the path to file `FILE.txt` .

```
sStartPath  = r"D:\pathtest\ABC\DEF\GHI"
sFolderName = "DEF"
sFileName   = "FILE.txt"
sDestPath, listDestPaths, sDestFile, listDestFiles, sDestPathParent = ↩
    ↪ CString.DetectParentPath(sStartPath, sFolderName, sFileName)
```

Outcome

```
sDestPath       = D:/pathtest/ABC/DEF
listDestPaths   = ['D:/pathtest/ABC/DEF']
sDestFile       = D:/pathtest/ABC/DEF/GHI/FILE.txt
listDestFiles   = ['D:/pathtest/ABC/DEF/GHI/FILE.txt']
sDestPathParent = D:/pathtest/ABC
```

`listDestPaths` contains a list of all paths to `sFolderName` within `sStartPath` . `sDestPath` contains the first element of `listDestPaths` . `listDestFiles` contains a list of all files with name `sFileName` found within `listDestPaths` . `sDestFile` contains the first element of `listDestFiles` .

A semicolon separated list of file names in `sFileName` (like for `sFolderName`) is not supported.

Providing more than one folder together with a file name may cause overlapping results. But `listDestFiles` will not contain any redundant paths to files.

```
sStartPath  = r"D:\pathtest\ABC\DEF\GHI"
sFolderName = "ABC;DEF"
sFileName   = "FILE.txt"
sDestPath, listDestPaths, sDestFile, listDestFiles, sDestPathParent = ↩
    ↪ CString.DetectParentPath(sStartPath, sFolderName, sFileName)
```

Outcome

```
sDestPath       = D:/pathtest/ABC/DEF
listDestPaths   = ['D:/pathtest/ABC/DEF', 'D:/pathtest/ABC']
sDestFile       = D:/pathtest/ABC/DEF/GHI/FILE.txt
listDestFiles   = ['D:/pathtest/ABC/DEF/GHI/FILE.txt', 'D:/pathtest/ABC/FILE.txt']
sDestPathParent = D:/pathtest/ABC
```

In the last example we go one further level up (`pathtest`). Because of the file search is recursive, also files in parallel trees are found now.

```
sStartPath  = r"D:\pathtest\ABC\DEF\GHI"
sFolderName = "pathtest"
sFileName   = "FILE.txt"
sDestPath, listDestPaths, sDestFile, listDestFiles, sDestPathParent = ↩
    ↪ CString.DetectParentPath(sStartPath, sFolderName, sFileName)
```

Outcome

```
sDestPath       = D:/pathtest
listDestPaths   = ['D:/pathtest']
sDestFile       = D:/pathtest/ABC/DEF/GHI/FILE.txt
listDestFiles   = ['D:/pathtest/ABC/DEF/GHI/FILE.txt', 'D:/pathtest/ABC/FILE.txt', ↩
    ↪ 'D:/pathtest/RST/UVW/XYZ/FILE.txt']
sDestPathParent = D:/
```

StringFilter

During the computation of strings there might occur the need to get to know if this string fulfils certain criteria or not. Such a criterion can e.g. be that the string contains a certain substring. Also an inverse logic might be required: In this case the criterion is that the string does **not** contain this substring.

It might also be required to combine several criteria to a final conclusion if in total the criterion for a string is fulfilled or not. For example: The string must start with the string `prefix` and must also contain either the string `substring1` or the string `substring2` but must also **not** end with the string `suffix`.

This method provides a bunch of predefined filters that can be used singly or combined to come to a final conclusion if the string fulfils all criteria or not.

These filters can be e.g. used to select or exclude lines while reading from a text file. Or they can be used to select or exclude files or folders while walking through the file system. The filters are divided into three different types:

1. Filters that are interpreted as raw strings (called 'standard filters'; no wild cards supported)
2. Filters that are interpreted as regular expressions (called 'regular expression based filters'; the syntax of regular expressions has to be considered)
3. Boolean switches (e.g. indicating if also an empty string is accepted or not)

The input string might contain leading and trailing blanks and tabs. This kind of horizontal space is removed from the input string before the standard filters start their work (except the regular expression based filters).

The regular expression based filters consider the original input string (including the leading and trailing space).

The outcome is that in case of the leading and trailing space shall be part of the criterion, the regular expression based filters can be used only.

It is possible to decide if the standard filters shall work case sensitive or not. This decision has no effect on the regular expression based filters.

The regular expression based filters always work with the original input string that is not modified in any way.

Except the regular expression based filters it is possible to provide more than one string for every standard filter (must be a semikolon separated list in this case). A semicolon that shall be part of the search string, has to be masked in this way: `";"` .

This method returns a boolean value that is `True` in case of all criteria are fulfilled, and `False` in case of some or all of them are not fulfilled.

The default value for all filters is `None` (except `bSkipBlankStrings`). In case of a filter value is `None` this filter has no influence on the result.

In case of all filters are `None` (default) the return value is `True` (except the string itself is `None` or the string is empty and `bSkipBlankStrings` is `True`).

In case of the string is `None` , the return value is `False` , because nothing concrete can be done with `None` strings.

Internally every filter has his own individual acknowledge that indicates if the criterion of this filter is fulfilled or not.

The meaning of *criterion fulfilled* of a filter is that the filter supports the final return value `bAck` of this method with `True` .

The final return value `bAck` of this method is a logical join (`AND`) of all individual acknowledges (except `bSkipBlankStrings` and `sComment` ; in case of their criteria are fulfilled, immediately `False` is returned).

Summarized:

- Filters are used to define *criteria*
- The return value of this method provides the *conclusion* - indicating if all criteria are fulfilled or not

All available filters are described in more detail in the interface description of `StringFilter` . Here we continue with some code examples.

Example 1

`sString` has to start with `sStartsWith` and has to contain `sContains` .

That's true. Therefore `StringFilter` returns `True` .

```
StringFilter(sString           = "Speed is 25 beats per minute",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = "Sp",
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = "beats",
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 2

`sString` must not end with `sEndsNotWith` . But does. Therefore `StringFilter` returns `False` - even in case of other criterions are fulfilled.

```
StringFilter(sString           = "Speed is 25 beats per minute",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = "Sp",
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = "minute",
             sContains         = "beats",
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 3

`sString` must not contain `sContainsNot` . Because `bCaseSensitive` is `True` the spelling does not fit. Therefore `StringFilter` returns `True` .

```
StringFilter(sString           = "Speed is 25 beats per minute",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = None,
             sContainsNot      = "Beats",
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 4

`sString` must contain exactly two digits (postulated by regular expression based `sInclRegEx`). That's true. Therefore `StringFilter` returns `True` .

```
StringFilter(sString           = "Speed is 25 beats per minute",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = None,
             sContainsNot      = None,
             sInclRegEx        = r"\d{2}",
             sExclRegEx        = None)
```

Example 5

`sString` must contain exactly three digits (postulated by regular expression based `sInclRegEx`). That's not true. Therefore `StringFilter` returns `False` - even in case of other criterions are fulfilled.

```
StringFilter(sString           = "Speed is 25 beats per minute",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = "Speed",
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = None,
             sContainsNot      = None,
             sInclRegEx        = r"\d{3}",
             sExclRegEx        = None)
```

Example 6

Leading and trailing spaces are removed from the input string `sString` at the beginning. In this example the result is an empty input string. `bSkipBlankStrings` is set to `True` . In this case `StringFilter` immediately returns `False` and all other filters are ignored.

```
StringFilter(sString           = "     ",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = None,
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 7

The input string `sString` starts with a character that is defined to be a comment character (`sComment`). Therefore `StringFilter` immediately returns `False` and all other filters are ignored.

```
StringFilter(sString           = "# Speed is 25 beats per minute",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = "#",
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = "beats",
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 8

Blanks around search strings (here `sContains` is `"   Alpha "`) are considered, whereas the blanks around the input string are removed before computation. Therefore `"   Alpha "` cannot be found within the (shortened) input string and `StringFilter` returns `False` .

```
StringFilter(sString           = "   Alpha is not beta; and beta is not gamma  ",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = "   Alpha ",
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 9

In case of blanks around search strings have to be considered, a regular expression based filter has to be used (here `sInclRegEx`).

This is possible because the regular expression based filters `sInclRegEx` and `sExclRegEx` work **with the original value** of `sString` , and not with the shortened version with leading and trailing blanks removed! The shortened version is applied to standard filters only.

In this example `StringFilter` returns `True` .

```
StringFilter(sString           = "   Alpha is not beta; and beta is not gamma  ",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = None,
             sContainsNot      = None,
             sInclRegEx        = r"\s{3}Alpha",
             sExclRegEx        = None)
```

Example 10

The meaning of `"beta; and"` in this example is: The criterion is fulfilled in case of either `"beta"` or `" and"` can be found. That's true - but this has nothing to do with the fact, that also this string `"beta; and"` can be found. The semikolon is a separator character and therefore part of the syntax.

Nevertheless `StringFilter` returns `True` .

```
StringFilter(sString           = "Alpha is not beta; and beta is not gamma",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = "beta; and",
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

Example 11

In this example the semicolon is masked with `"\;"` and therefore part of the search string `sContains` - and not part of the syntax any more.

The meaning of `"beta\; not"` in this example is: The criterion is fulfilled in case of `"beta; not"` can be found. That's **not** `True` . Therefore `StringFilter` returns `False` .

```
StringFilter(sString           = "Alpha is not beta; and beta is not gamma",
             bCaseSensitive    = True,
             bSkipBlankStrings = True,
             sComment          = None,
             sStartsWith       = None,
             sEndsWith         = None,
             sStartsNotWith    = None,
             sEndsNotWith      = None,
             sContains         = r"beta\; not",
             sContainsNot      = None,
             sInclRegEx        = None,
             sExclRegEx        = None)
```

2.2.2 File access with CFile

The motivation for the `CFile` module contains two main topics:

1. Extended user control by introducing further parameter for file access functions. With high priority `CFile` enables the user to take care about that nothing existing is overwritten accidently.
2. Hide the file handles und use the mechanism of class variables to avoid access violations independent from the way different operation systems like Windows and Unix are handling this.

This shortens the code, eases the implementation and makes tests (in which this module is used) more stable.

Examples

Define two variables with path and name of test files.

Under Windows:

```
sFile_1 = r"%TMP%\CFile_TestFile_1.txt"
sFile_2 = r"%TMP%\CFile_TestFile_2.txt"
```

Or under Linux:

```
sFile_1 = r"/tmp/CFile_TestFile_1.txt"
sFile_2 = r"/tmp/CFile_TestFile_2.txt"
```

The first class instance:

```
oFile_1 = CFile(sFile_1)
```

`oFile_1` is the instance of a class - *and not the file handle.* The file handle is hidden, the user has nothing to do with it.

Every class instance can work with one single file only (during the complete instance lifetime) and has exclusive access to this file.

No other class instance is allowed to use this file. Therefore the second line in the following code throws an exception:

```
oFile_1_A = CFile(sFile_1)
oFile_1_B = CFile(sFile_1)
```

It's more save to implement in this way:

```
try:
   oFile_1 = CFile(sFile_1)
except Exception as reason:
   print(str(reason))
```

For writing content to files two methods are available: `Write()` and `Append()` .

- Using `Write()` causes the class to open the file for writing (`w`) - in case of the file is not already opened for writing.
- Using `Append()` causes the class to open the file for appending (`a`) - in case of the file is not already opened for appending.

Switching between `Write()` and `Append()` causes an intermediate file handle `close()` internally!

Write some content to file:

```
bSuccess, sResult = oFile_1.Write("A B C")
print(f"sResult oFile_1.Write : '{sResult}' / bSuccess : {bSuccess}")
```

Most of the functions return at least `bSuccess` and `sResult` .

- `bSuccess` is `True` in case of no error occurred.

- `bSuccess` is `False` in case of an error occurred.
- `bSuccess` is `None` in case of a very fatal error occurred - like an exception.
- `sResult` contains details about what happens during computation.

It is possible now to continue with using `oFile_1.Write("...")` ; the content will be appended - as long as the file is still open for writing.

Some functions close the file handle (e.g. `ReadLines()`). Therefore sequences like

```
oFile_1.Write("...")
oFile_1.ReadLines("...")
oFile_1.Write("...")
```

should be avoided - because `Write()` after `ReadLines()` starts the file from scratch and the file content written by the previous `Write()` calls is lost.

For appending content to a file use the function `Append()` .

Append content to file:

```
bSuccess, sResult = oFile_1.Append("A B C")
```

For reading content from a file use the function `ReadLines()` .

Read from file:

```
listLines_1, bSuccess, sResult = oFile_1.ReadLines()
for sLine in listLines_1:
   print(f"{sLine}")
```

Additionally to `bSuccess` and `sResult` the function returnes a list of lines.

Internally `ReadLines()` takes care about:

- Closing the file - in case the file is still opened
- Opening the file for reading
- Reading the content line by line until the end of file is reached
- Closing the file

To avoid code like this

```
for sLine in listLines_1:
   print(f"{sLine}")
```

it is also possible to let `ReadLines()` do this:

```
listLines_1, bSuccess, sResult = oFile_1.ReadLines(bToScreen=True)
```

A function to read only a single line from a file is not available, but it is possible to use some filter parameter of `ReadLines()` to reduce the amount of content already during the file is read. This prevents the user from implementing further loops.

Internally `ReadLines()` uses the string filter method `StringFilter()` . All filter related input parameter of `ReadLines()` and `StringFilter()` are the same.

Let's assume the following:

- The file `sFile_1` contains empty lines
- The file `sFile_1` contains also lines, that are commented out (with a hash (`#`) at the beginning)
- We want `ReadLines()` to skip empty lines and lines that are commented out

This can be imlemented in the following way.

Read a subset of file content:

```
listLines_1, bSuccess, sResult = oFile_1.ReadLines(bSkipBlankLines=True,
                                                   sComment='#')
```

It is a good practice to close file handles as soon as possible. Therefore `CFile` provides the possibility to do this explicitely.

Close a file handle:

```
bSuccess, sResult = oFile_1.Close()
```

This makes sense in case of later again access to this file is needed (the class object `oFile_1` still exists).

Additionally to that the file handle is closed implicitely:

- in case of it is required (e.g. when switching between read and write access),
- in case of the class instance is destoyed.

Therefore an alternative to the `Close()` function is the deletion of the class instance:

```
del oFile_1
```

This makes sense in case of access to this file is not needed any more (therefore we also do not need the class object any more).

It is recommended to prefer `del` (instead of `Close()`) to avoid to keep too much not used objects for a too long length of time in memory.

A file can be copied to another file.

Copy a file:

```
bSuccess, sResult = oFile_1.CopyTo(sFile_2)
```

The destination (`sFile_2` in the example above) can either be a full path and name of a file or the path only.

It makes a difference if the destination file exists or not. The optional parameter `bOverwrite` controls the behavior of `CopyTo()`.

The default is that it is not allowed to overwrite an existing destination file: `bOverwrite` is `False`. `CopyTo()` returns `bSuccess = False` in this case.

In case the user want to allow `CopyTo()` to overwrite existing destination files, it has to be coded explicitely:

```
bSuccess, sResult = oFile_1.CopyTo(sFile_2, bOverwrite=True)
```

A file can be moved to another file.

Move a file:

```
bSuccess, sResult = oFile_1.MoveTo(sFile_2)
```

Also `MoveTo()` supports `bOverwrite`. The behavior is the same as `CopyTo()`.

A file can be deleted.

Delete a file:

```
bSuccess, sResult = oFile_1.Delete()
```

It is possible to distinguish between two different motivations to delete a file:

1. **Explicitely do a deletion**
   This requires that the file to be deleted, does exist.
2. **Making sure only that the files does not exist**
   In this case it doesn't matter that maybe there is nothing to delete because the file already does not exist.

The optional parameter `bConfirmDelete` controls this behavior.

Default is that `Delete()` requires an existing file to delete:

```
bSuccess, sResult = oFile_1.Delete(bConfirmDelete=True)
```

In case of the file does not exist, `Delete()` returns `bSuccess = False` .

`Delete()` also returns `bSuccess = False|None` in case of an existing file cannot be deleted (e.g. because of an access violation).

If it doesn't matter it the file exists or not, it has to be coded explicitely:

```
bSuccess, sResult = oFile_1.Delete(bConfirmDelete=False)
```

In this case `Delete()` only returns `bSuccess = False|None` in case of an existing file cannot be deleted (e.g. because of an access violation).

Avoid access violations

Like already mentioned above every instance of `CFile` has an exclusive access to it's own file.

Only in case of `CopyTo()` and `MoveTo()` other files are involved: the destination files.

To avoid access violations it is not possible to copy or move a file to another file, that is under access of another instance of `CFile` .

In the following example `oFile_1.CopyTo(sFile_2)` returns `bSuccess = False` because `sFile_2` is already in access by `oFile_2` .

```
oFile_1 = CFile(sFile_1)
bSuccess, sResult = oFile_1.Write("A B C")

oFile_2 = CFile(sFile_2)
listLines_2, bSuccess, sResult = oFile_2.ReadLines()

bSuccess, sResult = oFile_1.CopyTo(sFile_2)

del oFile_1
del oFile_2
```

The solution is to delete the class instances as early as possible.

In the following example the copying is successful:

```
oFile_1 = CFile(sFile_1)
bSuccess, sResult = oFile_1.Write("A B C")

oFile_2 = CFile(sFile_2)
listLines_2, bSuccess, sResult = oFile_2.ReadLines()
del oFile_2

bSuccess, sResult = oFile_1.CopyTo(sFile_2)
del oFile_1
```

2.2.3 Utilities

PrettyPrint

The idea behind the `PrettyPrint()` function is to resolve the content of composite data types and provide for every parameter inside:

- the type
- the total number of elements inside (e.g. the number of keys inside a dictionary)
- the counter number of the current element
- the value

Example

The following Python code defines a composite data type and prints the content with `PrettyPrint()` :

```
dictTest = {'K1' : 'value',
            'K2' : ["A", 22, True, (33, 'XYZ')],
            'K3' : 10,
            'K4' : {'A' : 1,
                    'B' : 2}}

PrettyPrint(dictTest)
```

Result

```
[DICT] (4/1) > {K1} [STR]  :  'value'
[DICT] (4/2) > {K2} [LIST] (4/1) > [STR]  :  'A'
[DICT] (4/2) > {K2} [LIST] (4/2) > [INT]  :  22
[DICT] (4/2) > {K2} [LIST] (4/3) > [BOOL]  :  True
[DICT] (4/2) > {K2} [LIST] (4/4) > [TUPLE] (2/1) > [INT]  :  33
[DICT] (4/2) > {K2} [LIST] (4/4) > [TUPLE] (2/2) > [STR]  :  'XYZ'
[DICT] (4/3) > {K3} [INT]  :  10
[DICT] (4/4) > {K4} [DICT] (2/1) > {A} [INT]  :  1
[DICT] (4/4) > {K4} [DICT] (2/2) > {B} [INT]  :  2
```

Every line of output has to be interpreted strictly from left to right.

For example the meaning of the fifth line of output

```
[DICT] (4/2) > {K2} [LIST] (4/4) > [TUPLE] (2/1) > [INT]  :  33
```

is:

- The type of input parameter `oData` is `dict`
- The dictionary contains 4 keys
- The current line gives information about the second key of the dictionary
- The name of the second key is `K2`
- The value of the second key is of type `list`
- The list contains 4 elements
- The current line gives information about the fourth element of the list
- The fourth element of the list is of type `tuple`
- The tuple contains 2 elements
- The current line gives information about the first element of the tuple
- The first element of the tuple is of type `int` and has the value `33`

Types are encapsulated in square brackets, counter in round brackets and key names are encapsulated in curly brackets.

Chapter 3

CComparison.py

3.1 Class: CComparison

Imported by:

```
from PythonExtensionsCollection.Comparison.CComparison import CComparison
```

The class `CComparison` contains mechanisms to compare two files either based on the original version of these files or based on an extract (made with regular expressions) to ensure that only relevant parts of the files are compared.

3.1.1 Method: Compare

Compares two files. While reading in all files empty lines are skipped.

Arguments:

- `sFile_1`
  / *Condition*: required / *Type*: str /
  First file used for comparison.
- `sFile_2`
  / *Condition*: required / *Type*: str /
  Second file used for comparison.
- `sPatternFile`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Pattern file containing a set of regular expressions (line by line). The regular expressions are used to make an extract of both input files. In this case the extracts are compared (instead of the original file content).

Returns:

- `bIdentical`
  / *Type*: bool /
  Indicates if the two input files have the same content or not.
- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method.

Chapter 4

CFile.py

4.1 Class: enFileStatiType

Imported by:

```
from PythonExtensionsCollection.File.CFile import enFileStatiType
```

The class `enFileStatiType` defines the sollowing file states:

- `closed`
- `openedforwriting`
- `openedforappending`
- `openedforreading`

4.2 Class: CFile

Imported by:

```
from PythonExtensionsCollection.File.CFile import CFile
```

The class `CFile` provides a small set of file functions with extended parametrization (like switches defining if a file is allowed to be overwritten or not).

Most of the functions at least returns `bSuccess` and `sResult`.

- `bSuccess` is `True` in case of no error occurred.
- `bSuccess` is `False` in case of an error occurred.
- `bSuccess` is `None` in case of a very fatal error occurred (exceptions).
- `sResult` contains details about what happens during computation.

Every instance of CFile handles one single file only and forces exclusive access to this file.

It is not possible to create an instance of this class with a file that is already in use by another instance.

It is also not possible to use `CopyTo` or `MoveTo` to overwrite files that are already in use by another instance. This makes the file handling more save against access violations.

4.2.1 Method: Close

Closes the opened file.

Arguments:

(no args)

Returns:

- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method.

4.2.2 Method: Delete

Deletes the current file.

Arguments:

- `bConfirmDelete`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  Defines if it will be handled as error if the file does not exist.
  If `True`: If the file does not exist, the method indicates an error (`bSuccess = False`).
  If `False`: It doesn't matter if the file exists or not.

Returns:

- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method.

4.2.3 Method: Write

Writes the content of a variable `Content` to file.

Arguments:

- `Content`
  / *Condition*: required / *Type*: one of: str, list, tuple, set, dict, dotdict /
  If `Content` is not a string, the `Write` method resolves the data structure before writing the content to file.
- `nVSpaceAfter`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Adds vertical space `nVSpaceAfter` (= number of blank lines) after `Content`.
- `sPrefix`
  / *Condition*: optional / *Type*: str / *Default*: None /
  sPrefix is added to every line of output (in case of sPrefix is not None').
- `bToScreen`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Prints `Content` also to screen (in case of `bToScreen` is `True`).

Returns:

- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method.

4.2.4 Method: Append

Appends the content of a variable `Content` to file.

Arguments:

- `Content`
  / *Condition*: required / *Type*: one of: str, list, tuple, set, dict, dotdict /
  If `Content` is not a string, the `Write` method resolves the data structure before writing the content to file.
- `nVSpaceAfter`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Adds vertical space `nVSpaceAfter` (= number of blank lines) after `Content`.
- `sPrefix`
  / *Condition*: optional / *Type*: str / *Default*: None /
  sPrefix is added to every line of output (in case of sPrefix is not None‘).
- `bToScreen`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Prints `Content` also to screen (in case of `bToScreen` is `True`).

Returns:

- `bSuccess`
  / *Type*: bool /
  Indicates if the computation of the method was successful or not.
- `sResult`
  / *Type*: str /
  The result of the computation of the method.

4.2.5 Method: ReadLines

Reads content from current file. Returns an array of lines together with `bSuccess` and `sResult` (feedback).

The method takes care of opening and closing the file. The complete file content is read by `ReadLines` in one step, but with the help of further parameters it is possible to reduce the content by including and excluding lines.

Internally `ReadLines` uses the string filter method `StringFilter`. All filter related input parameter of `ReadLines` and `StringFilter` are the same.

The logical join of all filter is: `AND`.

Arguments:

- `bCaseSensitive`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  - If `True`, the standard filters work case sensitive, otherwise not.
  - This has no effect to the regular expression based filters `sInclRegEx` and `sExclRegEx`.
- `bSkipBlankLines`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True`, blank lines will be skipped, otherwise not.
- `sComment`
  / *Condition*: optional / *Type*: str / *Default*: None /
  In case of a line starts with the string `sComment`, this line is skipped.

- `sStartsWith`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - The criterion of this filter is fulfilled in case of the input string starts with the string `sStartsWith`
  - More than one string can be provided (semicolon separated; logical join: `OR`)
- `sEndsWith`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - The criterion of this filter is fulfilled in case of the input string ends with the string `sEndsWith`
  - More than one string can be provided (semicolon separated; logical join: `OR`)
- `sStartsNotWith`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - The criterion of this filter is fulfilled in case of the input string starts not with the string `sStartsNotWith`
  - More than one string can be provided (semicolon separated; logical join: `AND`)
- `sEndsNotWith`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - The criterion of this filter is fulfilled in case of the input string ends not with the string `sEndsNotWith`
  - More than one string can be provided (semicolon separated; logical join: `AND`)
- `sContains`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - The criterion of this filter is fulfilled in case of the input string contains the string `sContains` at any position
  - More than one string can be provided (semicolon separated; logical join: `OR`)
- `sContainsNot`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - The criterion of this filter is fulfilled in case of the input string does **not** contain the string `sContainsNot` at any position
  - More than one string can be provided (semicolon separated; logical join: `AND`)
- `sInclRegEx`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - *Include* filter based on regular expressions (consider the syntax of regular expressions!)
  - The criterion of this filter is fulfilled in case of the regular expression `sInclRegEx` matches the input string
  - Leading and trailing blanks within the input string are considered
  - `bCaseSensitive` has no effect
  - A semicolon separated list of several regular expressions is **not** supported
- `sExclRegEx`
  / *Condition*: optional / *Type*: str / *Default*: None /
  - *Exclude* filter based on regular expressions (consider the syntax of regular expressions!)
  - The criterion of this filter is fulfilled in case of the regular expression `sExclRegEx` does **not** match the input string
  - Leading and trailing blanks within the input string are considered
  - `bCaseSensitive` has no effect
  - A semicolon separated list of several regular expressions is **not** supported

- `bLStrip`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True`, leading spaces are removed from line before the filters are used, otherwise not.
- `bRStrip`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If `True`, trailing spaces are removed from line before the filters are used, otherwise not.
- `bToScreen`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True`, the content read from file is also printed to screen, otherwise not.

4.2.6 Method: GetFileInfo

Returns the following informations about the file (encapsulated within a dictionary `dFileInfo`):

Returns:

- Key `sFile`
  / *Type*: str /
  Path and name of current file
- Key `bFileIsExisting`
  / *Type*: bool /
  `True` if file is existing, otherwise `False`
- Key `sFileName`
  / *Type*: str /
  The name of the current file (incl. extension)
- Key `sFileExtension`
  / *Type*: str /
  The extension of the current file
- Key `sFileNameOnly`
  / *Type*: str /
  The pure name of the current file (without extension)
- Key `sFilePath`
  / *Type*: str /
  The the path to current file
- Key `bFilePathIsExisting`
  / *Type*: bool /
  `True` if file path is existing, otherwise `False`

4.2.7 Method: CopyTo

Copies the current file to `sDestination`, that can either be a path without file name or a path together with a file name.

In case of the destination file already exists and `bOverwrite` is `True`, than the destination file will be overwritten.

In case of the destination file already exists and `bOverwrite` is `False` (default), than the destination file will not be overwritten and `CopyTo` returns `bSuccess = False`.

Arguments:

- `sDestination`

  / *Condition*: required / *Type*: string /

  The path to destination file (either incl. file name or without file name)

- `bOverwrite`

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

  - In case of the destination file already exists and `bOverwrite` is `True`, than the destination file will be overwritten.
  - In case of the destination file already exists and `bOverwrite` is `False` (default), than the destination file will not be overwritten and `CopyTo` returns `bSuccess = False`.

Returns:

- `bSuccess`

  / *Type*: bool /

  Indicates if the computation of the method was successful or not.

- `sResult`

  / *Type*: str /

  The result of the computation of the method.

4.2.8 Method: MoveTo

Moves the current file to `sDestination`, that can either be a path without file name or a path together with a file name.

Arguments:

- `sDestination`

  / *Condition*: required / *Type*: string /

  The path to destination file (either incl. file name or without file name)

- `bOverwrite`

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

  - In case of the destination file already exists and `bOverwrite` is `True`, than the destination file will be overwritten.
  - In case of the destination file already exists and `bOverwrite` is `False` (default), than the destination file will not be overwritten and `MoveTo` returns `bSuccess = False`.

Returns:

- `bSuccess`

  / *Type*: bool /

  Indicates if the computation was successful or not

- `sResult`

  / *Type*: str /

  Contains details about what happens during computation

Chapter 5

CFolder.py

5.1 Function: rm_dir_readonly

Calls `os.chmod` in case of `shutil.rmtree` (within `Delete()`) throws an exception (making files writable).

5.2 Class: CFolder

Imported by:

