

Specification of

RobotFramework AIO

v. 0.11.0.2

Implemented by:

Thomas Pollerspöck

Nguyen Huynh Tri Cuong

Tran Duy Ngoan

Mai Dinh Nam Son

Tran Hoang Nguyen

Holger Queckenstedt

04.2024

Contents

1	Introduction	1
2	Installation and start	2
3	Components	5
4	Environment	8
5	Code analysis	10
6	Logging	12
6.1	Background	12
6.2	Summary	12
7	Threading in Robot Framework	13
7.1	Background	13
7.2	Threading Implementation	13
7.3	New Threading Keywords	13
7.3.1	Send Thread Notification Keyword	13
7.3.2	Wait Thread Notification Keyword	14
7.4	Summary	14
8	Library documentation	15
8.1	GenPackageDoc	17
8.2	PythonExtensionsCollection	41
8.3	RobotframeworkExtensions	81
8.4	JsonPreprocessor	93
8.5	RobotFramework_TestsuitesManagement	123
8.6	QConnectBase	173
8.7	QConnectWinapp	220
8.8	RobotFramework_DBus	240
8.9	RobotLog2RQM	281
8.10	RobotLog2DB	311
8.11	RobotFramework_DoIP	348
8.12	PyTestLog2DB	372
8.13	TestResultWebApp	404
9	Appendix	424
9.1	Installed Python Modules	424

Chapter 1

Introduction

The Robot Framework is an automation framework that can be used to execute automated software tests. It's open source, keyword driven, and extensible. Tests are implemented in simple text files.

The extensibility of the Robot Framework gives every test developer the possibility to add missing functionality easily. But when thinking about missing functionality, some aspects urgently need to be considered:

- Additions should be designed carefully to make them as much generic as possible. This will make the additions useful for a wide range of test developers.
- Additions have to be bundled in a meaningful way, and they have to be set under version control.
- Tests need to be reproduced, and therefore also the entire test system (Robot Framework together with all additions) must be reproducible.

And these deliberations are not only related to the Robot Framework itself. Developing software in an efficient way also requires a development environment. And this development environment most probably needs to be configured - at least to be able to handle the additions also. It has to be ensured to use open source software only, to avoid any license fee. And finally it must be ensured that every test developer within a team of developers uses the same development environment with the same settings.

To cover all aspects mentioned above, the following things have been done:

- Download a certain Python version (because the Robot Framework is a Python based application).
- Add further useful Python modules to the Python installation.
- Download a certain version of the Robot Framework. This is the base.
- Implement (carefully) some changes within the core source code of the Robot Framework.
- Add further components to provide useful features that are currently missing in Robot Framework (like a management of test configuration values and a version control mechanism).
- Select a development environment (here: VSCodium).
- Configure this development environment to handle especially the Robot Framework code together with all additions.
- Put all things together in a separate installer.

The outcome is one single installer, that installs all together: the framework, additional libraries, the development environment and all settings.

This is what we call "**RobotFramework AIO**", with **AIO** means: All In One.

With this solution the entire test setup is reproducible by every developer on every computer.

And there is no need any more for any user, to install all affected components manually, and there is also no need any more for any user, to spend effort on the configuration of the test setup. After using the RobotFramework AIO installer, test developers immediately can start to implement tests.

Chapter 2

Installation and start

The **RobotFramework AIO** can be installed under Windows and under Linux in this way:

- **Win10**

Double-click the `setup_.*.exe` file and follow the instructions.

- **Linux**

If already installed, remove the previous version (this will not remove your own data):

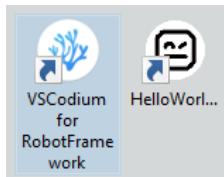
```
sudo apt-get remove robotframework-aio
```

Install/Update now the current version (assumed you are in the download directory):

```
sudo apt-get install ./setup_RobotFramework_AIO OSD6Linux\_* .deb
```

After the installation is finished, a new **VSCodium** icon is present on desktop:

- **Win10**