```
from PythonExtensionsCollection.Folder.CFolder import CFolder
```

The class `CFolder` provides a small set of folder functions with extended parametrization (like switches defining if a folder is allowed to be overwritten or not).

Most of the functions at least returns `bSuccess` and `sResult`.

- `bSuccess` is `True` in case of no error occurred.
- `bSuccess` is `False` in case of an error occurred.
- `bSuccess` is `None` in case of a very fatal error occurred (exceptions).
- `sResult` contains details about what happens during computation.

Every instance of CFolder handles one single folder only and forces exclusive access to this folder.

It is not possible to create an instance of this class with a folder that is already in use by another instance.

The constructor of `CFolder` requires the input parameter `sFolder`, that is the path and the name of a folder that is handled by the current class instance.

5.2.1 Method: Delete

Deletes the folder the current class instance contains.

Arguments:

- `bConfirmDelete`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  Defines if it will be handled as error if the folder does not exist.
  If `True`: If the folder does not exist, the method indicates an error (`bSuccess = False`).
  If `False`: It doesn't matter if the folder exists or not.

Returns:

- `bSuccess`

  / *Type*: bool /

  Indicates if the computation of the method was successful or not.

- `sResult`

  / *Type*: str /

  The result of the computation of the method.

5.2.2 Method: Create

Creates the current folder `sFolder`.

Arguments:

- `bOverwrite`

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

  - In case of the folder already exists and `bOverwrite` is `True`, than the folder will be deleted before creation.
  - In case of the folder already exists and `bOverwrite` is `False` (default), than the folder will not be touched.

  In both cases the return value `bSuccess` is `True` - because the folder exists.

- `bRecursive`

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

  - In case of `bRecursive` is `True`, than the complete destination path will be created (including all intermediate subfolders).
  - In case of `bRecursive` is `False`, than it is expected that the parent folder of the new folder already exists.

Returns:

- `bSuccess`

  / *Type*: bool /

  Indicates if the computation of the method was successful or not.

- `sResult`

  / *Type*: str /

  The result of the computation of the method.

5.2.3 Method: CopyTo

Copies the current folder to `sDestination`, that has to be a path to a folder **within** the source folder will be copied to (with it's original name),

In case of the destination folder already exists and `bOverwrite` is `True`, than the destination folder will be overwritten.

In case of the destination folder already exists and `bOverwrite` is `False` (default), than the destination folder will not be overwritten and `CopyTo` returns `bSuccess = False`.

Arguments:

- `sDestination`

  / *Condition*: required / *Type*: string /

  The path to destination folder

- `bOverwrite`

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

  - In case of the destination folder already exists and `bOverwrite` is `True`, than the destination folder will be overwritten.
  - In case of the destination folder already exists and `bOverwrite` is `False` (default), than the destination folder will not be overwritten and `CopyTo` returns `bSuccess = False`.

Returns:

- `bSuccess`

  / *Type*: bool /

  Indicates if the computation of the method was successful or not.

- `sResult`

  / *Type*: str /

  The result of the computation of the method.

Chapter 6

CString.py

6.1 Class: CString

Imported by:

```
from PythonExtensionsCollection.String.CString import CString
```

The class `CString` contains some string computation methods like e.g. normalizing a path.

6.1.1 Method: NormalizePath

Normalizes local paths, paths to local network resources and internet addresses

Arguments:

- `sPath`
  / *Condition*: required / *Type*: str /
  The path to be normalized
- `bWin`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True` then returned path contains masked backslashes as separator, otherwise slashes
- `sReferencePathAbs`
  / *Condition*: optional / *Type*: str / *Default*: None /
  In case of `sPath` is relative and `sReferencePathAbs` (expected to be absolute) is given, then the returned absolute path is a join of both input paths
- `bConsiderBlanks`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True` then the returned path is encapsulated in quotes - in case of the path contains blanks
- `bExpandEnvVars`
  / *Condition*: optional / *Type*: bool / *Default*: True /

  If `True` then in the returned path environment variables are resolved, otherwise not.
- `bMask`
  / *Condition*: optional / *Type*: bool / *Default*: True (requires `bWin=True`)/
  - If `bWin` is `True` and `bMask` is `True` then the returned path contains masked backslashes as separator.
  - If `bWin` is `True` and `bMask` is `False` then the returned path contains single backslashes only - this might be required for applications, that are not able to handle masked backslashes.
  - In case of `bWin` is `False` `bMask` has no effect.

Returns:

- `sPath`

  / *Type*: str /

  The normalized path (is `None` in case of `sPath` is `None`)

6.1.2 Method: DetectParentPath

Computes the path to any parent folder inside a given path. Optionally DetectParentPath is able to search for files inside the identified parent folder.

Arguments:

- `sStartPath`

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

  The path in which to search for a parent folder

- `sFolderName`

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

  The name of the folder to search for within `sStartPath`. It is possible to provide more than one folder name separated by semicolon

- `sFileName`

  / *Condition*: optional / *Type*: str / *Default*: None /

  The name of a file to search within the detected parent folder

Returns:

- `sDestPath`

  / *Type*: str /

  Path and name of parent folder found inside `sStartPath`, `None` in case of `sFolderName` is not found inside `sStartPath`. In case of more than one parent folder is found `sDestPath` contains the first result and `listDestPaths` contains all results.

- `listDestPaths`

  / *Type*: list /

  If `sFolderName` contains a single folder name this list contains only one element that is `sDestPath`. In case of `FolderName` contains a semicolon separated list of several folder names this list contains all found paths of the given folder names. `listDestPaths` is `None` (and not an empty list!) in case of `sFolderName` is not found inside `sStartPath`.

- `sDestFile`

  / *Type*: str /

  Path and name of `sFileName`, in case of `sFileName` is given and found inside `listDestPaths`. In case of more than one file is found `sDestFile` contains the first result and `listDestFiles` contains all results. `sDestFile` is `None` in case of `sFileName` is `None` and also in case of `sFileName` is not found inside `listDestPaths` (and therefore also in case of `sFolderName` is not found inside `sStartPath`).

- `listDestFiles`

  / *Type*: list /

  Contains all positions of `sFileName` found inside `listDestPaths`.

  `listDestFiles` is `None` (and not an empty list!) in case of `sFileName` is `None` and also in case of `sFileName` is not found inside `listDestPaths` (and therefore also in case of `sFolderName` is not found inside `sStartPath`).

- `sDestPathParent`

  / *Type*: str /

  The parent folder of `sDestPath`, `None` in case of `sFolderName` is not found inside `sStartPath` (`sDestPath` is `None`).

6.1.3 Method: StringFilter

This method provides a bunch of predefined filters that can be used singly or combined to come to a final conclusion if the string fulfils all criteria or not.

These filters can be e.g. used to select or exclude lines while reading from a text file. Or they can be used to select or exclude files or folders while walking through the file system.

The following filters are available:

bSkipBlankStrings

- Leading and trailing spaces are removed from the input string at the beginning
- In case of the result is an empty string and `bSkipBlankStrings` is `True`, the method immediately returns `False` and all other filters are ignored

sComment

- In case of the input string starts with the string `sComment`, the method immediately returns `False` and all other filters are ignored
- Leading blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- The idea behind this decision is: Ignore a string that is commented out

sStartsWith

- The criterion of this filter is fulfilled in case of the input string starts with the string `sStartsWith`
- Leading blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- More than one string can be provided (semicolon separated; logical join: `OR`)

sEndsWith

- The criterion of this filter is fulfilled in case of the input string ends with the string `sEndsWith`
- Trailing blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- More than one string can be provided (semicolon separated; logical join: `OR`)

sStartsNotWith

- The criterion of this filter is fulfilled in case of the input string does **not** start with the string `sStartsNotWith`
- Leading blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- More than one string can be provided (semicolon separated; logical join: `AND`)

sEndsNotWith

- The criterion of this filter is fulfilled in case of the input string does **not** end with the string `sEndsNotWith`
- Trailing blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- More than one string can be provided (semicolon separated; logical join: `AND`)

sContains

- The criterion of this filter is fulfilled in case of the input string contains the string `sContains` at any position

- Leading and trailing blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- More than one string can be provided (semicolon separated; logical join: `OR`)

sContainsNot

- The criterion of this filter is fulfilled in case of the input string does **not** contain the string `sContainsNot` at any position
- Leading and trailing blanks within the input string have no effect
- The decision also depends on `bCaseSensitive`
- More than one string can be provided (semicolon separated; logical join: `AND`)

sInclRegEx

- *Include* filter based on regular expressions (consider the syntax of regular expressions!)
- The criterion of this filter is fulfilled in case of the regular expression `sInclRegEx` matches the input string
- Leading and trailing blanks within the input string are considered
- `bCaseSensitive` has no effect
- A semicolon separated list of several regular expressions is **not** supported

sExclRegEx

- *Exclude* filter based on regular expressions (consider the syntax of regular expressions!)
- The criterion of this filter is fulfilled in case of the regular expression `sExclRegEx` does **not** match the input string
- Leading and trailing blanks within the input string are considered
- `bCaseSensitive` has no effect
- A semicolon separated list of several regular expressions is **not** supported

Further arguments:

- `sString`
  / *Condition*: required / *Type*: str /
  The input string that has to be investigated.
- `bCaseSensitive`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If `True`, the standard filters work case sensitive, otherwise not.
- `bDebug`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True`, additional output is printed to console (e.g. the decision of every single filter), otherwise not.

Returns:

- `bAck`
  / *Type*: bool /
  Final statement about the input string `sString` after filter computation

Further details together with codde examples can be found within chapter **Description**, subsubsection **StringFilter**.

6.1.4 Method: FormatResult

Formats the result string `sResult` depending on `bSuccess`:

- `bSuccess` is `True` indicates *success*
- `bSuccess` is `False` indicates an *error*
- `bSuccess` is `None` indicates an *exception*

Additionally the name of the method that causes the result, can be provided (*optional*). This is useful for debugging.

Arguments:

- `sMethod`
  / *Condition*: optional / *Type*: str / *Default*: (empty string) /
  Name of the method that causes the result.
- `bSuccess`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  Indicates if the computation of the method `sMethod` was successful or not.
- `sResult`
  / *Condition*: optional / *Type*: str / *Default*: (empty string) /
  The result of the computation of the method `sMethod`.

Returns:

- `sResult`
  / *Type*: str /
  The formatted result string.

Chapter 7

CUtils.py

7.1 Function: PrettyPrint

Wrapper function to create and use a `CTypePrint` object. This wrapper function is responsible for printing out the content to console and to a file (depending on input parameter).

The content itself is prepared by the method `TypePrint` of class `CTypePrint`. This happens `PrettyPrint` internally.

The idea behind the `PrettyPrint` function is to resolve also the content of composite data types and provide for every parameter inside:

- the type
- the total number of elements inside (e.g. the number of keys inside a dictionary)
- the counter number of the current element
- the value

Arguments:

- `oData`
  / *Condition*: required / *Type*: (*any Python data type*) /
  A variable of any Python data type.
- `hOutputFile`
  / *Condition*: optional / *Type*: file handle / *Default*: None /
  If handle is not `None` the content is written to this file, otherwise not.
- `bToConsole`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If `True` the content is written to console, otherwise not.
- `nIndent`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Sets the number of additional blanks at the beginning of every line of output (indentation).
- `sPrefix`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Sets a prefix string that is added at the beginning of every line of output.
- `bHexFormat`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True` the output is printed in hexadecimal format (but valid for strings only).

Returns:

- `listOutLines` (*list*)
  / *Type*: list /
  List of lines containing the prepared output

7.2 Class: CTypePrint

Imported by:

```
from PythonExtensionsCollection.Utils.CUtils import CTypePrint
```

The class `CTypePrint` provides a method (`TypePrint`) to compute the following data:

- the type
- the total number of elements inside (e.g. the number of keys inside a dictionary)
- the counter number of the current element
- the value

of simple and composite data types.
The call of this method is encapsulated within the function `PrettyPrint` inside this module.

7.2.1 Method: TypePrint

The method `TypePrint` computes details about the input variable `oData`.
Arguments:

- `oData`
  / *Condition*: required / *Type*: any Python data type /
  Python variable of any data type.
- `bHexFormat`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If `True` the output is provide in hexadecimal format.

Returns:

- `listOutLines`
  / *Type*: list /
  List of lines containing the resolved content of `oData`.

Chapter 8

Appendix

About this package:

Table 8.1: Package setup

Setup parameter	Value
Name	PythonExtensionsCollection
Version	0.11.3
Date	21.11.2022
Description	Additional Python functions
Package URL	python-extensions-collection
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.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 9

History

0.1.0	08/2021
Initial version	
0.2.0	02/2022
Code maintenance	
0.3.0	20.05.2022
Documentation tool chain switched to `GenPackageDoc`	
0.4.0	24.05.2022
- Documentation rebuild with `GenPackageDoc` *v. 0.13.0* *- Code maintenance*	
0.5.0	31.05.2022
Adapted to `GenPackageDoc` *v. 0.15.0*	
0.6.0	02.06.2022
- Documentation rebuild with `GenPackageDoc` *v. 0.16.0* *- Code maintenance*	
0.7.0	10.06.2022
Module `CFolder` *added (with methods:* `Create` *and* `Delete`*)*	
0.8.0	28.06.2022
Module `CFolder`*: Method:* `CopyTo` *added*	
0.9.0	27.07.2022
History reworked (requires `GenPackageDoc` *v. 0.26.0 at least)*	
0.10.0	25.10.2022
Added `CComparison` *module to compare two files (either based on original content or based on pattern)*	
0.11.0	15.11.2022
Converted parts of the documentation from RST format to LaTeX format (to improve the layout).	

PythonExtensionsCollection.pdf
Created at 13.12.2022 - 16:28:41
by GenPackageDoc v. 0.38.0

4.3 RobotframeworkExtensions

RobotframeworkExtensions

v. 0.8.5

Holger Queckenstedt

21.11.2022

Contents

Chapter 1

Introduction

The **RobotframeworkExtensions** extend the functionality of the Robot Framework by some useful keywords.

This covers for example string operations like normalizing a path and a pretty print method (especially for composite Python data types).

The sources of the **RobotframeworkExtensions** are available in GitHub.

Informations about how to install the **RobotframeworkExtensions** can be found in the README.

The **RobotframeworkExtensions** keywords are implemented in Python (as **PythonExtensionsCollection**) and the implementation can also be found in GitHub.

Informations about how to install the **PythonExtensionsCollection** can be found in the README.

Chapter 2

Description

2.1 Keywords

The `Collection` module of the **RobotframeworkExtensions** is the interface between the **PythonExtensionsCollection** and the RobotFramework AIO and contains the keyword definitions that can be imported in the following way:

```
Library    RobotframeworkExtensions.Collection    WITH NAME    rf.extensions
```

We recommend to use `WITH NAME` to shorten the long library name a little bit. That will make the robot code easier to read.

2.1.1 pretty_print

The `pretty_print` keyword logs the content of parameters of any Python data type. Simple data types are logged directly. Composite data types are resolved before.

The output contains for every parameter:

- the type
- the total number of elements inside (e.g. the number of keys inside a dictionary)
- the counter number of the current element
- the value

The trace level for output is `INFO` . The output is also returned as list of strings.

Example

The following RobotFramework AIO code defines - step by step - a parameter of composite data type (nested arrays and dictionaries) and prints the content of this with `pretty_print` :

```
set_test_variable    @{aItems1}    ${33}
...                                XYZ

set_test_variable    @{aItems}     A
...                                ${22}
...                                ${True}
...                                ${aItems1}

set_test_variable    &{dItems1}    A=${1}
...                                B=${2}

set_test_variable    &{dItems}     K1=value
...                                K2=${aItems}
...                                K3=${10}
...                                K4=${dItems1}

rf.extensions.pretty_print    ${dItems}
```

Result

```
[DOTDICT] (4/1) > {K1} [STR]  :  'value'
[DOTDICT] (4/2) > {K2} [LIST] (4/1) > [STR]  :  'A'
[DOTDICT] (4/2) > {K2} [LIST] (4/2) > [INT]  :  22
[DOTDICT] (4/2) > {K2} [LIST] (4/3) > [BOOL]  :  True
[DOTDICT] (4/2) > {K2} [LIST] (4/4) > [LIST] (2/1) > [INT]  :  33
[DOTDICT] (4/2) > {K2} [LIST] (4/4) > [LIST] (2/2) > [STR]  :  'XYZ'
[DOTDICT] (4/3) > {K3} [INT]  :  10
[DOTDICT] (4/4) > {K4} [DOTDICT] (2/1) > {A} [INT]  :  1
[DOTDICT] (4/4) > {K4} [DOTDICT] (2/2) > {B} [INT]  :  2
```

Every line of output has to be interpreted strictly from left to right.

For example the meaning of the fifth line of output

```
[DOTDICT] (4/2) > {K2} [LIST] (4/4) > [LIST] (2/1) > [INT]  :  33
```

is:

- The type of input parameter `dItems` is `dotdict`
- The dictionary contains 4 keys
- The current line gives information about the second key of the dictionary
- The name of the second key is `K2`
- The value of the second key is of type `list`
- The list contains 4 elements
- The current line gives information about the fourth element of the list
- The fourth element of the list is of type `list`
- The list contains 2 elements
- The current line gives information about the first element of the list
- The first element of the list is of type `int` and has the value `33`

Types are encapsulated in square brackets, counter in round brackets and key names are encapsulated in curly brackets.

2.1.2 normalize_path

The `normalize_path` keyword normalizes local paths, paths to local network resources and internet addresses.

Background

It's not easy to handle paths - and especially the path separators - independent from the operating system.

Under Linux it is obvious that single slashes are used as separator within paths. Whereas the Windows explorer uses single backslashes. In both operating systems web addresses contains single slashes as separator when displayed in web browsers.

Using single backslashes within code - as content of string variables - is dangerous because the combination of a backslash and a letter can be interpreted as escape sequence - and this is maybe not the effect a user wants to have.

To avoid unwanted escape sequences backslashes have to be masked (by the usage of two of them: `"\\"`). But also this could not be the best solution because there are also applications (like the Windows explorer) that are not able to handle masked backslashes. They expect to get single backslashes within a path.

Preparing a path for best usage within code also includes collapsing redundant separators and up-level references. Python already provides functions to do this, but the outcome (path contains slashes or backslashes) depends on the operating system. And like already mentioned above also under Windows backslashes might not be the preferred choice.

It also has to be considered that redundant separators at the beginning of an address of a local network resource (like `\\server.com`) and or inside an internet address (like `https:\\server.com`) must **not** be collapsed! Unfortunately the Python function `normpath` does not consider this context.

To give the user full control about the format of a path, independent from the operating system and independent if it's a local path, a path to a local network resource or an internet address, the keyword `normalize_path` provides lot's of parameters to influence the result.

Example 1

Variable containing a path with:

- different types of path separators
- redundant path separators (*but backslashes have to be masked in the definition of the variable, this is* ***not*** *an unwanted redundancy*)
- up-level references

```
set_test_variable    ${sPath}    C:\\\\subfolder1///../subfolder2\\\\\\\\../subfolder3\\\\
```

Printing the content of `sPath` shows how the path looks like when the masking of the backslashes is resolved:

```
C:\\subfolder1///../subfolder2\\\\../subfolder3\\
```

Usage of the `normalize_path` keyword:

```
${sPath}    rf.extensions.normalize_path    ${sPath}
```

Result (content of `sPath`):

```
C:/subfolder3
```

In case we need the Windows version (with masked backslashes instead of slashes):

```
${sPath}    rf.extensions.normalize_path    ${sPath}    bWin=${True}
```

Result (content of `sPath`):

```
C:\\subfolder3
```

The masking of backslashes can be deactivated:

```
${sPath}    rf.extensions.normalize_path    ${sPath}    bWin=${True}    bMask=${False}
```

Result (content of `sPath`):

```
C:\subfolder3
```

Example 2

Variable containing a path of a local network resource (path starts with two masked backslashes):

```
set_test_variable    ${sPath}    \\\\anyserver.com\\\\part1//part2\\\\part3/part4
```

Result of normalization:

```
//anyserver.com/part1/part2/part3/part4
```

Example 3

Variable containing an internet address:

```
set_test_variable    ${sPath}    http:\\\\anyserver.com\\\\part1//part2\\\\part3/part4
```

Result of normalization:

```
http://anyserver.com/part1/part2/part3/part4
```

Chapter 3

Collection.py

The Collection module is the interface between the PythonExtensionsCollection and the Robot Framework.
This library containing the keyword definitions, can be imported in the following way:

```
Library    RobotframeworkExtensions.Collection    WITH NAME    rf.extensions
```

3.1 Class: Collection

Imported by:

```
from RobotframeworkExtensions.Collection import Collection
```

Module main class

3.1.1 Keyword: pretty_print

The `pretty_print` keyword logs the content of parameters of any Python data type (input: `oData`).
Simple data types are logged directly. Composite data types are resolved before.
The output contains for every parameter:

- the type
- the total number of elements inside (e.g. the number of keys inside a dictionary)
- the counter number of the current element
- the value

The trace level for output is `INFO`.
The output is also returned as list of strings.
Arguments:

- `oData`
  / *Condition*: required / *Type*: any Python type /
  Data to be pretty printed

Returns:

- `listOutLines` (*list*)
  / *Type*: list /
  List of strings containing the resolved data structure of `oData` (same content as printed to console).

3.1.2 Keyword: normalize_path

The `normalize_path` keyword normalizes local paths, paths to local network resources and internet addresses

Arguments:

- `sPath`

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

  The path to be normalized

- `bWin`

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

  If `True` then the returned path contains masked backslashes as separator, otherwise slashes

- `sReferencePathAbs`

  / *Condition*: optional / *Type*: str / *Default*: None /

  In case of `sPath` is relative and `sReferencePathAbs` (expected to be absolute) is given, then the returned absolute path is a join of both input paths

- `bConsiderBlanks`

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

  If `True` then the returned path is encapsulated in quotes - in case of the path contains blanks

- `bExpandEnvVars`

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

  If `True` then in the returned path environment variables are resolved, otherwise not.

- `bMask`

  / *Condition*: optional / *Type*: bool / *Default*: True (requires `bWin=True`) /

  If `bWin` is `True` and `bMask` is `True` then the returned path contains masked backslashes as separator.

  If `bWin` is `True` and `bMask` is `False` then the returned path contains single backslashes only - this might be required for applications, that are not able to handle masked backslashes.

  In case of `bWin` is `False` `bMask` has no effect.

Returns:

- `sPath`

  / *Type*: str /

  The normalized path (is `None` in case of `sPath` is `None`)

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	RobotframeworkExtensions
Version	0.8.5
Date	21.11.2022
Description	Additional Robot Framework keywords
Package URL	robotframework-extensions-collection
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.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	01/2022
Initial version	
0.2.0	03/2022
Setup maintenance	
0.3.0	05/2022
Documentation tool chain switched to `GenPackageDoc`	
0.4.0	24.05.2022
- Documentation rebuild with `GenPackageDoc` *v. 0.13.0* *- Code maintenance*	
0.5.0	31.05.2022
Adapted to `GenPackageDoc` *v. 0.15.0*	
0.6.0	02.06.2022
- Documentation rebuild with `GenPackageDoc` *v. 0.16.0* *- Code maintenance*	
0.7.0	28.06.2022
`PythonExtensionsCollection` *updated to version 0.8.0*	
0.8.0	27.07.2022
History reworked (requires `GenPackageDoc` *v. 0.26.0 at least)*	

RobotframeworkExtensions.pdf
Created at 13.12.2022 - 16:28:50
by GenPackageDoc v. 0.38.0

4.4 JsonPreprocessor

JsonPreprocessor

v. 0.1.4

Mai Dinh Nam Son

14.09.2022

Contents

Chapter 1

Introduction

1.1 Json Preprocessor documentation

This is the documentation for Python JsonPreprocessor

Json is a format used to represent data and becomes the universal standard of data exchange. Today many software projects are using configuration file in Json format. For a big or a complex project there is a need to have some enhanced format in Json file such as adding the comments, importing other Json files, etc.

Based on that needs, we develop JsonPreprocessor package:

- Gives the possibility to comment out parts of the content. This feature can be used to explain the meaning of the parameters defined inside the configuration files.
- Has ability to import other Json files. This feature can be applied for complex project, users can create separated Json files then importing them to other Json file.
- Allows users using the defined parameter in Json file.
- Accepts **"True"**, **"False"**, and **"None"** in Json syntax

Chapter 2

Description

2.1 How to install

Firstly, clone python-jsonpreprocessor repository to your machine.

Then open the folder in which you have cloned the repository python-jsonpreprocessor, following the commands below to build or install this package:

```
setup.py build      will build the package underneath 'build/'
setup.py install    will install the package
```

After the build processes is completed, the package is located in 'build/', and the generated package documentation is located in **/JsonPreprocessor**.

We can use `--help` to discover the options for `build` command, example:

```
setup.py build      will build the package underneath 'build/'
setup.py install    will install the package
```

2.2 Features

2.2.1 Basic Json format

Users can use JsonPreprocessor to handle the json file with its original format.

Example:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "param_1" : "value_1",
      "param_2" : value_2,
      "structure_param": {
        "general": "general"
      }
    }
  },
  "device" : "device_name"
}
```

2.2.2 Adding the comments

Often large projects require a lot of configuration parameters. So adding comments to json files is useful in case of more and more content is added, e.g. because of a json file has to hold a huge number of configuration parameters for different features. Comments can be used here to clarify the meaning of these parameters or the differences between them.

Every line starting with "//", is commented out. Therefore a comment is valid for singles lines only.

Comment out a block of several lines with only one start and one end comment string, is currently not supported.

Example:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
{
  "Project": "name_of_prject",
  // <adding comment>
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    // <adding comment>
    "global": {
      "param_1" : "value_1",
      "param_2" : value_2,  // <adding comment>
      "structure_param": {
        "general": "general"
      }
    }
  },
  "device" : "device_name"
}
```

2.2.3 Imports other json files

This import feature enables developers to take over the content of other json files into the current json file. A json file that is imported into another json file, can contain imports also (allows nested imports).

A possible usecase for nested imports is to handle configuration parameters of different variants of a feature or a component within a bunch of several smaller files, instead of putting all parameter into only one large json file.

Example:

Suppose we have the json file `params_global.json` with the content:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
//
//  This is to distinguish the different types of resets
{
  "import_param_1" : "value_1",

  "import_param_2" : "value_2",
```

```
  "import_structure_1": {   // <adding comment>
    "general": "general"
  }
}
```

And other json file `preprocessor_definitions.json` with content:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
{
  "import_param_3" : "value_3",

  "import_param_4" : "value_4",

  // <adding comment>

  "import_structure_2": {
     "general": "general"
   }
}
```

Then we can import these 2 files above to the json file `config.json` with the [import] statement:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
           "[import]": "<path_to_the_imported_file>/params_global.json"
     }
   },
  "preprocessor": {
    "definitions": {
        "[import]": "<path_to_the_imported_file>/preprocessor_definitions.json"
      }
  },
  "device" : "device_name"
}
```

After all imports are resolved by the JsonPreprocessor, this is the resulting of data structure:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
```

```
    "patch": "1"
  },
  "params": {
    "global": {
      "import_param_1" : "value_1",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general"
        }
    }
  },
  "preprocessor": {
    "definitions": {
      "import_param_3" : "value_3",
      "import_param_4" : "value_4",
      "import_structure_2": {
         "general": "general"
       }
    }
  },
  "device" : "device_name"
}

Add new or overwrites existing parameters
```

This JsonPreprocessor package also provides developers ability to add new as well as overwrite existing parameters. Developers can update parameters which are already declared and add new parameters or new element into existing parameters. The below example will show the way to do these features.

In case we have many different variants, and each variant requires a different value assigned to the parameter, users can use this feature to add new parameters and update new values for existing parameters of existing configuation object.

Example:

Suppose we have the json file `params_global.json` with the content:

```
//************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//************************************************************************
//
//  This is to distinguish the different types of resets
{
  "import_param_1" : "value_1",

  "import_param_2" : "value_2",

  "import_structure_1": {   // <adding comment>
    "general": "general"
  }
}
```

Then we import `params_global.json` to json file `config.json` with content:

```
{
  "Project": "name_of_prject",
  "version": {
```

```
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
       "[import]": "<path_to_the_imported_file>/params_global.json"
      }
    },
  "device" : "device_name",
  // Overwrite parameters
  "${params}['global']['import_param_1']": "new_value_1",
  "${version}['patch']": "2",
  // Add new parameters
  "new_param": {
       "abc": 9,
           "xyz": "new param"
  },
  "${params}['global']['import_structure_1']['new_structure_param']":
↪  "new_structure_value"
}
```

After all imports are resolved by the JsonPreprocessor, this is the resulting of data structure:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "2"
  },
  "params": {
    "global": {
      "import_param_1" : "new_value_1",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general",
            "new_structure_param": "new_structure_value"
       }
     }
  },
  "device" : "device_name",
  "new_param": {
      "abc": 9,
      "xyz": "new param"
  }
}
```

2.2.4 Using defined parameters

With JsonPreprocessor package, users can also use the defined parameters in Json file. The value of the defined parameter could be called with syntax `${<parameter_name>}`

Example:

Suppose we have the json file `config.json` with the content:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
```

```
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "import_param_1" : "value_1",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general"
       }
    }
  },
  "preprocessor": {
    "definitions": {
      "import_param_3" : "value_3",
      "import_param_4" : "value_4",
        "ABC": "param_ABC",
      "import_structure_1": {
         "general": "general"
       }
    }
  },
  "device" : "device_name",
  // Using the defined parameters
  "${params}['global'][${preprocessor}['definitions']['ABC']]": True,
  "${params}['global']['import_param_1']":
↪  ${preprocessor}['definitions']['import_param_4']
}
```

After all imports are resolved by the JsonPreprocessor, this is the resulting of data structure:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "import_param_1" : "value_4",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general"
        },
        "param_ABC": True
    }
  },
  "preprocessor": {
    "definitions": {
      "import_param_3" : "value_3",
      "import_param_4" : "value_4",
        "ABC": "param_ABC",
      "import_structure_1": {
         "general": "general"
       }
    }
  },
  "TargetName" : "device_name"
}
```

2.2.5 Accepted `True`, `False`, and `None`

Some keywords are different between Json and Python syntax:

- Json syntax: **“true“**, **“false“**, **“null“**
- Python syntax: **“True“**, **“False“**, **“None“**

To facilitate the usage of configuration files in Json format, both ways of syntax are accepted.

Chapter 3

CJsonPreprocessor.py

3.1 Class: CSyntaxType

Imported by:

```
from JsonPreprocessor.CJsonPreprocessor import CSyntaxType
```

3.2 Class: CPythonJSONDecoder

Imported by:

```
from JsonPreprocessor.CJsonPreprocessor import CPythonJSONDecoder
```

Class: PythonJSONDecoder

Add python data types and syntax to json. `True`, `False` and `None` will be a accepted as json syntax elements.

Args:

json.JSONDecoder (*object*)

Decoder object provided by `json.loads`

3.2.1 Method: custom_scan_once

3.3 Class: CJsonPreprocessor

Imported by:

```
from JsonPreprocessor.CJsonPreprocessor import CJsonPreprocessor
```

Class: CJsonPreprocessor

CJsonPreprocessor extends the syntax of json.

Features are

- Allow c/c++-style comments within json files.
  // single line or part of single line and /* */ multline comments are possible
- Allow to import json files into json files
  `"[import]" : "relative/absolute path"`, imports another json file to exactly this location.

- Allow use of the defined paramaters within json files
  In any place the syntax `${basenode.subnode. ... nodename}` allows to reference an already existing parameter.
  - Example:

```
{
    "basenode" : {
                    subnode : {
                                    "myparam" : 5
                                },

                },

    "myVar" : ${basenode.subnode.myparam}
}
```

- Allow Python data types `True`, `False` and `None`

3.3.1 Method: jsonLoad

Method: jsonLoad

This function is the entry point of JsonPreprocessor.

It loads the json file, preprocesses it and returns the preprocessed result as data structure.

Args:

jFile (*string*)

Relative/absolute path to main json file.
`%envvariable%` and `${envvariable}` can be used, too in order to access environment variables.

Returns:

oJson (*dict*)

Preprocessed json file(s) as dictionary data structure

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	JsonPreprocessor
Version	0.1.4
Date	14.09.2022
Description	Preprocessor for json files
Package URL	python-jsonpreprocessor
Author	Mai Dinh Nam Son
Email	son.maidinhnam@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	01/2022
Initial version	
0.1.4	09/2022
Updated documentation	

4.5 RobotFramework_Testsuites

RobotFramework_Testsuites

v. 0.2.2

Mai Dinh Nam Son

18.07.2022

Contents

Chapter 1

Introduction

1.1 RobotFramework AIO testsuites management documentation

This is the documentation for RobotFramework_Testsuites

The RobotFramework_Testsuites package works together with JsonPreprocessor Python package to provide the enhanced features such as json configuration files, 4 different levels of configuation, global parameters, schema validation, etc.

This RobotFramework_Testsuites package will support testing for many variants of product on the same Robot project by switching between different configuration files via variant name.

Chapter 2

Description

2.1 Getting Started

2.1.1 How to install

Firstly, clone **RobotFramework_Testsuites** repository to your machine.

```
git clone
 ↪  https://github.com/test-fullautomation/robotframework-testsuitesmanagement.git
```

Go to **robotframework-testsuitesmanagement**, using the 2 common commands below to build or install this package:

```
setup.py build      will build the package underneath 'build/'
setup.py install    will install the package
```

After the build processes are completed, the package is located in **build/**, and the documents are located in **build/lib/RobotFramework_Testsuites**.

We can use `--help` to discover the options for `build` command, example:

```
setup.py build      will build the package underneath 'build/'
setup.py install    will install the package

Global options:
--verbose (-v)      run verbosely (default)
--quiet (-q)        run quietly (turns verbosity off)
--dry-run (-n)      don't actually do anything
--help (-h)         show detailed help message
--no-user-cfg       ignore pydistutils.cfg in your home directory
--command-packages  list of packages that provide distutils commands

Information display options (just display information, ignore any commands)
--help-commands     list all available commands
--name              print package name
--version (-V)      print package version
--fullname          print <package name>-<version>
--author            print the author's name
--author-email      print the author's email address
--maintainer        print the maintainer's name
--maintainer-email  print the maintainer's email address
--contact           print the maintainer's name if known, else the author's
--contact-email     print the maintainer's email address if known, else the
                    author's
--url               print the URL for this package
--license           print the license of the package
```

```
--licence           alias for --license
--description       print the package description
--long-description  print the long package description
--platforms         print the list of platforms
--classifiers       print the list of classifiers
--keywords          print the list of keywords
--provides          print the list of packages/modules provided
--requires          print the list of packages/modules required
--obsoletes         print the list of packages/modules made obsolete

usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
 or: setup.py --help [cmd1 cmd2 ...]
 or: setup.py --help-commands
 or: setup.py cmd --help
```

2.2 Features

2.2.1 Using configuration files in Json format

Nowadays, Json is the leading of structuring data for exchange not only for web applications but also for other software applications. Json format is used to represent data, and become the universal standard of data exchange. That is the reason we decided using Json format for configuration files of RobotFramework AIO.

Together with `JsonPreprocessor` package, `RobotFramework_Testsuites` supports configuring RobotFramework AIO automation test project with json files which allow users to add the comments, and to import params from other json files. Adding comments and importing json files are enhanced features which are developed and documented in `JsonPreprocessor` python package.

2.2.2 Define 4 levels of configuration

`RobotFramework_Testsuites` management defines 4 different configuration levels, from level 1 to level 4. Level 1 is highest priority, and level 4 is lowest priority.

The 4 different configuration levels helps users more convenient to configure RobotFramework test project:

- Level 1 supports users execute robot run with specific configuration file.
- Level 2 supports users loading configuration file base on variant name.
- Level 3 supports users creating different separated configuration files for indiviual robot testsuite files.
- Level 4 supports users practicing to learn RobotFramework AIO.

Level 1: Loads configuration file via input parameter of robot command

This is highest priority of loading configuration method, that means, configuration level 2 or 3 will be ignored even it is set.

This level 1 configuration is designed for some purpose:

- In case the use wants to execute the robot run with specific configuration file for the particular purposes.
- User re-produces and verifies an issue or a corner case with new configuration file and doesn't want to modify the current configuration file.

User can address the json configuration file when executing robot testsuite with input parameter `--variable config_file:"<path_to_json_file>"`

```

```

robot --variable config_file:"<path_to_json_file>" <path_to_testsuite>

Level 2: Loads Json configuration according to variant name

This level 2 is designed for the scenario that user creates the automation testing project which running for many different variants. When trigger robot run, it will load the appropriate json configuration file.

To set RobotFramework AIO run with level 2, first user has to create a json file which contains different variants point to different configuration files.

For example, we create the `variants_cfg.json` with content below:

```
{
  "default": {
    "name": "<default_cfg_file>",
    "path": "<path>"
  },
  "variant_0": {
    "name": "<file_name_variant_0>",
    "path": "<path>"
  },
  "variant_1": {
    "name": "<file_name_variant_1>",
    "path": "<path>"
  },
  "variant_2": {
    "name": "<file_name_variant_2>",
    "path": "<path>"
  }
}
```

Then the path of `variants_cfg.json` file has to be added as input parameter of `testsuites.testsuite_setup` in `Suite Setup` of a testsuite.

In case of user wants to set configuration level 2 for entire RobotFramework test project instead of indiviual robot testsuite file, `__init__.robot` file has to be created at the highest folder of RobotFrameowork test project, and the path of `variants_cfg.json` file has to be added as input parameter of `testsuites.testsuite_setup` in `Suite Setup` of the `__init__.robot` file.

```
*** Settings ***
Library      RobotFramework_Testsuites    WITH NAME    testsuites
Suite Setup      testsuites.testsuite_setup    <Path_to_the_file_variants_cfg.json>
```

Level 3: Find the "config/" folder in current testsuite directory

Configuration level 3 is triggered only in case of level 1 and level 2 were not set.

The configuration level 3 will check in `config/` folder in current testsuite directory the existence of json file which has the same name with testsuite file (ex: `abc.rotbot` & `./config/abc.json`), then it will load this configuration file. In case there is no json file has the same name with robot testsuite file, it will check the existence of `./config/robot_config.json` then load this `./config/robot_config.json` file as configuration file.

Level 4: Lowest priority level, it reads default configuration file

In case testsuites management library detects that configuration level 1, level 2, and level 3 are not set, the robot execution will use the configuration level 4 by default.

The default configuration file (`robot_config.json`) in installation directory:

`\RobotFramework_Testsuites\Config\robot_config.json`

2.2.3 Access to configuration parameters

User can access dictionary object which is defined in configuration file in robot test script by traditional way or using ".". For example, users can call `${dict}[abc][def]` or `${dict.abc.def}`

Note: In case a parameter name contains a ".", then it is not possible to use dotdict but the traditional way `${dict}[abc][def]` is still working.

Chapter 3

CConfig.py

3.1 Class: dotdict

Imported by:

```
from RobotFramework_Testsuites.Config.CConfig import dotdict
```

Subclass: dotdict

Subclass of dict, with "dot" (attribute) access to keys.

3.2 Class: CConfig

Imported by:

```
from RobotFramework_Testsuites.Config.CConfig import CConfig
```

Class: CConfig

Defines the properties of configuration and holds the identified config files.

The loading configuration method is divided into 4 levels, level1 is highest priority, Level4 is lowest priority.

Level1: Handed over by command line argument.

Level2: Read from content of json config file

```
{
   "default": {
      "name": "robot_config.json",
      "path": ".../config/"
   },
   "variant_0": {
      "name": "robot_config.json",
      "path": ".../config/"
   },
   "variant_1": {
      "name": "robot_config_variant_1.json",
      "path": ".../config/"
   },
      ...
      ...
}
```

According to the `ConfigName`, Testsuites-Management package will choose the corresponding config file. `".../config/"` indicats the relative path to json config file, Testsuites-Management will recursively find the `config` folder.

Level3: Read in testsuite folder `/config/robot_config.json`

Level4: Read from RobotFramework AIO install folder `/RobotFramework/defaultconfig/robot_config.json`

3.2.1 Method: loadCfg

Method: loadCfg

This loadCfg method uses to load configuration's parameters from json files.

Arguments:

- No input parameter is required

Returns:

- No return variable

3.2.2 Method: updateCfg

Method: updateCfg

This updateCfg method updates preprocessor, global or local params base on RobotFramework AIO local config or any json config file according to purpose of specific testsuite.

Arguments:

- `sUpdateCfgFile`

  / *Condition*: required / *Type*: string
  The path of json file which wants to update configuration parameters.

Returns:

- No return variable

3.2.3 Method: verifyRbfwVersion

Method: verifyRbfwVersion

This verifyRbfwVersion validates the current RobotFramework AIO version with maximum and minimum version (if provided in the configuration file).

In case the current version is not between min and max version, then the execution of testsuite is terminated with "unknown" state

Arguments:

- No input parameter is required

Returns:

- No return variable

3.2.4 Method: bValidateMinVersion

Method: bValidateMinVersion

This bValidateMinVersion validates the current version with required minimun version.

Arguments:

- `tCurrentVersion`

  / *Condition*: required / *Type*: tuple
  Current RobotFramework AIO version.

- `tMinVersion`

  / *Condition*: required / *Type*: tuple
  The minimum version of RobotFramework AIO.

Returns:

- `True or False`

3.2.5 Method: bValidateMaxVersion

Method: bValidateMaxVersion

This bValidateMaxVersion validates the current version with required minimun version.

Arguments:

- `tCurrentVersion`

  / *Condition*: required / *Type*: tuple
  Current RobotFramework AIO version.

- `tMinVersion`

  / *Condition*: required / *Type*: tuple
  The minimum version of RobotFramework AIO.

Returns:

- `True or False`

3.2.6 Method: bValidateSubVersion

Method: bValidateSubVersion

This bValidateSubVersion validates the format of provided sub version and parse it into sub tuple for version comparision.

Arguments:

- `sVersion`

  / *Condition*: required / *Type*: string
  The version of RobotFramework AIO.

Returns:

- `lSubVersion`

  / *Type*: tuple /

3.2.7 Method: tupleVersion

Method: tupleVersion

This tupleVersion returns a tuple which contains the (major, minor, patch) version.
(remaining content needs to be fixed and restored)

Arguments:

- `sVersion`

  / *Condition*: required / *Type*: string
  The version of RobotFramework AIO.

Returns:

- `lVersion`

  / *Type*: tuple /

3.2.8 Method: versioncontrol_error

Method: versioncontrol_error

Wrapper version control error log:

Log error message of version control due to reason and set to unknown state.
reason can only be "conflict_min", "conflict_max" and "wrong_minmax".

Arguments:

- `reason`

  / *Condition*: required / *Type*: string

- `version1`

  / *Condition*: required / *Type*: string

- `version2`

  / *Condition*: required / *Type*: string

Returns:

- No return variable

Chapter 4

COnFailureHandle.py

4.1 Class: COnFailureHandle

Imported by:

```
from RobotFramework_Testsuites.Keywords.COnFailureHandle import COnFailureHandle
```

4.1.1 Method: is_noney

Chapter 5

CSetup.py

5.1 Class: CSetupKeywords

Imported by:

```
from RobotFramework_Testsuites.Keywords.CSetup import CSetupKeywords
```

Class: CSetupKeywords

This CSetupKeywords class uses to define the setup keywords which are using in suite setup and teardown of robot test script.

`Testsuite Setup` keyword loads the RobotFramework AIO configuration, checks the version of RobotFramework AIO, and logs out the basic information of the robot run.

`Testsuite Teardown` keyword currently do nothing, it's defined here for future requirements.

`Testcase Setup` keyword currently do nothing, it's defined here for future requirements.

`Testcase Teardown` keyword currently do nothing, it's defined here for future requirements.

5.1.1 Keyword: testsuite_setup

Method: testsuite_setup

This testsuite_setup defines the `Testsuite Setup` which is used to loads the RobotFramework AIO configuration, checks the version of RobotFramework AIO, and logs out the basic information of the robot run.

Arguments:

- `sTestsuiteCfgFile`

  / *Condition*: required / *Type*: string

  `sTestsuiteCfgFile=''` and vairiable `config_file` is not set Robotframework AIO will check for co
  level 3, and level 4.

  `sTestsuiteCfgFile` is set with a <json_config_file_path> and vairiable `config_file` is not set Robotframework AIO will load configuration level 2.

Returns:

- No return variable

5.1.2 Keyword: testsuite_teardown

Method: testsuite_teardown

This testsuite_teardown defines the `Testsuite Teardown` keyword, currently this keyword does nothing, it's defined here for future requirements.

5.1.3 Keyword: testcase_setup

Method: testcase_setup

This testcase_setup defines the `Testcase Setup` keyword, currently this keyword does nothing, it's defined here for future requirements.

5.1.4 Keyword: testcase_teardown

Method: testcase_teardown

This testcase_teardown defines the `Testcase Teardown` keyword, currently this keyword does nothing, it's defined here for future requirements.

5.1.5 Keyword: update_config

Method: update_config

This update_config defines the `Update Config` keyword which is using update the configuration object of RobotFramework AIO.

Arguments:

- `sCfgFile`

  / *Condition*: required / *Type*: string
  The path of Json configuration file.

Returns:

- No return variable

5.2 Class: CGeneralKeywords

Imported by:

```
from RobotFramework_Testsuites.Keywords.CSetup import CGeneralKeywords
```

Class: CGeneralKeywords

This CGeneralKeywords class defines the keywords which will be using in RobotFramework AIO test script.

`Get Config` keyword gets the current config object of robot run.

`Load Json` keyword loads json file then return json object.

In case new robot keyword is required, it will be defined and implemented in this class.

5.2.1 Keyword: get_config

Method: get_config

This get_config defines the `Get Config` keyword gets the current config object of RobotFramework AIO.

Arguments:

- No parameter is required

Returns:

- `oConfig.oConfigParams`

  / *Type*: json /

5.2.2 Keyword: load_json

Method: load_json

This load_json defines the `Load Json` keyword which loads json file then return json object.

Arguments:

- `jsonfile`

  / *Condition*: required / *Type*: string
  The path of Json configuration file.

- `level`

  / *Condition*: required / *Type*: int
  Level = 1 -> loads the content of jsonfile.
  level != 1 -> loads the json file which is set with variant (likes loading config level2)

Returns:

- `oJsonData`

  / *Type*: json /

Chapter 6

CStruct.py

6.1 Class: CStruct

Imported by:

```
from RobotFramework_Testsuites.Utils.CStruct import CStruct
```

Class: CStruct

This `CStruct` class creates the given attributes dynamically at runtime.

Chapter 7

Event.py

7.1 Class: Event

Imported by:

```
from RobotFramework_Testsuites.Utils.Events.Event import Event
```

7.1.1 Method: trigger

Chapter 8

ScopeEvent.py

8.1 Class: ScopeEvent

Imported by:

```
from RobotFramework_Testsuites.Utils.Events.ScopeEvent import ScopeEvent
```

8.1.1 Method: trigger

8.2 Class: ScopeStart

Imported by:

```
from RobotFramework_Testsuites.Utils.Events.ScopeEvent import ScopeStart
```

8.3 Class: ScopeEnd

Imported by:

```
from RobotFramework_Testsuites.Utils.Events.ScopeEvent import ScopeEnd
```

Chapter 9

__init__.py

9.1 Function: on

9.2 Function: dispatch

9.3 Function: register_event

Chapter 10

LibListener.py

10.1 Class: LibListener

Imported by:

```
from RobotFramework_Testsuites.Utils.LibListener import LibListener
```

Class: LibListener

This `LibListener` class defines the hook methods.

- `_start_suite` hooks to every starting testsuite of robot run.
- `_end_suite` hooks to every ending testsuite of robot run.
- `_start_test` hooks to every starting test case of robot run.
- `_end_test` hooks to every ending test case of robot run.

Chapter 11

__init__.py

11.1 Class: RobotFramework_Testsuites

Imported by:

```
from RobotFramework_Testsuites.__init__ import RobotFramework_Testsuites
```

Class: RobotFramework_Testsuites

RobotFramework_Testsuites is the Bosch testing library for Robot Framework.
RobotFramework_Testsuites control peripheral devices, tools and target under testing.

11.1.1 Method: run_keyword

11.1.2 Method: get_keyword_tags

11.1.3 Method: get_keyword_documentation

11.1.4 Method: failure_occurred

11.2 Class: CTestsuitesCfg

Imported by:

```
from RobotFramework_Testsuites.__init__ import CTestsuitesCfg
```

Chapter 12

version.py

12.1 Function: robfwaio_version

Return testsuitemanagement version as Robot framework AIO version

Chapter 13

Appendix

About this package:

Table 13.1: Package setup

Setup parameter	Value
Name	RobotFramework_Testsuites
Version	0.2.2
Date	18.07.2022
Description	Functionality to manage RobotFramework testsuites
Package URL	robotframework-testsuitesmanagement
Author	Mai Dinh Nam Son
Email	son.maidinhnam@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 14

History

0.1.0	06/2022
Initial version	

RobotFramework_Testsuites.pdf
Created at 13.12.2022 - 16:29:04
by GenPackageDoc v. 0.38.0

4.6 QConnectBase

QConnectBase

v. 1.1.1

Nguyen Huynh Tri Cuong

12.09.2022

Contents

CONTENTS CONTENTS

Chapter 1

Introduction

QConnectBaseLibrary is a connection testing library for Robot Framework. Library will be supported to downloaded from PyPI soon. It provides a mechanism to handle trace log continously receiving from a connection (such as Raw TCP, SSH, Serial, etc.) besides sending data back to the other side. It's especially efficient for monitoring the overflood response trace log from an asynchronous trace systems. It is supporting Python 3.7+ and RobotFramework 3.2+.

Chapter 2

Description

2.1 Getting Started

We have a plan to publish all the sourcecode as OSS in the near future so that you can downloaded from PyPI. For the current period, you can checkout

QConnectBaseLibrary

After checking out the source completely, you can install by running below command inside **robotframework-qconnect-base** directory.

```
python setup.py install
```

2.2 Usage

QConnectBase Library support following keywords for testing connection in RobotFramework.

2.2.1 connect

Use for establishing a connection.

Syntax:

connect `[conn_name]` `[conn_type]` `[conn_mode]` `[conn_conf]` *(All parameters are required to be in order)* or

connect `conn_name=[conn_name]` `conn_type=[conn_type]` `conn_mode=[conn_mode]` `conn_conf=[conn_conf]` *(All parameters are assigned by name)*

Arguments:

conn_name: Name of the connection.

conn_type: Type of the connection. QConnectBaseLibrary has supported below connection types:

- **TCPIPClient**: Create a Raw TCPIP connection to TCP Server.
- **SSHClient**: Create a client connection to a SSH server.
- **SerialClient**: Create a client connection via Serial Port.

conn_mode: (unused) Mode of a connection type.

conn_conf: Configurations for making a connection. Depend on **conn_type** (Type of Connection), there is a suitable configuration in JSON format as below.

- **TCPIPClient**

```
{
    "address": [server host], # Optional. Default value is ↩
    ↪ "localhost".
    "port": [server port]     # Optional. Default value is 1234.
    "logfile": [Log file path. Possible values: 'nonlog', ↩
    ↪ 'console', [user define path] ]
 }
```

- **SSHClient**

```
{
    "address" : [server host],  # Optional. Default value is "localhost".
    "port" : [server host],     # Optional. Default value is 22.
    "username" : [username],    # Optional. Default value is "root".
    "password" : [password],    # Optional. Default value is "".
    "authentication" : "password" | "keyfile" | "passwordkeyfile",  # Optional. Default value is "".
    "key_filename" : [filename or list of filenames], # Optional. Default value is null.
    "logfile": [Log file path. Possible values: 'nonlog', 'console', [user define path] ]
}
```

- **SerialClient**

```
{
    "port" : [comport or null],
    "baudrate" : [Baud rate such as 9600 or 115200 etc.],
    "bytesize" : [Number of data bits. Possible values: 5, 6, 7, 8],
    "stopbits" : [Number of stop bits. Possible values: 1, 1.5, 2],
    "parity" : [Enable parity checking. Possible values: 'N', 'E', 'O', 'M', 'S'],
    "rtscts" : [Enable hardware (RTS/CTS) flow control.],
    "xonxoff" : [Enable software flow control.],
    "logfile": [Log file path. Possible values: 'nonlog', 'console', [user define path] ]
}
```

2.2.2 disconnect

Use for disconnect a connection by name.

Syntax:

disconnect `conn_name`

Arguments:

conn_name: Name of the connection.

2.2.3 send command

Use for sending a command to the other side of connection.

Syntax:

send command `[conn_name] [command]` *(All parameters are required to be in order)* or
send command `conn_name=[conn_name] command=[command]` *(All parameters are assigned by name)*

Arguments:

conn_name: Name of the connection.
command: Command to be sent.

2.2.4 verify

Use for verifying a response from the connection if it matched a pattern.

Syntax:

verify `[conn_name] [search_pattern] [timeout] [fetch_block] [eob_pattern] [filter_pattern] [send_cmd]` *(All parameters are required to be in order)* or
verify `conn_name=[conn_name] search_pattern=[search_pattern] timeout=[timeout] fetch_block=[fetch_block] eob_pattern=[eob_pattern] filter_pattern=[filter_pattern] send_cmd=[send_cmd]` *(All parameters are assigned by name)*

Arguments:

conn_name: Name of the connection.
search_pattern: Regular expression for matching with the response.
timeout: Timeout for waiting response matching pattern.
fetch_block: If this value is true, every response line will be put into a block untill a line match **eob_pattern** pattern.
eob_pattern: Regular expression for matching the endline when using **fetch_block**.
filter_pattern: Regular expression for filtering every line of block when using **fetch_block**.
send_cmd: Command to be sent to the other side of connection and waiting for response.

Return value:

A corresponding match object if it is found.
E.g.

```
${result} = verify  conn_name=SSH_Connection
                    search_pattern=(?<=\s).*([0-9]..).*(command).$
                    send_cmd=*echo This is the 1st test command.*
```

- ${result}[0] will be **"This is the 1st test command."** which is the matched string.
- ${result}[1] will be **"1st"** which is the first captured string.
- ${result}[2] will be **"command"** which is the second captured string.

2.3 Example

```
*** Settings ***
Documentation    Suite description
Library     QConnectBase.ConnectionManager

*** Test Cases ***
Test SSH Connection
    # Create config for connection.
    ${config_string}=    catenate
    ...  {
    ...   "address": "127.0.0.1",
    ...   "port": 8022,
    ...   "username": "root",
    ...   "password": "",
    ...   "authentication": "password",
    ...   "key_filename": null
    ...  }
    log to console       \nConnecting with configurations:\n${config_string}
    ${config}=             evaluate        json.loads('''${config_string}''')    json

    # Connect to the target with above configurations.
    connect             conn_name=test_ssh
    ...                 conn_type=SSHClient
    ...                 conn_conf=${config}

    # Send command 'cd ..' and 'ls' then wait for the response '.*' pattern.
    send command                conn_name=test_ssh
    ...                         command=cd ..

    ${res}=     verify                  conn_name=test_ssh
    ...                                 search_pattern=.*
    ...                                 send_cmd=ls
```

```
    log to console      ${res}

    # Disconnect
    disconnect  test_ssh
```

Listing 2.1: Robot code example

2.4 Contribution Guidelines

QConnectBaseLibrary is designed for ease of making an extension library. By that way you can take advantage of the QConnectBaseLibrary's infrastructure for handling your own connection protocal. For creating an extension library for QConnectBaseLibrary, please following below steps.

1. Create a library package which have the prefix name is **robotframework-qconnect-***[your specific name]*.
2. Your hadling connection class should be derived from **QConnectionLibrary.connection_base.ConnectionBase** class.
3. In your *Connection Class*, override below attributes and methods:

   - **_CONNECTION_TYPE**: name of your connection type. It will be the input of the conn_type argument when using **connect** keyword. Depend on the type name, the library will detemine the correct connection handling class.
   - **__init__(self, _mode, config)**: in this constructor method, you should:
     - Prepare resource for your connection.
     - Initialize receiver thread by calling **self._init_thread_receiver(cls._socket_instance, mode="")** method.
     - Configure and initialize the lowlevel receiver thread (if it's necessary) as below

       ```
       self._llrecv_thrd_obj = None
        self._llrecv_thrd_term = threading.Event()
        self._init_thrd_llrecv(cls._socket_instance)
       ```

     - Incase you use the lowlevel receiver thread. You should implement the **thrd_llrecv_from_connection_interface** method. This method is a mediate layer which will receive the data from connection at the very beginning, do some process then put them in a queue for the **receiver thread** above getting later.
     - Create the queue for this connection (use Queue.Queue).
   - **connect()**: implement the way you use to make your own connection protocol.
   - **_read()**: implement the way to receive data from connection.
   - **_write()**: implement the way to send data via connection.
   - **disconnect()**: implement the way you use to disconnect your own connection protocol.
   - **quit()**: implement the way you use to quit connection and clean resource.

2.5 Configure Git and correct EOL handling

Here you can find the references for Dealing with line endings.

Every time you press return on your keyboard you're actually inserting an invisible character called a line ending. Historically, different operating systems have handled line endings differently. When you view changes in a file, Git handles line endings in its own way. Since you're collaborating on projects with Git and GitHub, Git might produce unexpected results if, for example, you're working on a Windows machine, and your collaborator has made a change in OS X.

To avoid problems in your diffs, you can configure Git to properly handle line endings. If you are storing the .gitattributes file directly inside of your repository, than you can asure that all EOL are manged by git correctly as defined.

2.6 Sourcecode Documentation

For investigating sourcecode, please refer to QConnectBase library documentation

2.7 Feedback

If you have any problem when using the library or think there is a better solution for any part of the library, I'd love to know it, as this will all help me to improve the library. Please don't hesitate to contact me.

Do share your valuable opinion, I appreciate your honest feedback!

2.8 About

2.8.1 Maintainers

Nguyen Huynh Tri Cuong

2.8.2 Contributors

Nguyen Huynh Tri Cuong

Thomas Pollerspöck

2.9 License

Chapter 3

__init__.py

3.1 Class: ConnectionManager

Imported by:

```
from QConnectBase.__init__ import ConnectionManager
```

Class to manage all connections.

Chapter 4

connection_base.py

4.1 Class: BrokenConnError

Imported by:

```
from QConnectBase.connection_base import BrokenConnError
```

4.2 Class: ConnectionBase

Imported by:

```
from QConnectBase.connection_base import ConnectionBase
```

Base class for all connection classes.

4.2.1 Method: is_supported_platform

Check if current platform is supported.

Returns:

/ *Type*: bool /
True if platform is supported.
False if platform is not supported.

4.2.2 Method: is_precondition_pass

Check for precondition.

Returns:

/ *Type*: bool /
True if passing the precondition.
False if failing the precondition.

4.2.3 Method: error_instruction

Get the error instruction.

Returns:

/ *Type*: str /
Error instruction string.

4.2.4 Method: quit

>> This method MUST be overridden in derived class <<
Abstract method for quiting the connection.
Arguments:

- `is_disconnect_all`
  / *Condition*: optional / *Type*: bool /
  Determine if it's necessary to disconnect all connections.

Returns:
(*no returns*)

4.2.5 Method: connect

>> This method MUST be overridden in derived class <<
Abstract method for quiting the connection.
Arguments:

- `device`
  / *Condition*: required / *Type*: str /
  Device name.
- `files`
  / *Condition*: optional / *Type*: list /
  Trace file list if using dlt connection.
- `test_connection`
  / *Condition*: optional / *Type*: bool /
  Determmine if it's necessary for testing the connection.

Returns:
(*no returns*)

4.2.6 Method: disconnect

>> This method MUST be overridden in derived class <<
Abstract method for disconnecting connection.
Arguments:

- `n_thrd_id`
  / *Condition*: required / *Type*: int /
  Thread id.

Returns:
(*no returns*)

4.2.7 Method: send_obj

Wrapper method to send message to a tcp connection.
Arguments:

- `obj`
  / *Condition*: required / *Type*: str /
  Data to be sent.
- `cr`
  / *Condition*: optional / *Type*: str /
  Determine if it's necessary to add newline character at the end of command.

Returns:

(*no returns*)

4.2.8 Method: read_obj

Wrapper method to get the response from connection.

Returns:

- `msg`
  / *Type*: str /
  Responded message.

4.2.9 Method: wait_4_trace

Suspend the control flow until a Trace message is received which matches to a specified regular expression.

Arguments:

- `search_obj`
  / *Condition*: required / *Type*: str /
  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.
- `use_fetch_block`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Determine if 'fetch block' feature is used.
- `end_of_block_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  The end of block pattern.
- `filter_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  Pattern to filter message line by line.
- `timeout`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Timeout parameter specified as a floating point number in the unit 'seconds'.
- `fct_args`
  / *Condition*: optional / *Type*: Tuple / *Default*: None /
  List of function arguments passed to be sent.

Returns:

- `match`
  / *Type*: re.Match /
  If no trace message matched to the specified regular expression and a timeout occurred, return None.
  If a trace message has matched to the specified regular expression, a match object is returned as the result.The complete trace message can be accessed by the 'string' attribute of the match object. For access to groups within the regular expression, use the group() method. For more information, refer to Python documentation for module 're'.

4.2.10 Method: wait_4_trace_continuously

Getting trace log continuously without creating a new trace queue.

Arguments:

- `trace_queue`
  / *Condition*: required / *Type*: Queue /
  Queue to store the traces.
- `timeout`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Timeout for waiting a matched log.
- `fct_args`
  / *Condition*: optional / *Type*: Tuple / *Default*: None /
  Arguments to be sent to connection.

Returns:

- `None`
  / *Type*: None /
  If no trace message matched to the specified regular expression and a timeout occurred.
- `match`
  / *Type*: re.Match /
  If a trace message has matched to the specified regular expression, a match object is returned as the result.The complete trace message can be accessed by the 'string' attribute of the match object. For access to groups within the regular expression, use the group() method. For more information, refer to Python documentation for module 're'.

4.2.11 Method: create_and_activate_trace_queue

Create Queue and assign it to _trace_queue object and activate the queue with the search element.

Arguments:

- `search_element`
  / *Condition*: required / *Type*: str /
  Regular expression all received trace messages are compare to.
  Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.#
- `use_fetch_block`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Determine if 'fetch block' feature is used.
- `end_of_block_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  The end of block pattern.
- `regex_line_filter_pattern`
  / *Condition*: optional / *Type*: re.Pattern / *Default*: None /
  Regular expression object to filter message line by line.

Returns:

- `trq_handle, trace_queue`
  / *Type*: tuple /
  The handle and search object

4.2.12 Method: deactivate_and_delete_trace_queue

Deactivate trace queue and delete.

Arguments:

- `trq_handle`
  / *Condition*: required / *Type*: int /
  Trace queue handle.
- `trace_queue`
  / *Condition*: required / *Type*: Queue /
  Trace queue object.

Returns:

(*no returns*)

4.2.13 Method: activate_trace_queue

Activates a trace message filter specified as a regular expression. All matching trace messages are put in the specified queue object.

Arguments:

- `search_obj`
  / *Condition*: required / *Type*: str /
  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.
- `trace_queue`
  / *Condition*: required / *Type*: Queue /
  A queue object all trace message which matches the regular expression are put in. The using application must assure, that the queue is emptied or deleted.
- `use_fetch_block`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Determine if 'fetch block' feature is used.
- `end_of_block_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  The end of block pattern.
- `line_filter_pattern`
  / *Condition*: optional / *Type*: re.Pattern / *Default*: None /
  Regular expression object to filter message line by line.

Returns:

- `handle_id`
  / *Type*: int /
  Handle to deactivate the message filter.

4.2.14 Method: deactivate_trace_queue

Deactivates a trace message filter previously activated by ActivateTraceQ() method.

Arguments:

- `handle`
  / *Condition*: required / *Type*: int /
  Integer object returned by ActivateTraceQ() method.

Returns:

* `is_success`

  / *Type*: bool / . False : No trace message filter active with the specified handle (i.e. handle is not in use).
  True : Trace message filter successfully deleted.

4.2.15 Method: check_timeout

>> This method will be override in derived class <<

Check if responded message come in cls._RESPOND_TIMEOUT or we will raise a timeout event.

Arguments:

- `timeout`
  / *Condition*: required / *Type*: int /
  Timeout in seconds.

Returns:

(*no returns*)

4.2.16 Method: pre_msg_check

>> This method will be override in derived class <<

Pre-checking message when receiving it from connection.

Arguments:

- `msg`
  / *Condition*: required / *Type*: str /
  Received message to be checked.

Returns:

(*no returns*)

4.2.17 Method: post_msg_check

>> This method will be override in derived class <<

Post-checking message when receiving it from connection.

Arguments:

- `msg`
  / *Condition*: required / *Type*: str /
  Received message to be checked.

Returns:

(*no returns*)

Chapter 5

connection_manager.py

5.1 Class: InputParam

Imported by:

```
from QConnectBase.connection_manager import InputParam
```

5.1.1 Method: get_attr_list

5.2 Class: ConnectParam

Imported by:

```
from QConnectBase.connection_manager import ConnectParam
```

Class for storing parameters for connect action.

5.3 Class: SendCommandParam

Imported by:

```
from QConnectBase.connection_manager import SendCommandParam
```

Class for storing parameters for send command action.

5.4 Class: VerifyParam

Imported by:

```
from QConnectBase.connection_manager import VerifyParam
```

Class for storing parameters for verify action.

5.5 Class: ConnectionManager

Imported by:

```
from QConnectBase.connection_manager import ConnectionManager
```

Class to manage all connections.

5.5.1 Method: quit

Quit connection manager.

Returns:

(*no returns*)

5.5.2 Method: add_connection

Add a connection to managed dictionary.

Arguments:

- `name`
  / *Condition*: required / *Type*: str /
  Connection's name.
- `conn`
  / *Condition*: required / *Type*: socket.socket /
  Connection object.

Returns:

(*no returns*)

5.5.3 Method: remove_connection

Remove a connection by name.

Arguments:

- `connection_name`
  / *Condition*: required / *Type*: str /
  Connection's name.

Returns:

(*no returns*)

5.5.4 Method: get_connection_by_name

Get an exist connection by name.

Arguments:

- `connection_name`
  / *Condition*: required / *Type*: str /
  Connection's name.

Returns:

- `conn`
  / *Type*: socket.socket /
  Connection object.

5.5.5 Keyword: disconnect

Keyword for disconnecting a connection by name.

Arguments:

- `connection_name`
  / *Condition*: required / *Type*: str /
  Connection's name.

Returns:

(*no returns*)

5.5.6 Keyword: connect

Keyword for making a connection.

Arguments:

(*refer to connect_unnamed_args method for details*)

- `args`
  / *Condition*: required / *Type*: tuple /
  Non-Keyword Arguments.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

Returns:

(*no returns*)

5.5.7 Method: connect_named_args

Making a connection with name arguments.

Arguments:

(*refer to connect_unnamed_args method for details*)

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

Returns:

(*no returns*)

5.5.8 Method: connect_unnamed_args

Making a connection.

Arguments:

- `connection_name`
  / *Condition*: required / *Type*: str /
  Name of connection.
- `connection_type`
  / *Condition*: required / *Type*: str /
  Type of connection.

- `mode`
  / *Condition*: required / *Type*: str /
  Connection mode.
- `config`
  / *Condition*: required / *Type*: json /
  Configuration for connection.

Returns:
(*no returns*)

5.5.9 Keyword: send_command

Keyword for sending command to a connection.
Arguments:
(*refer to send_unnamed_args method for details*)

- `args`
  / *Condition*: require / *Type*: tuple /
  Non-Keyword Arguments.
- `kwargs`
  / *Condition*: require / *Type*: dict /
  Keyword Arguments.

Returns:
(*no returns*)

5.5.10 Method: send_command_named_args

Send command to a connection with name arguments.
Arguments:
(*refer to send_unnamed_args method for details*)

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

Returns:
(*no returns*)

5.5.11 Method: send_command_unnamed_args

Send command to a connection.
Arguments:

- `connection_name`
  / *Condition*: required / *Type*: str /
  Name of connection.
- `command`
  / *Condition*: required / *Type*: str /
  Command to be sent.

Returns:
(*no returns*)

5.5.12 Keyword: verify

Keyword uses to verify a pattern from connection response after sending a command.

Arguments:

(*refer to verify_unnamed_args method for details*)

- `args`
  / *Condition*: required / *Type*: tuple /
  Non-Keyword Arguments.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

Returns:

- `match_res`
  / *Type*: str /
  Matched string.

5.5.13 Method: verify_named_args

Verify a pattern from connection response after sending a command with named arguments.

Arguments:

(*refer to verify_unnamed_args method for details*)

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

Returns:

- `match_res`
  / *Type*: str /
  Matched string.

5.5.14 Method: verify_unnamed_args

Verify a pattern from connection response after sending a command.

Arguments:

- `connection_name`
  / *Condition*: required / *Type*: str /
  Name of connection.
- `search_obj`
  / *Condition*: required / *Type*: str /
  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.
- `fetch_block`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Determine if 'fetch block' feature is used.

- `eob_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  The end of block pattern.
- `filter_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  Pattern to filter message line by line.
- `timeout`
  / *Condition*: optional / *Type*: float / *Default*: 0 /
  Timeout parameter specified as a floating point number in the unit 'seconds'.
- `fct_args`
  / *Condition*: optional / *Type*: Tuple / *Default*: None /
  List of function arguments passed to be sent.

Returns:

- `match_res`
  / *Type*: str /
  Matched string.

5.6 Class: TestOption

Imported by:

```
from QConnectBase.connection_manager import TestOption
```

Chapter 6

constants.py

6.1 Class: SocketType

Imported by:

```
from QConnectBase.constants import SocketType
```

6.2 Class: String

Imported by:

```
from QConnectBase.constants import String
```

Chapter 7

qlogger.py

7.1 Class: ColorFormatter

Imported by:

```
from QConnectBase.qlogger import ColorFormatter
```

Custom formatter class for setting log color.

7.1.1 Method: format

Set the color format for the log.
Arguments:

- `record`
  / *Condition*: required / *Type*: str /
  Log record.

Returns:

/ *Type*: logging.Formatter /
Log with color formatter.

7.2 Class: QFileHandler

Imported by:

```
from QConnectBase.qlogger import QFileHandler
```

Handler class for user defined file in config.

7.2.1 Method: get_log_path

Get the log file path for this handler.
Arguments:

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

Returns:

/ *Type*: str /
Log file path.

7.2.2 Method: get_config_supported

Check if the connection config is supported by this handler.
Arguments:

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

Returns:

/ *Type*: bool /
True if the config is supported.
False if the config is not supported.

7.3 Class: QDefaultFileHandler

Imported by:

```
from QConnectBase.qlogger import QDefaultFileHandler
```

Handler class for default log file path.

7.3.1 Method: get_log_path

Get the log file path for this handler.
Arguments:

- `logger_name`
  / *Condition*: required / *Type*: str /
  Name of the logger.

Returns:

/ *Type*: str /
Log file path.

7.3.2 Method: get_config_supported

Check if the connection config is supported by this handler.
Arguments:

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

Returns:

/ *Type*: bool /
True if the config is supported.
False if the config is not supported.

7.4 Class: QConsoleHandler

Imported by:

```
from QConnectBase.qlogger import QConsoleHandler
```

Handler class for console log.

7.4.1 Method: get_config_supported

Check if the connection config is supported by this handler.
Arguments:

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

Returns:

/ *Type*: bool /
True if the config is supported.
False if the config is not supported.

7.5 Class: QLogger

Imported by:

```
from QConnectBase.qlogger import QLogger
```

Logger class for QConnect Libraries.

7.5.1 Method: get_logger

Get the logger object.
Arguments:

- `logger_name`
  / *Condition*: required / *Type*: str /
  Name of the logger.

Returns:

- `logger`
  / *Type*: Logger /
  Logger object. .

7.5.2 Method: set_handler

Set handler for logger.
Arguments:

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

Returns:

- `handler_ins`
  / *Type*: logging.handler /
  None if no handler is set.
  Handler object.

Chapter 8

serial_base.py

8.1 Class: SerialConfig

Imported by:

```
from QConnectBase.serialclient.serial_base import SerialConfig
```

Class to store the configuration for Serial connection.

8.2 Class: SerialSocket

Imported by:

```
from QConnectBase.serialclient.serial_base import SerialSocket
```

Class for handling serial connection.

8.2.1 Method: connect

Connect to serial port.
Returns:
(*no returns*)

8.2.2 Method: disconnect

Disconnect serial port.
Arguments:

- `_device`
  / *Condition*: required / *Type*: str /
  Unused

Returns:
(*no returns*)

8.2.3 Method: quit

Quit serial connection.
Returns:
(*no returns*)

8.3 Class: SerialClient

Imported by:

```
from QConnectBase.serialclient.serial_base import SerialClient
```

Serial client class.

8.3.1 Method: connect

Connect to the Serial port.

Returns:

(*no returns*)

Chapter 9

raw_tcp.py

9.1 Class: RawTCPBase

Imported by:

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPBase
```

Base class for a raw tcp connection.

9.2 Class: RawTCPServer

Imported by:

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPServer
```

Class for a raw tcp connection server.

9.3 Class: RawTCPClient

Imported by:

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPClient
```

Class for a raw tcp connection client.

Chapter 10

ssh_client.py

10.1 Class: AuthenticationType

Imported by:

```
from QConnectBase.tcp.ssh.ssh_client import AuthenticationType
```

10.2 Class: SSHConfig

Imported by:

```
from QConnectBase.tcp.ssh.ssh_client import SSHConfig
```

Class to store the configuration for SSH connection.

10.3 Class: SSHClient

Imported by:

```
from QConnectBase.tcp.ssh.ssh_client import SSHClient
```

SSH client connection class.

10.3.1 Method: connect

Implementation for creating a SSH connection.

Returns:

(*no returns*)

10.3.2 Method: close

Close SSH connection.

Returns:

(*no returns*)

10.3.3 Method: quit

Quit and stop receiver thread.

Returns:

(*no returns*)

Chapter 11

tcp_base.py

11.1 Class: TCPConfig

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPConfig
```

Class to store configurations for TCP connection.

11.2 Class: TCPBase

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPBase
```

Base class for a tcp connection.

11.2.1 Method: close

Close connection.

Returns:

(*no returns*)

11.2.2 Method: quit

Quit connection.

Arguments:

- `is_disconnect_all`
  / *Condition*: required / *Type*: bool /
  Determine if it's necessary for disconnect all connection.

Returns:

(*no returns*)

11.2.3 Method: connect

>> Should be override in derived class.

Establish the connection.

Returns:

(*no returns*)

11.2.4 Method: disconnect

>> Should be override in derived class.

Disconnect the connection.

Returns:

(*no returns*)

11.3 Class: TCPBaseServer

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPBaseServer
```

Base class for TCP server.

11.3.1 Method: accept_connection

Wrapper method for handling accept action of TCP Server.

Returns:

(*no returns*)

11.3.2 Method: connect

11.3.3 Method: disconnect

11.4 Class: TCPBaseClient

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPBaseClient
```

Base class for TCP client.

11.4.1 Method: connect

11.4.2 Method: disconnect

Chapter 12

utils.py

12.1 Class: Singleton

Imported by:

```
from QConnectBase.utils import Singleton
```

Class to implement Singleton Design Pattern. This class is used to derive the TTFisClientReal as only a single instance of this class is allowed.

Disabled pyLint Messages: R0903: Too few public methods (%s/%s) Used when class has too few public methods, so be sure it's really worth it.

> This base class implements the Singleton Design Pattern required for the TTFisClientReal. Adding further methods does not make sense.

12.2 Class: DictToClass

Imported by:

```
from QConnectBase.utils import DictToClass
```

Class for converting dictionary to class object.

12.2.1 Method: validate

12.3 Class: Utils

Imported by:

```
from QConnectBase.utils import Utils
```

Class to implement utilities for supporting development.

12.3.1 Method: get_all_descendant_classes

Get all descendant classes of a class

Arguments: cls: Input class for finding descendants.

Returns:

> / *Type*: list /
> Array of descendant classes.

12.3.2 Method: get_all_sub_classes

Get all children classes of a class

Arguments:

- `cls`
  / *Condition*: required / *Type*: class /
  Input class for finding children.

Returns:

/ *Type*: list /
Array of children classes.

12.3.3 Method: is_valid_host

12.3.4 Method: execute_command

12.3.5 Method: kill_process

12.3.6 Method: caller_name

Get a name of a caller in the format module.class.method

Arguments:

- `skip`
  / *Condition*: required / *Type*: int /

  Specifies how many levels of stack to skip while getting caller name. skip=1 means "who calls me", skip=2 "who calls my caller" etc.

Returns:

/ *Type*: str /
An empty string is returned if skipped levels exceed stack height

12.3.7 Method: load_library

Load native library depend on the calling convention.

Arguments:

- `path`
  / *Condition*: required / *Type*: str /
  Library path.
- `is_stdcall`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  Determine if the library's calling convention is stdcall or cdecl.

Returns:

Loaded library object.

12.3.8 Method: is_ascii_or_unicode

Check if the string is ascii or unicode

Arguments: str_check: string for checking codecs: encoding type list

Returns:

/ *Type*: bool /
True : if checked string is ascii or unicode
False : if checked string is not ascii or unicode

12.4 Class: Job

Imported by:

```
from QConnectBase.utils import Job
```

12.4.1 Method: stop

12.4.2 Method: run

12.5 Class: ResultType

Imported by:

```
from QConnectBase.utils import ResultType
```

Result Types.

12.6 Class: ResponseMessage

Imported by:

```
from QConnectBase.utils import ResponseMessage
```

Response message class

12.6.1 Method: get_json

Convert response message to json
Returns:
Response message in json format

12.6.2 Method: get_data

Get string data result
Returns:
String result

12.6.3 Method: create_from_string

Chapter 13

Appendix

About this package:

Table 13.1: Package setup

Setup parameter	Value
Name	QConnectBase
Version	1.1.1
Date	12.09.2022
Description	Robot Framework test library for TCP, SSH, serial connection
Package URL	robotframework-qconnect-base
Author	Nguyen Huynh Tri Cuong
Email	cuong.nguyenhuynhtri@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 14

History

1.1.0	07/2022
Initial version	

QConnectBase.pdf
Created at 13.12.2022 - 16:29:12
by GenPackageDoc v. 0.38.0

4.7 RobotLog2RQM

RobotLog2RQM

v. 1.1.4

Tran Duy Ngoan

10.11.2022

Contents

Chapter 1

Introduction

RobotLog2RQM tool provides the ability to interact with RQM resources (test plan, test case, build, ...).

RobotLog2RQM tool uses RqmAPI to:

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

Chapter 2

Description

2.1 Sample Robot Framework Testcase:

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

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

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

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

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

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

Listing 2.1: Sample Robot Framework testcase

> **Hint**
>
> In case you are using RobotFramework AIO with RobotFramework_Testsuites library for executing Robot testcase(s). You do not need to define above highlighted `Metadata` in your Robot test case.
> These `Metadata` information will be defined implicitly within the `Suite Setup` bases on your environment and configuration in *.json file.

2.2 Tool features:

By default, tool will base on provided arguments (see Usage) and `tcid` information in Robot test case(s) to:

- Login the RQM server and verify the provided `project` , `testplan` .
- Create build record and test environment (if already provided in Robot test case and not existing on RQM) for execution.

- Create new Test Case Execution Record - TCER (if it is not existing) bases on test case ID and `testplan` ID.
- Create new Test Case Execution Result which contents the detail and result state of test case.
- Link all test case(s) to provided `testplan` .

Besides, **RobotLog2RQM** tool also supports to:

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

2.3 Robot Testcase information on RQM:

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

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

Table 2.1: RQM data & Robot Framework testcase/output.xml

Chapter 3

CRQM.py

3.1 Function: get_xml_tree

Parse xml object from file.

Arguments:

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

Returns:

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

3.2 Class: CRQMClient

Imported by:

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

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

Resoure type mapping:

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

3.2.1 Method: login

Log in RQM by provided user & password.

Arguments:

(*no arguments*)

Returns:

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

3.2.2 Method: verifyProjectName

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

Arguments:

(*no arguments*)

Returns:

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

3.2.3 Method: disconnect

Disconnect from RQM.

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.2.4 Method: config

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

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

Arguments:

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

- `updatetestcase`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If True, the information of testcase on RQM will be updated bases on robot testfile.
- `suite_id (optional)`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Testsuite ID of RQM project for importing result(s).

Returns:
(*no returns*)

3.2.5 Method: userURL

Return interaction URL of provided userID
Arguments:

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

Returns:

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

3.2.6 Method: integrationURL

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

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

Returns:

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

3.2.7 Method: webIDfromResponse

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

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

Arguments:

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

Returns:

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

3.2.8 Method: webIDfromGeneratedID

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

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

Arguments:

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

Returns:

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

3.2.9 Method: getResourceByID

Return data of provided resource and ID by GET method
Arguments:

- `resourrceType`
  / *Condition*: required / *Type*: str /
  The RQM resource type.

- `id`
  / *Condition*: required / *Type*: str /
  ID of resource.

Returns:

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

3.2.10 Method: getAllByResource

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

Arguments:

- `resourrceType`
  / *Condition*: required / *Type*: str /
  The RQM resource type.

Returns:

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

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

3.2.11 Method: getAllBuildRecords

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

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.2.12 Method: getAllConfigurations

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

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.2.13 Method: getAllTeamAreas

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

Example:

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

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.2.14 Method: addTeamAreaNode

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

Note: team-area information is case-casesensitive

Arguments:

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

Returns:

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

3.2.15 Method: createTestcaseTemplate

Return testcase template from provided information.

Arguments:

- `testcaseName`
  / *Condition*: required / *Type*: str /
  Testcase name.
- `sDescription`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Testcase description.
- `sComponent`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Component which testcase is belong to.
- `sFID`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Function ID (requirement ID) for linking.
- `sTeam`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Team name for linking.

- `sRobotFile`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Link to robot file on source control.
- `sTestType`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Test type information.
- `sASIL`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  ASIL information.
- `sOwnerID`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  User ID of testcase owner.
- `sTCtemplate`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Existing testcase template as xml string.
  If not provided, template file under RQM_templates is used as default.

Returns:

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

3.2.16 Method: createTCERTemplate

Return testcase execution record template from provided information.

Arguments:

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

Returns:

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

3.2.17 Method: createExecutionResultTemplate

Return testcase execution result template from provided information.

Arguments:

- `testcaseID`
  / *Condition*: required / *Type*: str /
  Testcase ID for linking.
- `testcaseName`
  / *Condition*: required / *Type*: str /
  Testcase name.
- `testplanID`
  / *Condition*: required / *Type*: str /
  Testplan ID for linking.
- `TCERID`
  / *Condition*: required / *Type*: str /
  Testcase execution record (TCER) ID for linking.
- `resultState`
  / *Condition*: required / *Type*: str /
  Testcase result status.
- `startTime`
  / *Condition*: required / *Type*: str /
  Testcase start time.
- `endTime`
  / *Condition*: optional / *Type*: str / *Default*: " /
  Testcase end time.
- `duration`
  / *Condition*: optional / *Type*: str / *Default*: " /
  Testcase duration.
- `testPC`
  / *Condition*: optional / *Type*: str / *Default*: " /
  Test PC which executed testcase.
- `testBy`
  / *Condition*: optional / *Type*: str / *Default*: " /
  User ID who executed testcase.
- `lastlog`
  / *Condition*: optional / *Type*: str / *Default*: " /
  Traceback information (for Failed testcase).

- `buildrecordID`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Build Record ID for linking.
- `sTeam`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Team name for linking.
- `sOwnerID`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  User ID of testcase owner.

Returns:

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

3.2.18 Method: createBuildRecordTemplate

Return build record template from provided build name.
Arguments:

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

Returns:

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

3.2.19 Method: createConfigurationTemplate

Return configuration - Test Environment template from provided configuration name.
Arguments:

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

Returns:

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

3.2.20 Method: createTSERTemplate

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

Arguments:

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

Returns:

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

3.2.21 Method: createTestsuiteResultTemplate

Return testsuite execution result template from provided configuration name.

Arguments:

- `testsuiteID`
  / *Condition*: required / *Type*: str /
  Testsuite ID.
- `testsuiteName`
  / *Condition*: required / *Type*: str /
  Testsuite name.
- `TSERID`
  / *Condition*: required / *Type*: str /
  Testsuite execution record (TSER) ID for linking.
- `lTCER`
  / *Condition*: required / *Type*: str /
  List of testcase execution records (TCER) for linking.
- `lTCResults`
  / *Condition*: required / *Type*: str /
  List of testcase results for linking.

- `startTime`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Testsuite start time.
- `endTime`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Testsuite end time.
- `duration`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  Testsuite duration.
- `sOwnerID`
  / *Condition*: optional / *Type*: str / *Default*: ” /
  User ID of testsuite owner.

Returns:

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

3.2.22 Method: createResource

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

Arguments:

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

Returns:

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

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

3.2.23 Method: createBuildRecord

Create new build record.

Arguments:

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

Returns:

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

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

3.2.24 Method: createConfiguration

Create new configuration - test environment.

Arguments:

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

Returns:

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

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

3.2.25 Method: updateResourceByID

Update data of provided resource and ID by PUT method.

Arguments:

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

Returns:

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

3.2.26 Method: linkListTestcase2Testplan

Link list of test cases to provided testplan ID.

Arguments:

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

Returns:

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

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

3.2.27 Method: linkListTestcase2Testsuite

Link list of test cases to provided testsuite ID

Arguments:

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

Returns:

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

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

Chapter 4

robotlog2rqm.py

4.1 Function: get_from_tags

Extract testcase information from tags.

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

Arguments:

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

Returns:

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

4.2 Function: convert_to_datetime

Convert time string to datetime.

Arguments:

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

Returns:

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

4.3 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.

4.4 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.

4.5 Function: process_suite

Process robot suite for importing to RQM.

Arguments:

- `RQMClient`
  / *Condition*: required / *Type*: RQMClient object/
  RQMClient object.
- `suite`
  / *Condition*: required / *Type*: TestSuite object/
  Robot suite object.

Returns:

(*no returns*)

4.6 Function: process_test

Process robot test for importing to RQM.

Arguments:

- `RQMClient`
  / *Condition*: required / *Type*: RQMClient object/
  RQMClient object.
- `test`
  / *Condition*: required / *Type*: TestCase object/
  Robot test object.

Returns:

(*no returns*)

4.7 Function: RobotLog2RQM

Import robot results from output.xml to RQM - IBM Rational Quality Manager.
Flow to import Robot results to RQM:

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

Arguments:

- `args`
  / *Condition*: required / *Type*: ArgumentParser object /
  Argument parser object which contains:
  - resultxmlfile : path to the xml result file or directory of result files to be imported.
  - host : RQM host url.
  - project : RQM project name.
  - user : user for RQM login.
  - password : user password for RQM login.
  - testplan : RQM testplan ID.
  - recursive : if True, then the path is searched recursively for log files to be imported.
  - createmissing : if True, then all testcases without fcid are created when importing.
  - updatetestcase : if True, then testcases information on RQM will be updated bases on robot testfile.
  - dryrun : if True, then just check the RQM authentication and show what would be done.

Returns:

(*no returns*)

4.8 Class: Logger

Imported by:

```
from RobotLog2RQM.robotlog2rqm import Logger
```

Logger class for logging message.

4.8.1 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*)

4.8.2 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*)

4.8.3 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*)

4.8.4 Method: log_error

Write error message to console/file output.

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

Returns:
(*no returns*)

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	RobotLog2RQM
Version	1.1.4
Date	10.11.2022
Description	Imports robot result(s) to IBM Rational Quality Manager (RQM)
Package URL	robotframework-robotlog2rqm
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	07/2022
Initial version	
1.1.1	01.08.2022
Rework repository's document bases on GenPackageDoc	
1.1.2	25.08.2022
- Correct indent of sourcode's docstring. *- Update new style for history.*	
1.1.3	13.10.2022
- Fix findings and enhance README and document files *- Change argument name 'outputfile' to 'resultxmlfile'*	
1.1.4	10.11.2022
Rename package to RobotLog2RQM	

RobotLog2RQM.pdf

Created at 13.12.2022 - 16:29:21

by GenPackageDoc v. 0.38.0

4.8 TMLLog2RobotLog

TMLLog2RobotLog

v. 1.2.0

Tran Duy Ngoan

29.08.2022

Contents

Chapter 1

Introduction

If you have already worked or are currently working with TML (Test Modelling Language) Framework and prefer to view the TML result(s) as report/log of Robot Framework or merge with the existing Robot Framework result (to have the common report), TMLLog2RobotLog tool helps to do it.

This tool provides ability to convert the TML result (*AutoTestSummary.log.xml*) file(s) to Robot Framework result (*output.xml*) file without losing any test execution result's information.

It requires only the TML result file(s)/folder which need to be converted as the mandatory argument.

Besides, tool also provides the features to:

- Combine or merge the converted results to the existing Robot Framework result file.
- Append new result(s) to existing testsuite of Robot Framework result file.
- Rename the root testsuite of new result file as your wish.

Chapter 2

Description

2.1 Installation & Usage:

This tool is a part of RobotFramework AIO and will be installed along with RobotFramework AIO installation.

However, I can install this tool alone as the instruction **"How to install"** in the README file of tool's repository.
You can also find the tool's usage and some examples for how to use it on above page.

For more detail about the features of this tool, please refer below Tool features section.

2.2 Tool features:

2.2.1 Convert TML result file(s) to Robot result file:

The simple command to convert the single TML result file to Robot result file: :

```
TMLLog2RobotLog path_to_TML_result_file
```

To convert all TML result files in the folder (use `--recursive` for serching recursively for result files in all sub folders):

```
TMLLog2RobotLog path_to_TML_result_folder
```

The output *robot_output.xml* file will be created under working folder as default.
However, you can specify the output file name with `--output path_to_new_file` as below (the `path_to_new_file` should not be existing):

```
TMLLog2RobotLog path_to_TML_result_file --output path_to_new_file
```

In case `path_to_new_file` is existing, the converted result will be combined with the existing Robot Framework result file.

2.2.2 Combine TML result file(s) with existing Robot Framework result file:

In case you already have the Robot Framework result file and want to combine the TML result(s) with it to get the union Robot Framework file.
Just specify the path to existing Robot Framework result file in `--output` argument as below:

```
TMLLog2RobotLog path_to_TML_result_file --output path_to_existing_Robot_result_file
```

The result will be combination of converted suite(s) and existing suite.
The root suite name will also be the combination name of all suite names.
In order to rename the root suite name, you can add `--name NEW_SUITE_NAME` for the better view on html log file and report.

2.2.3 Rename the root suite name for the output Robot log file:

By default, the file name of TML log file is used as suite name when converting.

Tool provides the ability to rename the suite name as your wish with `--name NAME` argument:

```
TMLLog2RobotLog path_to_TML_result_file --name suite_name
```

This feature will help in case you convert multiple TML log files in folder or combine with existing Robot log file.

The root suite name will be displayed simple as better as your wish on html log and report instead the long suite name such `File_name_1 & File_name_2 & File_name_3` or `File_name_1 & Existing_suite` .

2.2.4 Merge TML log file(s) into existing Robot log file:

In case you want to merge instead of combining with the existing Robot log file, just add --merge argument as below:

```
TMLLog2RobotLog path_to_TML_log_file --output path_to_existing_Robot_log_file --merge
```

2.2.5 Append TML results to existing Robot suite:

In case the Robot result file is existing and the results in TML result file is a part of the specific test suite of that Robot Framework result. You can convert and append to suite with `--appendtosuite` argument:

```
TMLLog2RobotLog path_to_TML_log_file --output path_to_existing_Robot_log_file ↩
    ↪ --appendtosuite existing_suitename_to_append
```

Chapter 3

resultbuilder.py

3.1 Function: TMLExecutionResult

Factory method to constructs ~.executionresult.Result objects.

This TMLExecutionResult method is based on the ExecutionResult method of Robot Framework.

Arguments:

- sources
  / *Condition*: required / *Type*: str /
  XML source(s) containing TML results. Can be specified as paths, opened file objects, or strings/bytes containing XML directly.
- options
  / *Condition*: required /
  Configuration options.

Returns:

/ *Type*: Result object /
Robot Framework result which is built from TML result(s).

3.2 Class: TMLExecutionResultBuilder

Imported by:

```
from TMLLog2RobotLog.resultbuilder import TMLExecutionResultBuilder
```

Build Robot Framework Result object from the TML execution result.

3.2.1 Method: build

Build Robot Framework Result object.

Arguments:

- result
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

- result
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object after parsing information from XML source.

Chapter 4

tml2robotlog.py

4.1 Function: process_commandline

Process provided argument(s) from command line.

Avalable arguments in command line:

- -v : tool version information.
- tmllogs : path to the TML log file or directory for conversion.
- --recursive : if set, then the path is searched recursively for log files to be converted.
- --output : path to Robotframework output file. If not provided, the robot output file is created in current directory as default.
- --name : name of the root suite of output. If not provided, the log file name is used as default.
- --merge : if set, merge the converted tml result to existing output file instead of combining results together
- --appendtosuite : suite name in existing output Robot to add the TML results into.

Arguments:

(*no arguments*)

Returns:

/ *Type*: ArgumentParser object /
ArgumentParser object.

4.2 Function: validate_cli_arguments

Validate the given arguments.

Arguments:

- `args`
  / *Condition*: required / *Type*: ArgumentParser object /
  ArgumentParser object from command line.

Returns:

(*no returns*)

4.3 Function: TMLLog2RobotLog

Convert TML log file(s) to Robot log file.

Flow for the conversion:

1. Process provided arguments from command line and validate them.
2. Process the CTMLLog2Robot converter.
3. Save converted result as output file.

Arguments:

- `args`
  / *Condition*: optional / *Type*: ArgumentParser object / *Default*: None /
  Argument parser object which contains:
  - tmllogs : path to the TML log file or directory for conversion.
  - recursive : if set, then the path is searched recursively for log files to be converted.
  - output : path to Robotframework output file. If not provided, the robot output file is created in current directory as default.
  - name : name of the root suite of output. If not provided, the log file name is used as default.
  - merge : if set, merge the converted tml result to existing output file instead of combining results together
  - appendtosuite : suite name in existing output Robot to add the TML results into.

Returns:

(*no returns*)

4.4 Class: Logger

Imported by:

```
from TMLLog2RobotLog.tml2robotlog import Logger
```

Logger class for logging message.

4.4.1 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*)

4.4.2 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*)

4.4.3 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*)

4.4.4 Method: log_error

Write error message to console/file output.

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

Returns:

(*no returns*)

4.5 Class: CTMLLog2Robot

Imported by:

```
from TMLLog2RobotLog.tml2robotlog import CTMLLog2Robot
```

CTMLLog2Robot class.

4.5.1 Method: config

Prepare configuration for robot result.

Arguments:

(*no arguments*)

Returns:

(*no returns*)

4.5.2 Method: run

Process TMLLog2Robot converter.

Arguments:

(*no arguments*)

Returns:

- `result`
  / *Type*: Result object /
  Robot Framework result.

4.5.3 Method: get_tml_logs

Searching for all TML log file from given tml_logpath. Then parse them into to Robot result format.

Arguments:

- `recursive`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If True, search recursively tml log files.

Returns:

/ *Type*: Result object /
Robot Framework result which is converted from TML result(s).

4.5.4 Method: get_robot_log

Get Robot log result from existing output.

Arguments:

(*no arguments*)

Returns:

- `robot_result`
  / *Type*: Result object /
  Robot Framework result from existing robot log file.

4.5.5 Method: convert_to_robot

Convert TML log results to Robot Framework result.

Arguments:

- `tml_results`
  / *Condition*: required / *Type*: str /
  TML result file(s) for conversion.

Returns:

- `robot_result`
  / *Type*: Result object /
  Robot Framework result after converting.

4.5.6 Method: merge_results

Merge convert TML results to exisitng Robot result.

Arguments:

(*no arguments*)

Returns:

- `robot_result`
  / *Type*: Result object /
  Robot Framework result after merging.

4.5.7 Method: save_result_file

Save Robot resuls as output_file file.

Arguments:

- `robot_result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework result for saving.
- `output_file`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Output file name to save the Robot Framework result.

Returns:

(*no returns*)

4.5.8 Method: find_suite

Find Robot suite by name for appending the convert results.

Arguments:

- `root_suite`
  / *Condition*: required / *Type*: TestSuite object /
  Robot Framework root suite object.
- `suite_name`
  / *Condition*: required / *Type*: str /
  Suite name to find in given Robot Framework suite.

Returns:

/ *Type*: TestSuite object /
Robot Framework suite which matchs the given suite name. None is returned in cases there is no matched suite name.

Chapter 5

xmlelementhandlers.py

5.1 Class: XmlElementHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import XmlElementHandler
```

XMl handler for TML execution result.

5.1.1 Method: start

Start handler for parsing XML element.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:
(*no returns*)

5.1.2 Method: end

End handler for parsing XML element.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:
(*no returns*)

5.2 Class: _Handler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import _Handler
```

Base XMl handler class.

5.2.1 Method: get_child_handler

Base method to get child handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

/ *Type*: _Handler object /
Child handler.

5.2.2 Method: start

Base start handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

- `result`
  / *Type*: Result object /
  Robot Framework Result object after processing start handler.

5.2.3 Method: end

Base end handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

(*no returns*)

5.2.4 Method: to_timestamp

Convert to Robot Framework timestamp.

Arguments:

- `stime`
  / *Condition*: required / *Type*: str /
  Time to be converted.

Returns:

- `timestamp`
  / *Type*: str /
  Robot Framework timestamp.

5.2.5 Method: filename

Get file name from full path file.

Arguments:

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

Returns:

/ *Type*: str /
File name from path.

5.3 Class: UnknownHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import UnknownHandler
```

Handler class for unknown element.

5.3.1 Method: get_child_handler

Return itself as child handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

- `self`
  / *Type*: UnknownHandler object /
  Itself.

5.4 Class: RootHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import RootHandler
```

Handler class for root element.

5.5 Class: TMLHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TMLHandler
```

Handler class for autotest element.

5.5.1 Method: start

Start handler to proccess root suite information.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

/ *Type*: TestSuite object /
Robot Framework TestSuite object.

5.6 Class: MetadataSet

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import MetadataSet
```

Base class for metadata information.

5.6.1 Method: end

End handler to store metadata information to Result object.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

(*no returns*)

5.7 Class: MetadataHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import MetadataHandler
```

Handler class for metadata element.

5.8 Class: JenkinsurlHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import JenkinsurlHandler
```

Handler class for jenkinsurl element.

5.9 Class: VersionsHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import VersionsHandler
```

Handler class for versions element.

5.10 Class: VersionSWHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import VersionSWHandler
```

Handler class for version_sw element.

5.11 Class: VersionHWHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import VersionHWHandler
```

Handler class for version_hw element.

5.12 Class: VersionTestHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import VersionTestHandler
```

Handler class for version_test element.

5.13 Class: VersionTMLHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import VersionTMLHandler
```

Handler class for version_tml element.

5.14 Class: TagsHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TagsHandler
```

Handler class for tags element.

5.15 Class: CategoryHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import CategoryHandler
```

Handler class for category element.

5.16 Class: QualitygateHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import QualitygateHandler
```

Handler class for qualitygate element.

5.17 Class: FileHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import FileHandler
```

Handler class for file element.

5.17.1 Method: start

Start handler to create TestSuite object.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

/ *Type*: TestSuite object /
Robot Framework TestSuite object.

5.18 Class: HeaderHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import HeaderHandler
```

Handler class for header element.

5.19 Class: TeststartedHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TeststartedHandler
```

Handler class for teststarted element.

5.19.1 Method: end

End handler to store starttime information to Result object.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.20 Class: TesttoolconfigurationHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TesttoolconfigurationHandler
```

Handler class for testtoolconfiguration element.

5.21 Class: TesttoolnameHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TesttoolnameHandler
```

Handler class for testtoolname element.

5.21.1 Method: end

End handler to store Testtool information as metadata.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.22 Class: TesttoolversionHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TesttoolversionHandler
```

Handler class for testtoolversionstring element.

5.22.1 Method: end

End handler to store testtoolversionstring information as Testtool metadata.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.23 Class: ProjectHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import ProjectHandler
```

Handler class for confname element.

5.23.1 Method: end

End handler to store confname information as Project metadata.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.24 Class: TestfileheaderHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestfileheaderHandler
```

Handler class for testfileheader element.

5.25 Class: AuthorHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import AuthorHandler
```

Handler class for author element.

5.26 Class: DescriptionHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import DescriptionHandler
```

Handler class for shortdescription element.

5.26.1 Method: end

End handler to store shortdescription information as document of Result object.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.27 Class: TestexecutionHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestexecutionHandler
```

Handler class for testexecution element.

5.28 Class: TesterHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TesterHandler
```

Handler class for useraccount element.

5.28.1 Method: end

End handler to store useraccount information as Tester metadata.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.29 Class: TestmachineHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestmachineHandler
```

Handler class for computername element.

5.29.1 Method: end

End handler to store computername information as Machine metadata.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:
(*no returns*)

5.30 Class: TestrequirementsHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestrequirementsHandler
```

Handler class for testrequirements element.

5.30.1 Method: get_child_handler

Return itself as child handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

- `self`
  / *Type*: TestrequirementsHandler object /
  Itself.

5.31 Class: TestbenchconfigHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestbenchconfigHandler
```

Handler class for testbenchconfig element.

5.31.1 Method: get_child_handler

Return itself as child handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

- `self`
  / *Type*: TestbenchconfigHandler object /
  Itself.

5.32 Class: PreprocessorparamsHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import PreprocessorparamsHandler
```

Handler class for preprocessorparams element.

5.32.1 Method: get_child_handler

Return itself as child handler.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

- `self`
  / *Type*: PreprocessorparamsHandler object /
  Itself.

5.33 Class: TestresultsHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestresultsHandler
```

Handler class for testresults element.

5.34 Class: TestHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestHandler
```

Handler class for test element.

5.34.1 Method: start

Start handler to create TestCase object.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

/ *Type*: TestCase object /
Robot Framework TestCase object.

5.34.2 Method: end

End handler to store information from test element.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

(*no returns*)

5.35 Class: ResultHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import ResultHandler
```

Handler class for result element.

5.36 Class: MainHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import MainHandler
```

Handler class for main element.

5.36.1 Method: end

End handler to store main result as status of Result object.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

(*no returns*)

5.37 Class: StateHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import StateHandler
```

Handler class for state element.

5.37.1 Method: get_child_handler

Return itself as child handler.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

- `self`
  / *Type*: StateHandler object /
  Itself.

5.38 Class: ReturnHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import ReturnHandler
```

Handler class for return element.

5.38.1 Method: get_child_handler

Return itself as child handler.
Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.

Returns:

- `self`
  / *Type*: ReturnHandler object /
  Itself.

5.39 Class: LastlogHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import LastlogHandler
```

Handler class for lastlog element.

5.39.1 Method: end

End handler to store lastlog information as message of Result object.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

(*no returns*)

5.40 Class: FooterHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import FooterHandler
```

Handler class for footer element.

5.41 Class: TestdoneHandler

Imported by:

```
from TMLLog2RobotLog.xmlelementhandlers import TestdoneHandler
```

Handler class for testdone element.

5.41.1 Method: end

End handler to store endtime information to Result object.

Arguments:

- `elem`
  / *Condition*: required / *Type*: ElementTree object /
  XML ElementTree object.
- `result`
  / *Condition*: required / *Type*: Result object /
  Robot Framework Result object.

Returns:

(*no returns*)

Chapter 6

Appendix

About this package:

Table 6.1: Package setup

Setup parameter	Value
Name	TMLLog2RobotLog
Version	1.2.0
Date	29.08.2022
Description	Convert TML-Framework results to Robot Framework results
Package URL	robotframework-tmllog2robotlog
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 7

History

0.1.0	31.05.2021
Initial version	
1.1.0	17.03.2022
Initial documentation for package	
1.2.0	29.08.2022
Rework repository's document bases on GenPackageDoc	

TMLLog2RobotLog.pdf
Created at 13.12.2022 - 16:29:29
by GenPackageDoc v. 0.38.0

4.9 RobotLog2DB

RobotLog2DB

v. 1.2.4

Tran Duy Ngoan

18.11.2022

Contents

Chapter 1

Introduction

RobotLog2DB tool helps to import robot *output.xml* result file(s) to TestResultWebApp's database for presenting an overview about the execution and detail of each test result.

In order to display the Robot Framework 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.

However, you can also provide required information as arguments when executing **RobotLog2DB** tool.

Chapter 2

Description

2.1 Robot Framework Testcase Settings:

Hint

The below document is common for Robot Framework Testcase Settings before execution. So that, the generated *.xml result file will contains all required information for importing to TestResultWebApp's database.

However, we suppose to use RobotFramework AIO with RobotFramework_Testsuites library for executing Robot testcase(s).
It will help to define the below `Metadata` information implicitly within the `Suite Setup` bases on your environment and configuration in *.json file:

- `project`
- `version_sw`
- `version_hw`
- `version_test`
- `machine`
- `tester`
- `testtool`

So that, you do not need to define these `Metadata` in your Robot test case.

For the whole test execution:

- Project/Variant (can be overwritten by argument `--variant` or `--config` of **RobotLog2DB** tool when importing):

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

- Versions (can be overwritten by argument `--versions` or `--config` of **RobotLog2DB** 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 **RobotLog2DB** 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}
```

2.2 Sample Robot Framework Testcase:

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

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

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

```
*** 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 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

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

Listing 2.1: Sample Robot Framework testcase

Hint

Above highlighted `Metadata` definitions are not required when using RobotFramework AIO. RobotFramework_Testsuites library will handle these definitions within `Suite Setup` .

2.3 Tool features:

2.3.1 Usage:

Please refer to Usage section of package's repository or try with 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 of the RobotLog2DB importer.
-recursive           if set, then the path is searched recursively for output
                    files to be imported.
-dryrun              if set, then just 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.
```

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

```
RobotLog2DB resultxmlfile server user password database
```

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

```
python -m RobotLog2DB resultxmlfile server user password database
```

2.3.2 Handle missing information:

Default values:

TestResultWebApp requires `Project` , `version_sw` to manage the execution results and `component` group test cases in the displayed charts.

In case above information is missing in testcase settings during the test case execution, that leads to the missing information in the *output.xml* result file. So, these missing information will be set to default value 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 missing information with optional arguments:

But, you can also provide the missing information as command arguments when executing the **RobotLog2DB** 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`
  to provide a configuration **.json* file as `CONFIG` argument. Currently, the configuration **.json* supports 3 types of settings:
  - `"version_sw"` to specify the **Software version** information as string value.
  - `"variant"` to specify the **Project/Variant** as string value.

    Warning:

    These above settings will have lower priority than the commandline arguments `--variant VARIANT` and `--versions VERSIONS`
  - `"component"` to specify the **Component** information which will be displayed on TestResultWebApp. Value can be:
    * A string to apply a single **Component** or all test cases within the execution result. E.g:

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

    * A json object to define the mapping between testcase folders and **Components**. E.g:

      ```
        {
          "component" : {
            "cli"        : "robot/cli",
            "core"       : "robot/core",
            "external"   : "robot/external",
            "keywords"   : "robot/keywords",
            "libdoc"     : "robot/libdoc",
            ...
      ```

```
    },
    ...
}
```

The error will be occurred when the provided configuration **.json* schema is not correct.

Sample configuration json file:

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

2.3.3 Append to existing execution result:

In case you want to append new test result(s) into an existing execution result (is identified by the **UUID**) in TestResultWebApp's database, the combincation of optional arguments `-UUID <UUID>` and `-append` will help to do it.

For example, the result with UUID **c7991c07-4de2-4d65-8568-00c5556c82aa** is already existing in TestResultWebApp's database and you want to append new result(s) from **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 without `-UUID <UUID>` , only a warning message will be showed as below:

```
WARN: '-append' argument should be used in combination with '-UUID UUID` argument.
```

2.4 Display on WebApp:

When the *output.xml* file(s) is importing sucessfully 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:

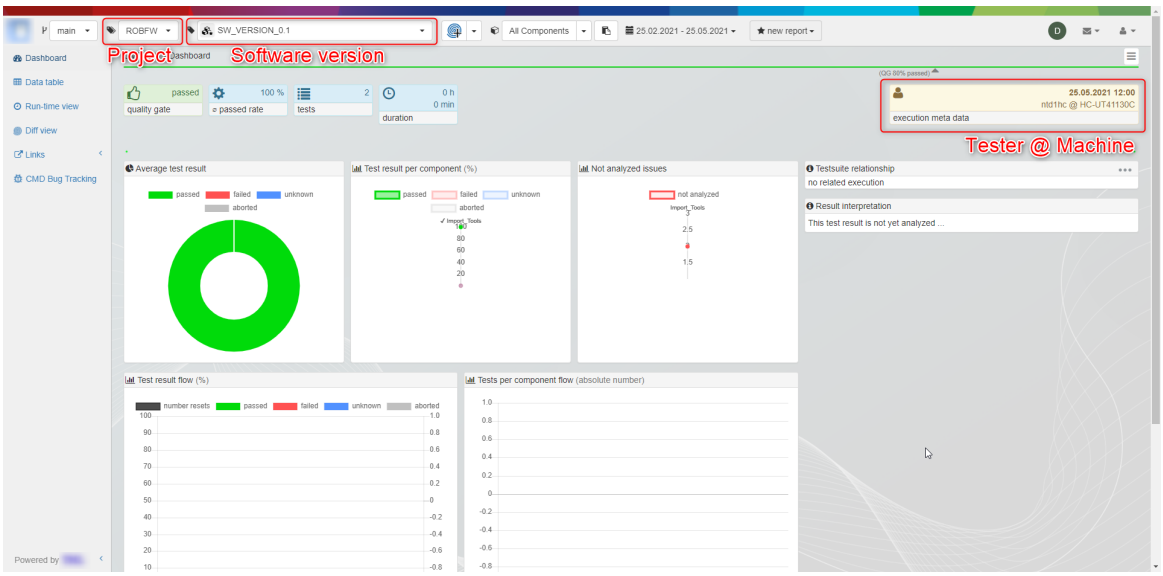


Figure 2.1: Dashboard view

Suite/File metadata and Testcase information:

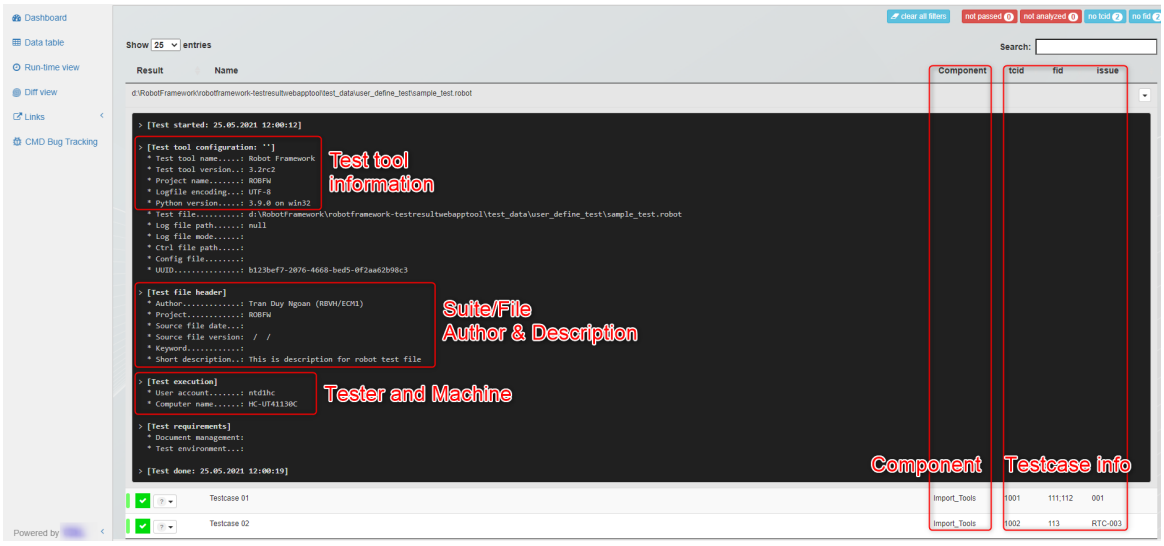


Figure 2.2: Datatable view

Chapter 3

CDataBase.py

3.1 Class: CDataBase

Imported by:

```
from RobotLog2DB.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*: int /
  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*: int /
  Result UUID to be verified.

Returns:

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

Chapter 4

robotlog2db.py

4.1 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.

4.2 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)

4.3 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.

4.4 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`.

4.5 Function: process_suite_metadata

Try to find metadata information from all suite levels.

Metadata at top suite level has a highest priority.

Arguments:

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

Returns:

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

4.6 Function: process_metadata

Extract metadata from suite result bases on DEFAULT_METADATA.

Arguments:

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

Returns:

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

4.7 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*)

4.8 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*)

4.9 Function: process_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 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.

4.10 Function: validate_config

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,
}
```

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.

4.11 Function: normalize_path

Normalize path file.
Arguments:

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

4.12 Function: truncate_string

Truncate input string before importing to database.

Arguments:

- `sString`
  / *Condition*: required / *Type*: str /
  Input string for truncation.
- `iMaxLength`
  / *Condition*: required / *Type*: int /
  Max length of string to be allowed.
- `sEndChars`
  / *Condition*: optional / *Type*: str / *Default*: '...' /
  End characters which are added to end of truncated string.

Returns:

- `content`
  / *Type*: str /
  String after truncation.

4.13 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. Connect to database.
3. Parse Robot results.
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 just check the RQM authentication 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*)

4.14 Class: Logger

Imported by:

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

Logger class for logging message.

4.14.1 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*)

4.14.2 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*)

4.14.3 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*)

4.14.4 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	RobotLog2DB
Version	1.2.4
Date	18.11.2022
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 6

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	

RobotLog2DB.pdf
Created at 13.12.2022 - 16:29:36
by GenPackageDoc v. 0.38.0

4.10 PyTestLog2DB

PyTestLog2DB

v. 0.1.1

Tran Duy Ngoan

22.11.2022

Contents

Chapter 1

Introduction

TODO

Chapter 2

Description

2.1 Get the pytest XML result

In order to import the execution result(s), the ***.xml** file which contains result of all executed pytest testcases is required.

But that file is not generated by default. The argument --junit-xml=<log> needs to be specified when executing the pytest to get the generated JUnit XML result file at given path.

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 usage should be showed as below: :

```
usage: PyTestLog2DB (PyTestXMLReport to TestResultWebApp importer) [-h] [-v]
                [--recursive] [--dryrun] [--append] [--UUID UUID]
                [--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 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.
--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 ***.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)

TODO

2.2.4 Handle missing information

The ***.xml** report file which is generated by PyTest contains only the testcase result(s) and less metadata information about the test execution such as *project/variant*, *software version*, *tester* , *component*, ... which are required by TestResultWebApp.

So that, PyTestLog2DB will handle those information with the default values as below:

- *project/variant* : **PyTest**
- *version_sw* : execution time as *%Y%m%d_%H%M%S* format. E.g **20221128_143547**
- *version_hw* : **""**
- *version_test* : **""**
- *component* : **unknown**
- *testtool* : current python and pytest version. E.g **PyTest 6.2.5 (Python 3.9.0)**
- *tester* : current user.

However, those information can be specified in the configuration json file with option argument --config CONFIG when executing import command.

Required type for those information is **string** except the *component*. Type of *component* info can be:

- **string**: to specify the same *component* for all testcase within this execution.

- **object**: to specify the mapping between *component* info and *classname* of testcase.

Sample configuration json file: :

```
{
   "variant"      : "MyProject",
   "version_sw"   : "0.1.1",
   "component"    : {
      "Testsuite1"      : "test-data.test_tsclass.TestSuite1",
      "Testsuite2"      : "test-data.test_tsclass.TestSuite2",
      "Others"          : [
         "test-data.test_ts1",
         "test-data.test_ts2"
      ]
   },
   "tester"       : "Tran Duy Ngoan"
}
```

As above sample configuration, the component mapping can be explained as below:

- Testcase(s) with classname **test-data.test_tsclass.TestSuite1** is belong to component **Testsuite1**
- Testcase(s) with classname **test-data.test_tsclass.TestSuite2** is belong to component **Testsuite2**
- And component **Others** contains all testcases with classnames **test-data.test_ts1** and **test-data.test_ts2**.

With this feature, the importing execution result can be displayed properly without missing any required information.

2.2.5 Append mode

TODO

2.3 Display on TestResultWebApp

TODO

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*: int /
  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*: int /
  Result UUID to be verified.

Returns:

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

Chapter 4

pytestlog2db.py

4.1 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.

4.2 Function: is_valid_config

Validate the json configuration base on given schema.

Default schema supports below information:

```
CONFIG_SCHEMA = {
    "component" : [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.

4.3 Function: validate_db_str_field

Validate the string value for database field bases on its acceptable length.The error will be thrown and tool terminates if the verification is failed.

Arguments:

- `field`
  / *Condition*: required / *Type*: str /
  Field name in the database.
- `value`
  / *Condition*: required / *Type*: str /
  String value to be verified.

Returns:

/ *Type*: str /
String value if the verification is fine.

4.4 Function: truncate_db_str_field

Truncate input string before importing to database.

Arguments:

- `sString`
  / *Condition*: required / *Type*: str /
  Input string for truncation.
- `iMaxLength`
  / *Condition*: required / *Type*: int /
  Max length of string to be allowed.
- `sEndChars`
  / *Condition*: optional / *Type*: str / *Default*: "..." /
  End characters which are added to end of truncated string.

Returns:

- `content`
  / *Type*: str /
  String after truncation.

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`:

```
{
   "component" : {
      "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.

- `_tbl_test_result_id`
  / *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 just check the RQM authentication 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.
  - 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	0.1.1
Date	22.11.2022
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	

PyTestLog2DB.pdf
Created at 13.12.2022 - 16:29:45
by GenPackageDoc v. 0.38.0

4.11 TestResultWebApp

TestResultWebApp

v. 0.1.3

Thomas Pollerspöck

18.10.2022

Contents

Chapter 1

Introduction

TestResultWebApp is a web-based application which is developed and used at BOSCH since 2016 and was published as open source on Github in 2020.

TestResultWebApp is used for visualizing and tracking test execution results. It provides charts from an overview of the test result to the detail of all included test cases.

It also provides tools to control the quality of developing software version (under testing) by the a graphical comparison of test results from different test executions.

TestResultWebApp is highly modular written. Therefore it is also easy to extend it in order to add new graphics or evaluations of the test result data.

TestResultWebApp uses bootstrap, jquery, nodejs and mysql. For the charts is uses chartjs and D3.

Chapter 2

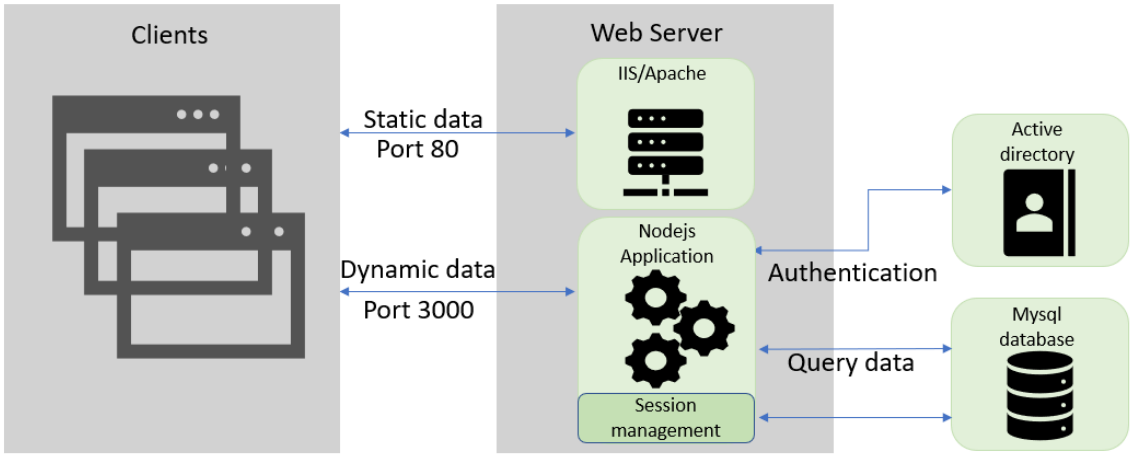
Description

2.1 TestResultWebApp Architecture

The TestResultWebApp application includes:

- Database: MySQL - which contains all test execution results. For details of all tables in the database, please refer the database model.
- Active Directory: for the authentication.
- Web server: which hosts the static data and runs the Node.js application for providing the dynamic data.
- Web client: is written in javascript which also use some libraries such as jQuery, bootstrap, Chart.js, D3 for data visualization.

Please refer below architecture for more detail:



2.2 Import Data

The data which will be visualized on WebApp comes directly from the database. For this the test execution result data must be transformed first into the defined database model then imported into the database.

The data base model is generic, so test result data can be any. Only a test result importer must transfrom and import the data.

We provide already the RobotResults2DB which helps to import the Robot Framework result file(s) *output.xml* to the database of the TestResultWebApp.

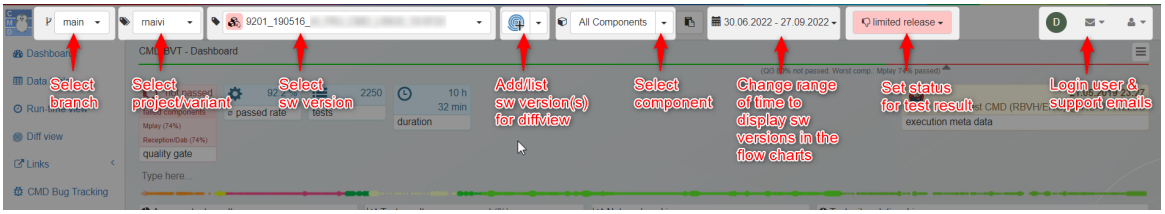
You will need to provide only the Robot Framework result file(s) and database's credential information, that RobotResults2DB will parse the test execution result data and interact with the provided database to import the data.

Please refer RobotResults2DB's usage and its document for more detail.

2.3 Data Visualization

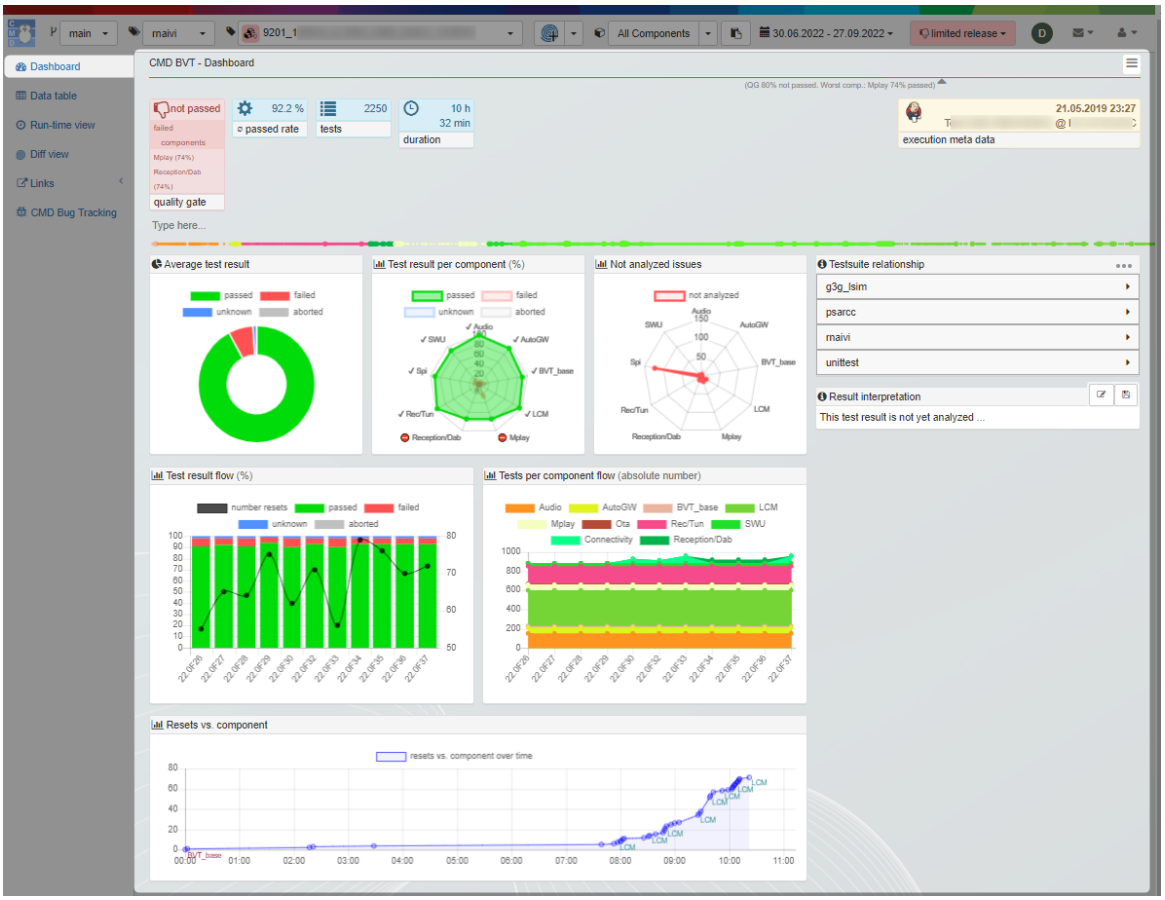
2.3.1 Main menu

The WebApp provides a main menu for selecting a specific **branch**, **project/variant** and **software version** to be displayed.



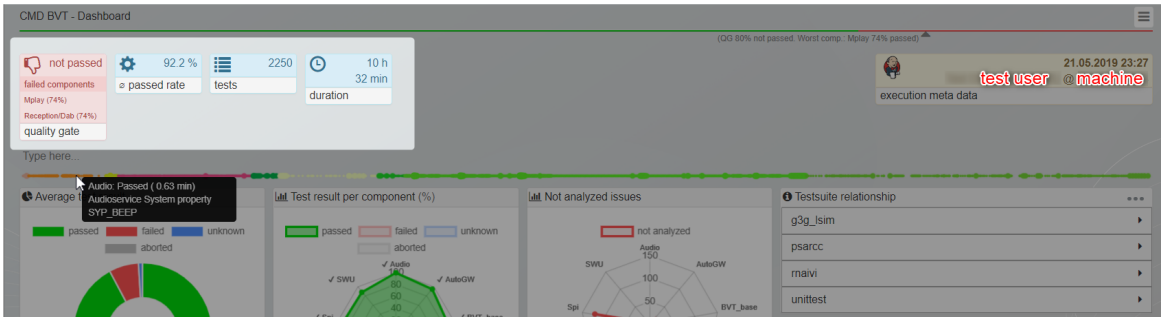
2.3.2 Dashboard View

The Dashboard view does not only provide an overview of test execution results but also shows the correlation betweeen components within the result and relationship with other test execution results.



Result overview

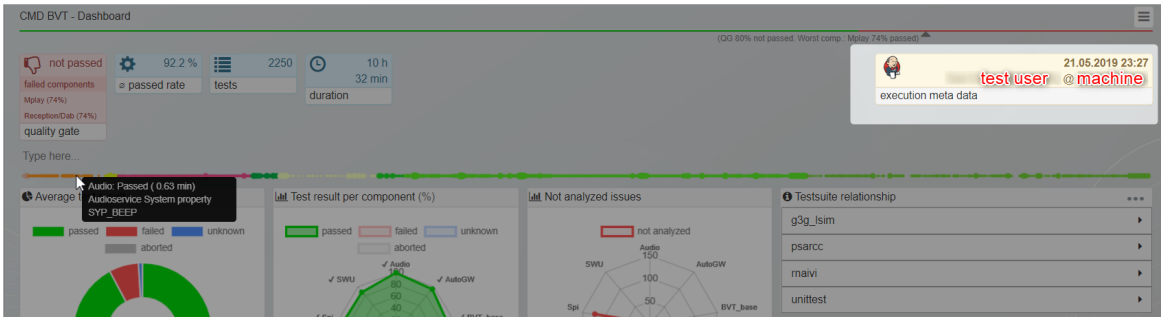
At the top left corner of the Dashboard view you find the test execution result statistics:



Which contains:

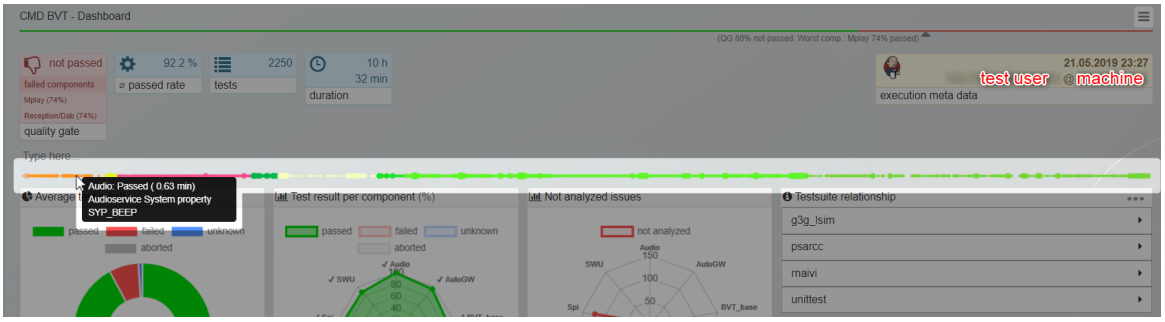
- Overrall status (you can define a qualitiy gate).
- Passed rate.
- Total number of executed test cases.
- Test execution duration.

On other right-hand side, there is information about the execution environment:



- Execution time (start of test execution)
- Test machine
- Test user
- Jenkins link (embedded URL in the Jenkins's icon)

Below them is the test execution result timeline:



It provides:

- The timeline of the executed test cases which are grouped by components (different color). Left side is start of testing. Right side the end.
- How much time is consumed by the individual test case by the distance to the next test at the time line or the detail pop-up when hovering on the dot at the timeline.
- Test status result: A small dot for Passed status and a big dot for others.

Average test result

This chart will give you the detail of the test result with the percentage (number of test cases will be shown when moving the mouse over the pie chart) of each result status (Passed, Failed, Unknown or Aborted) of the execution.

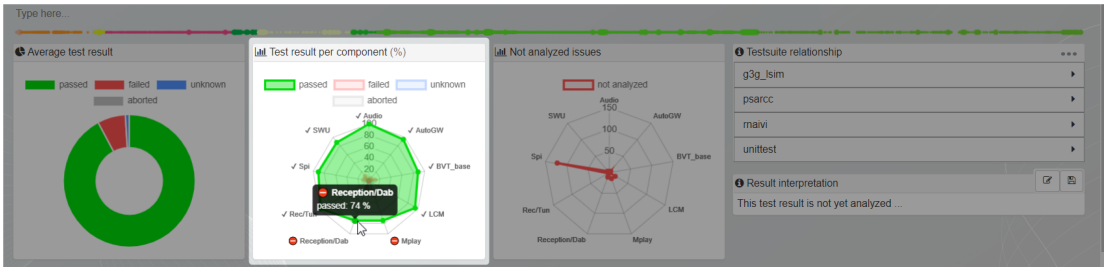
So that, you can qualify this test execution result is good or not.

Component's correlation

The next charts will help you to get the correlation between components within the test result, so that you can know which component(s) impact(s) the test result.

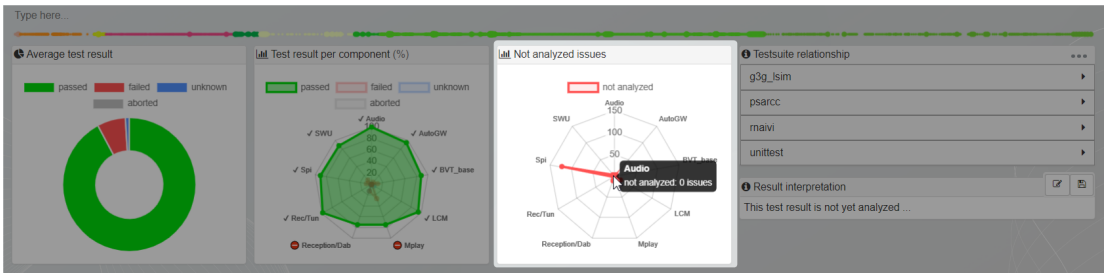
- **Test result per component** chart: provides a fast overview of test result percentages over all components. Here you can quickly see the quality of the system under test. Idially the spider chart should be fully green. This means that all component tests result in 100% Passed.

  You can also define a quality gate (default 80%) which results in a "minus" in front of the component name if the quality gate is not reached.



- **Not analyzed issues** chart: you can known how many test cases of components are issued without analysis.

  The Datatable view provides a process to set Failed test cases to "analyzed". Ideally this spider chart should have a red dot in the center. This means all Failed test cases are analyzed.



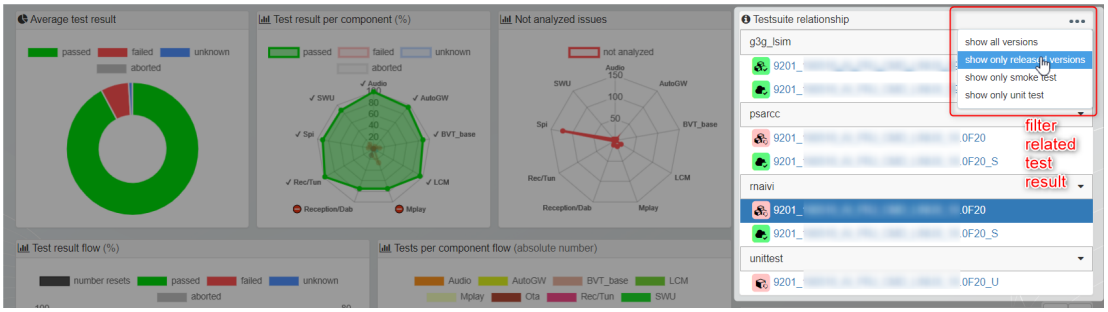
Relationships with other test execution results

- **Testsuite relationship**: will let you know all the related test results (grouped by project/variant) of the current selected version.

  This allows to quickly go to related test results to have a fast comparison about the quality of the selected version across all projects/variants.

For example: the selected version have been executed for 4 projects/variants: *g3g_lsim*, *psarcc*, *rnaivi* and *unittest*. Each variant (except the *unittest*) has 2 test results (one for the smoke test and one for the whole test execution result).

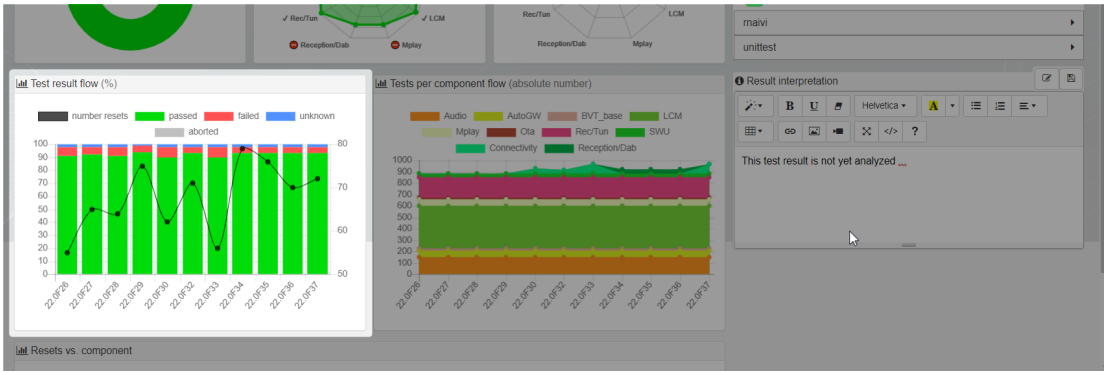
Then, all the related test execution results will be displayed as below:



There is also the context menu (**...**) that allows you to filter the related test results.

- **Test result flow** chart: provides the picture of quality change (percentages of each status) between versions.

  So that, you can understand the quality of testing software is being improved or vice versa.



- **Tests per component flow** chart: provides the change of number test cases per component between versions.

  You can aware the number of test cases per component and how many test cases are added or removed (per component or the whole test result) from those versions.

  This allows to quickly verify if all expected tests are executed, or if expected tests were not executed.

> **Notice**
>
> The versions which are displayed in **Test result flow** and **Tests per component flow** charts are the executed test results within the selected range of time in the main menu (default is **last 90 days**).

Result interpretation

You can give more information, provide an analysis, take notes, ... on the execution result by leaving comments in the **Result interpretation** section.

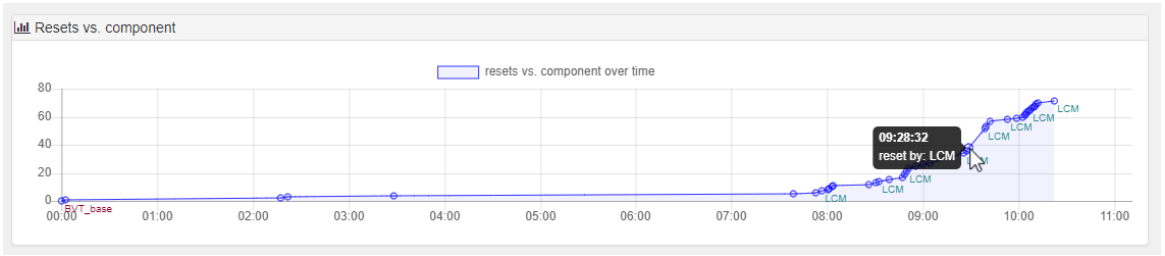
As soon as the Result interpretion is saved, that information will be updated to the database. So that, other users who browse to this test result can see the analysis/comment for reference.

Resets vs. component

The **Resets vs. component** chart will help you to know the interaction of component tests with the DUT (device under test) by providing the reset counter per component during the execution.

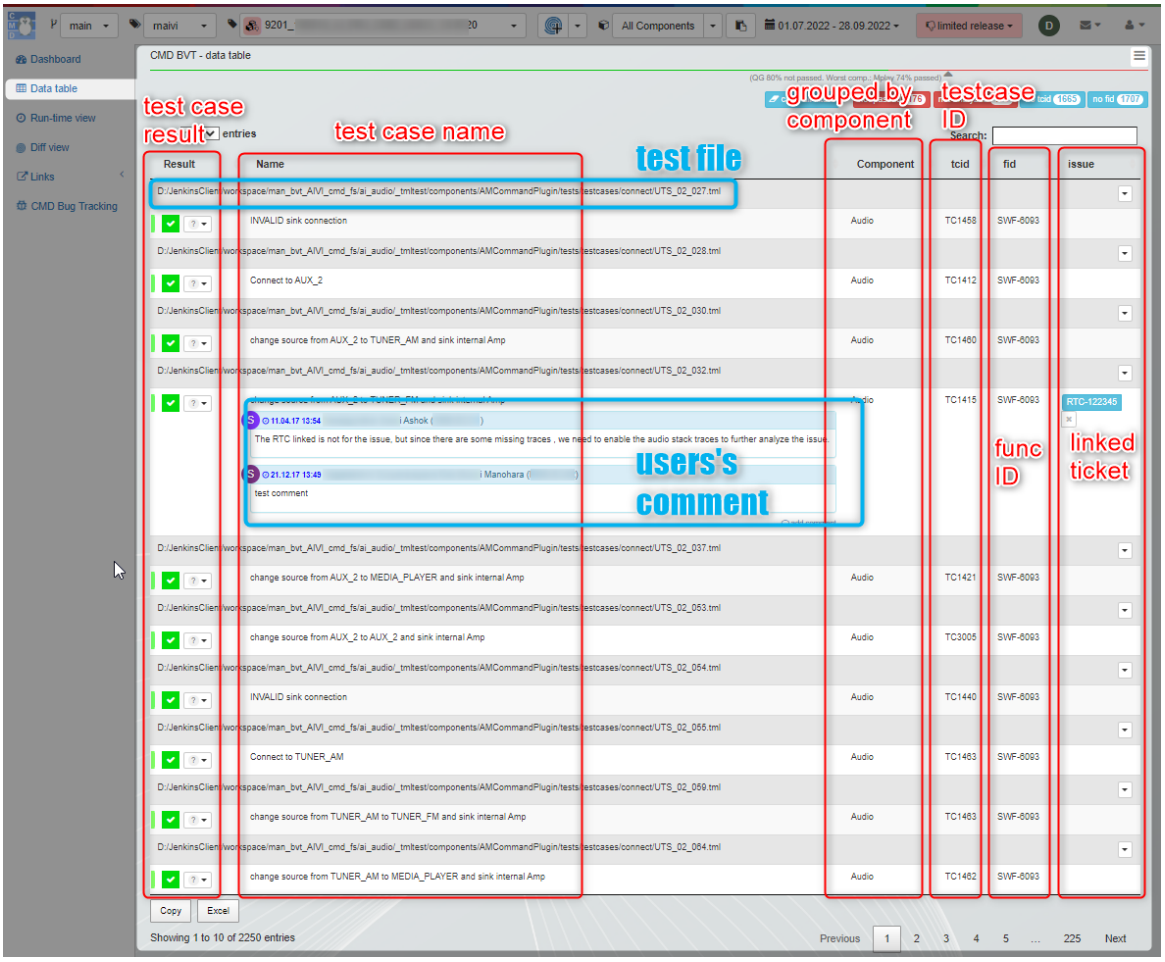
You can have awareness that:

- When the DUT has been reset.
- Which component tests were executed when a reset happened.
- How many reset has been done during each component and the whole test execution.



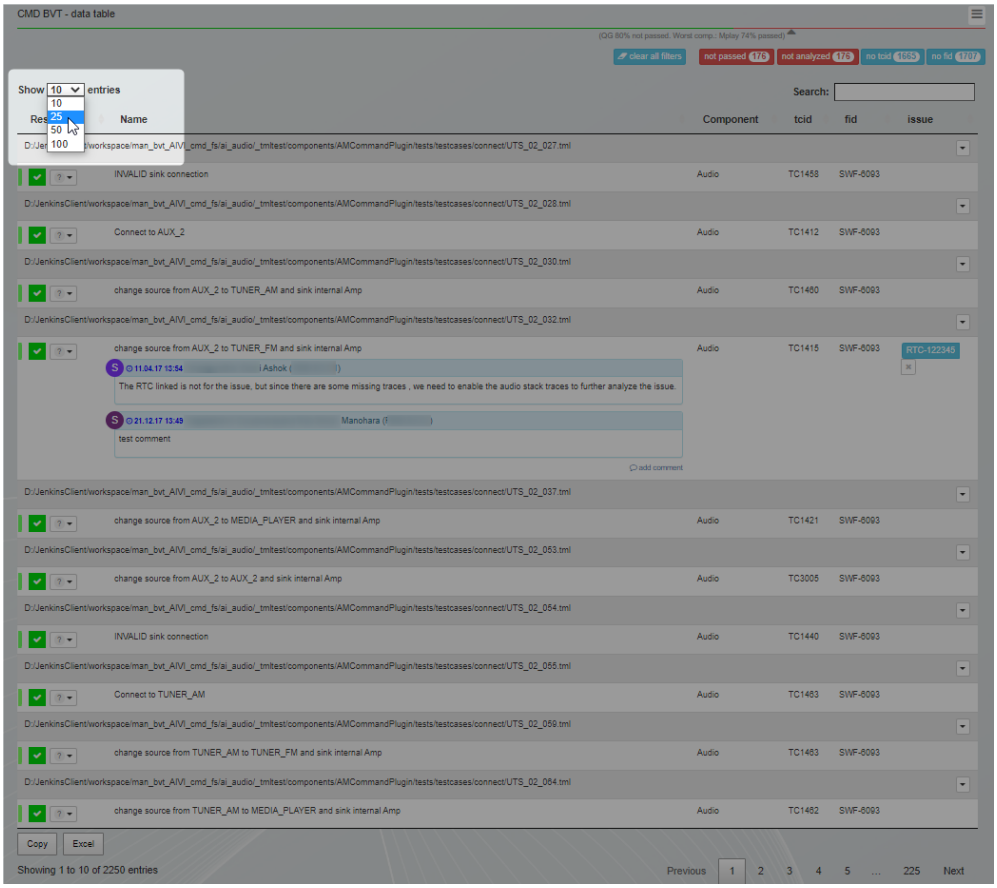
2.3.3 DataTable View

The Datatable view provides the summary table which contains all detail information of each test case (grouped by component) within the test execution.

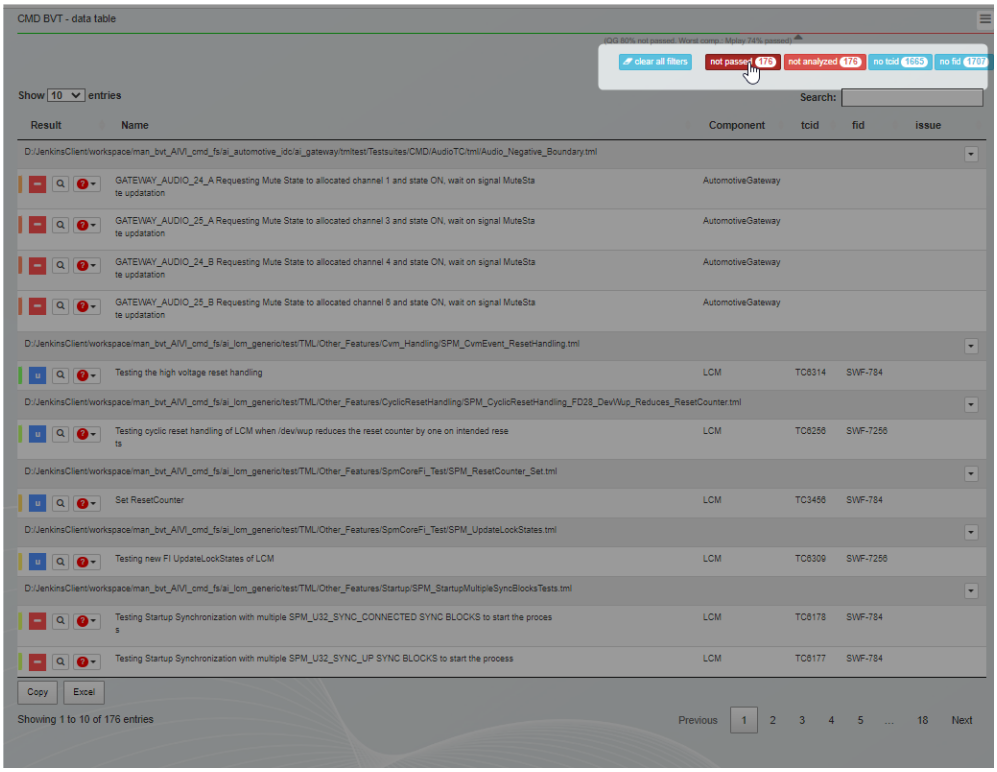


Besides, you can aslo:

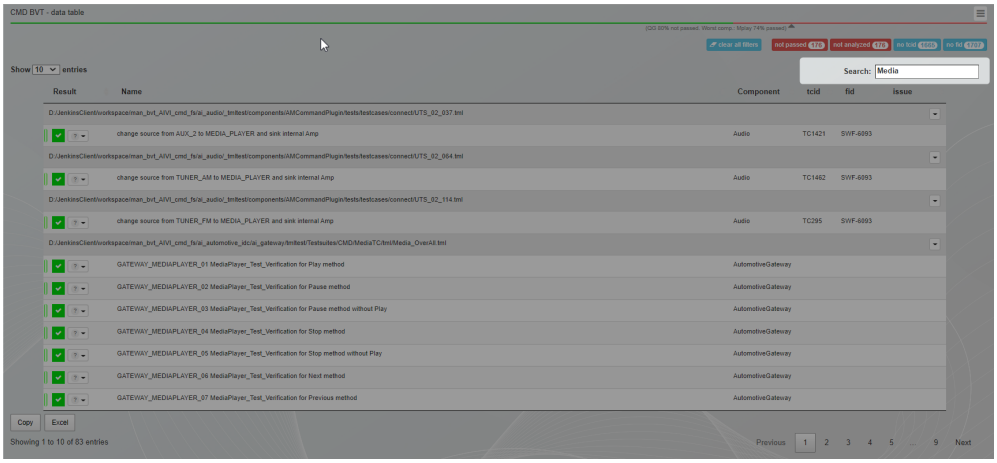
- determine how many test cases (entries) are displayed in the table page.



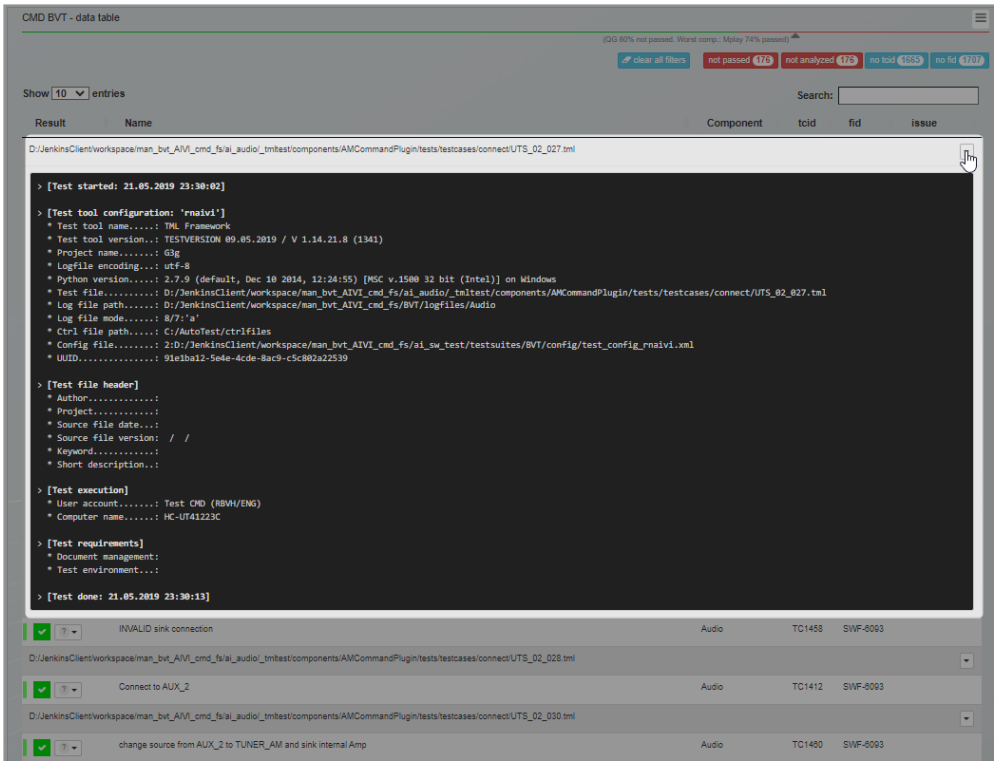
- apply filters to display such as not Passed test cases.



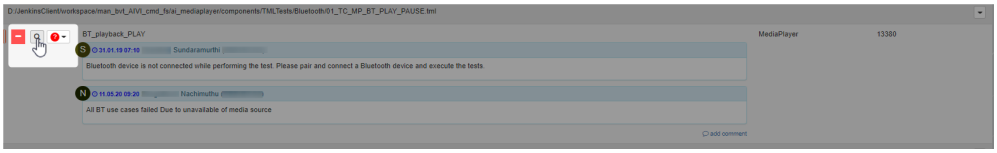
- search for the specific test case for reference.



- get more information about test environment, configurations.

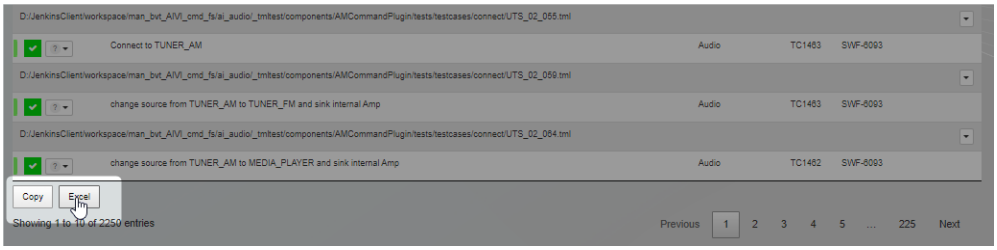


- get traceback information for not passed test case(s). An pop-up will be displayed which show all recorded traceback (error) information.



- give comment to the test case, link an issue ticket with the test case or watch the history of the user interaction on the test case.

- copy data table or export them to an excel file.

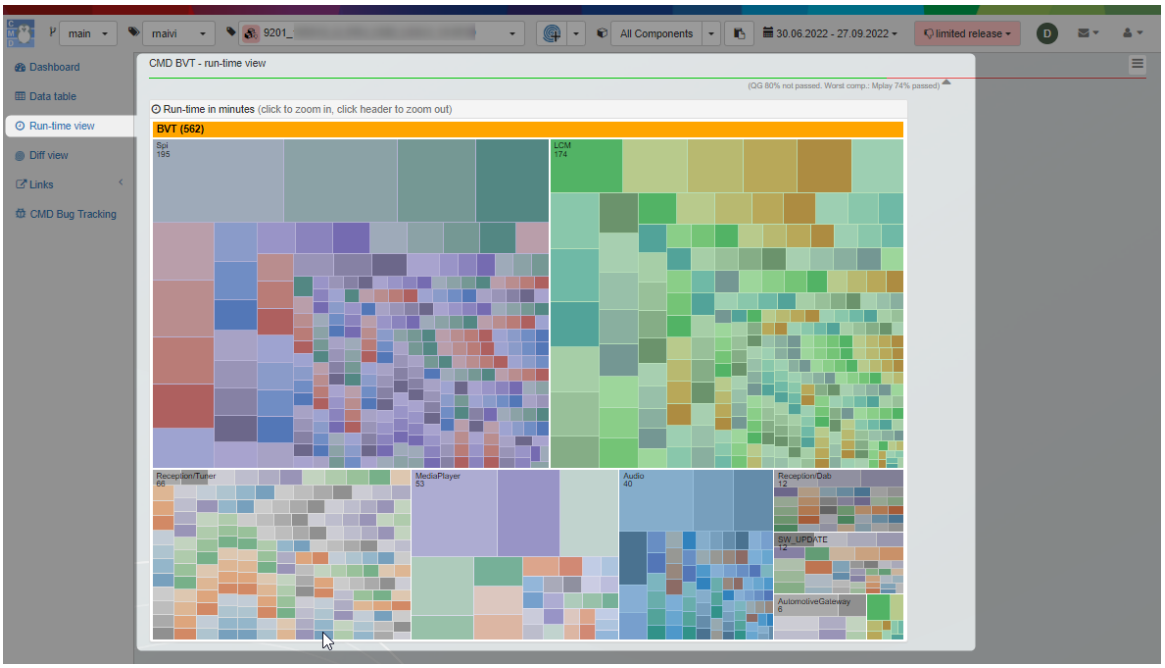


2.3.4 Runtime View

The runtime provides a treemap chart which helps you to know the runtime of components within test execution result or test cases within each component. The displayed number is always the runtime in minutes. The header shows the total runtime.

You can click on the component to go to detail runtime of all test cases within it and go back by clicking the component header.

With this view, you can understand the runtime of a your test suite then optimize the specific component or testcase if required.



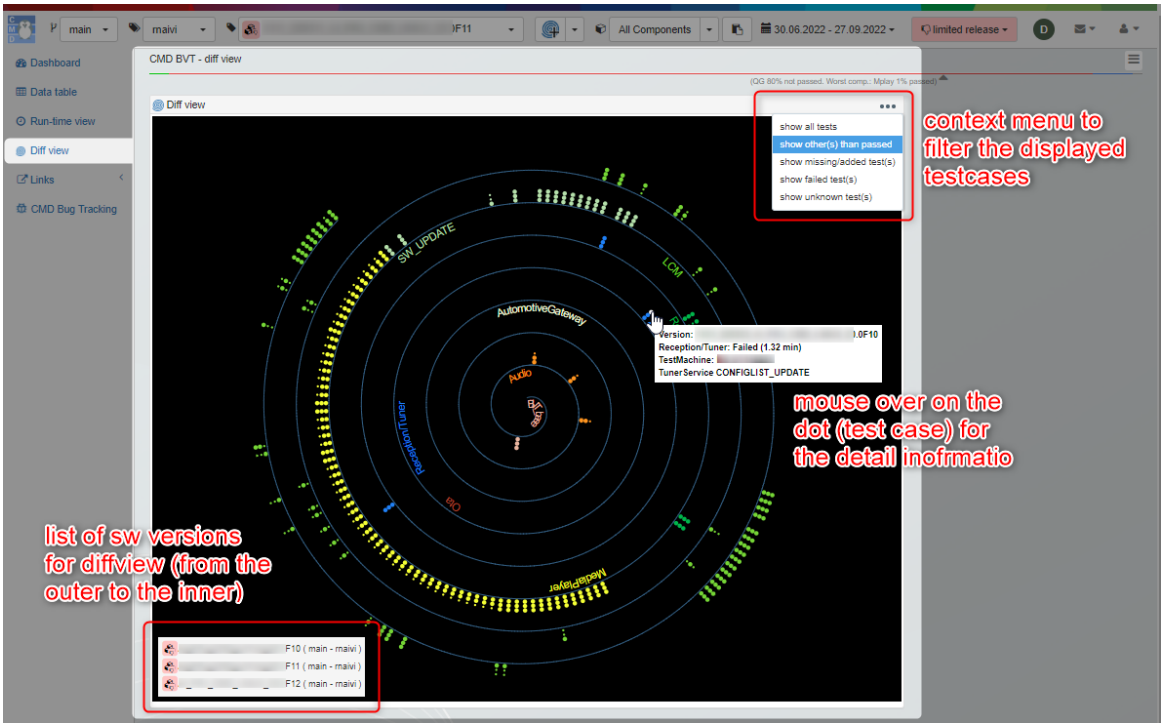
2.3.5 Diff View

The diffview contains a spiral chart which displays the differences between the software versions you want to compare. The center of the spiral is the start time of the test execution, the end of the sprial the end time. You find also the name of the component in the spiral. By default only changed or Failed test results are displayed.

So that you can:

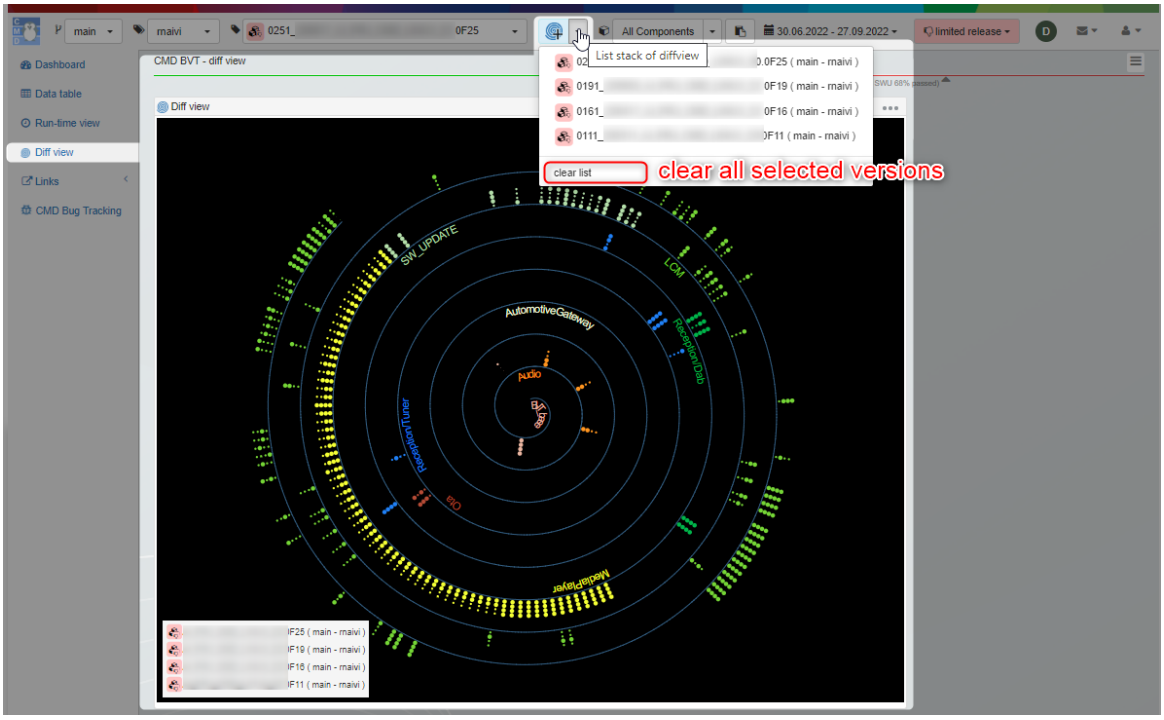
- see a test case result change (e.g from Passed to Failed).
- quickly find new or removed test case(s).
- recognize unstable test case(s)/component(s).

Be default, without adding the version for diffview, the current selected version and its around versions (the previous and the next version if existing) will be chosen for the diffview as below:



In case you want to add specific versions for the diffview, select the version from the version select box then click the add button  to add it to the list of versions to diff.

The dropdown button is used for viewing your selected versions. You can also clear your selection with **clear list** option.



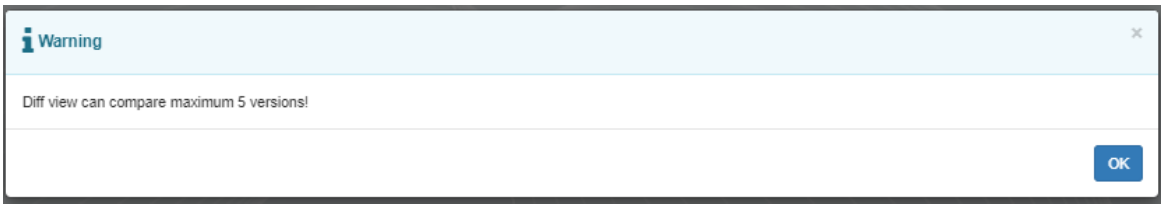
As soon as a new version is added for diffview, the spiral chart is updated immediately. Dependent on the number of executed test cases and browser rendering can need some time.

The dots which present the test cases along the spiral line (small dot is Passed and bigger one for other status) are interactable. It means that you can:

- movethe mouse over the dots to see the test case information.
- click on the failed test case (bigger dot) for the traceback information.

Notice:

You can only select maximum of **5** versions for diffview.
If the maximum of selected versions is reached and you click on the button to add more, a warning message will be displayed to prevent that action.



2.4 Developer guidance

Notice

In order to run up the TestResultWebApp, it requires some knowledge about:

- Web server: setup and run web server for web hosting.
- Nodejs platform and Express framework: adapt the sourcecode with your environemnt: domain, configurations, ...
- Mysql database: schema, tables, SQL scripting. We propose to use MySQL Workbench tool for working with Mysql database.

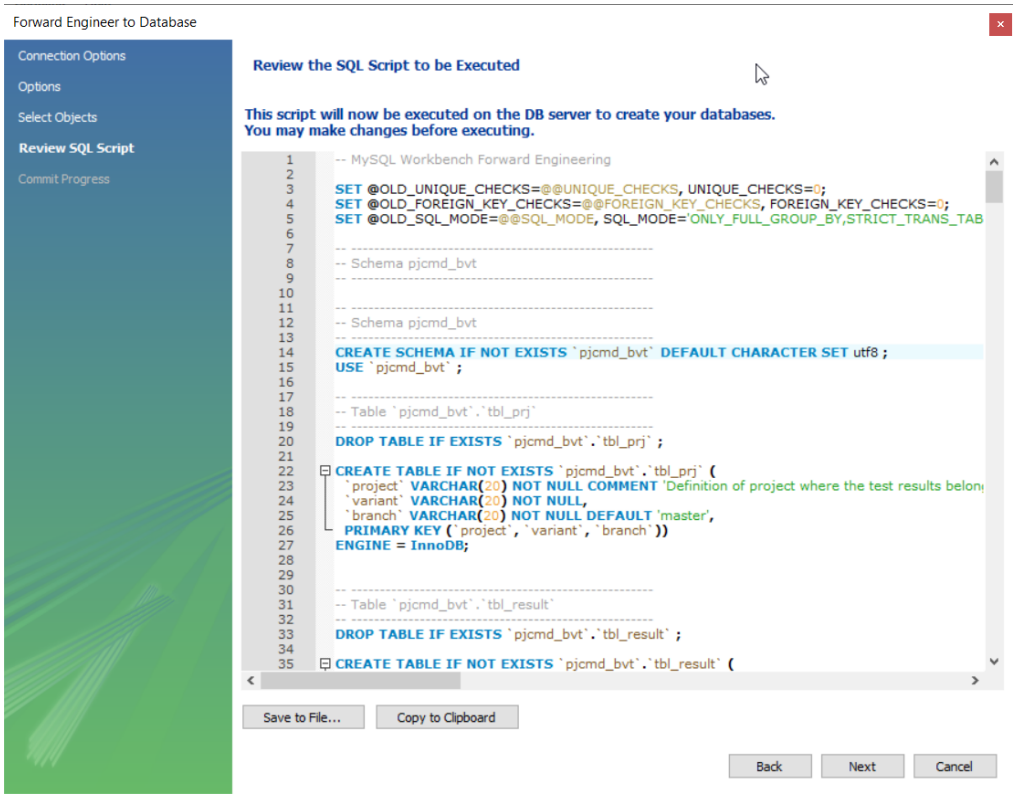
2.4.1 How to run new TestResultWebApp instance

1. Precondition:
   - Node.js shoulde be installed.
   - MySQL server should be installed.
   - A cloned package's resource from Github repo

     ```
     git clone https://github.com/test-fullautomation/testresultwebapp.git
     ```

   - The port which will run Node.js application (default is **3000**) is opened (enable) in firewall.
2. Setup mysql database:
   - create *user*, *password* and *schema* then update them as `global.mySQLOptions` in `webapp/web_server/lib/global.js` file.
   - create required tables in your database schema with MySQL Workbench.
     - open the datamodel which is located at `mysql_server/datamodel/test_result.mwb`
     - export all defined tables in datamodel to your schema, use **Foreward Enginneer to Database** feature of MySQL Workbench.

       ```
       EER Diagram > Database > Forward Engineer... > your schema
       ```



> **Notice**
>
> If you have created your schema with other name than the default name *pjcmd_bvt*, you should replace the schema name at the **Review SQL Script** step:
>
> * copy SQL script to text editor such as VsCode
> * replace *pjcmd_bvt* which your new schema name
> * paste SQL script back to the **Foreward Enginneer to Database** tool
>
> before moving to next step to execute SQL script.

– create all store procedures by loading all SQL scripts under `mysql_server/TMLdb_sproc/` folder then execute them.

> **Notice**
>
> If you have created your schema with other name than the default name *pjcmd_bvt*, you should replace the schema name before executing SQL scripts.

3. Import the initial data to the database with RobotResults2DB tool.
4. Adapt Web server:
   - `web_server/lib/global.js` : for domain, database's configuration, keys for authentication, ldap server for authentication, ...
   - `web_server/test.js` : for listening port of nodejs application.
5. Adapt Web client:
   - `web_client/dashboard/dist/js/common/global.js` : for domain.
   - `web_client/dashboard/dist/js/common/communication.js` : for listening port of Node.js application's API.
6. Start nodejs application by command:

```
node testdb.js
```

7. Start web server for hosting the static files:

   You can use any web server Apache, IIS or Nginx for hosting the static files under `web_client/dashboad/` folder when running as production.

   For development, you can use directly `express.static` which is supported by Express framework for hosting static files. The `web_server/test.js` file should be modified to add this setting.

```
'use strict';

var global = require('./lib/global');
var path = require('path');

...

//for local GUI tests deliver static HTML
//content from port 3000
app.use(express.static(path.join(__dirname, "../web_client/dashboad/")));

var session = require('express-session');
var MySQLStore = require('express-mysql-session')(session);
...
```

8. Now open your favourite browser, go to the domain of webapp and enjoy.

Chapter 3

Appendix

About this package:

Table 3.1: Package setup

Setup parameter	Value
Name	TestResultWebApp
Version	0.1.3
Date	18.10.2022
Description	Web based display of test results
Package URL	testresultwebapp
Author	Thomas Pollerspöck
Email	Thomas.Pollerspoeck@de.bosch.com
Language	Programming Language :: JavaScript
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 4

History

0.1.0	07/2022
Initial version	
0.1.1	09/2022
Update README file and package's document	
0.1.2	05/10/2022
Fix findings with package's document	

TestResultWebApp.pdf
Created at 13.12.2022 - 16:29:52
by GenPackageDoc v. 0.38.0

RobotFrameworkAIO_Reference.pdf

Created at 13.12.2022 - 16:28:14

by genmaindoc v. 0.26.0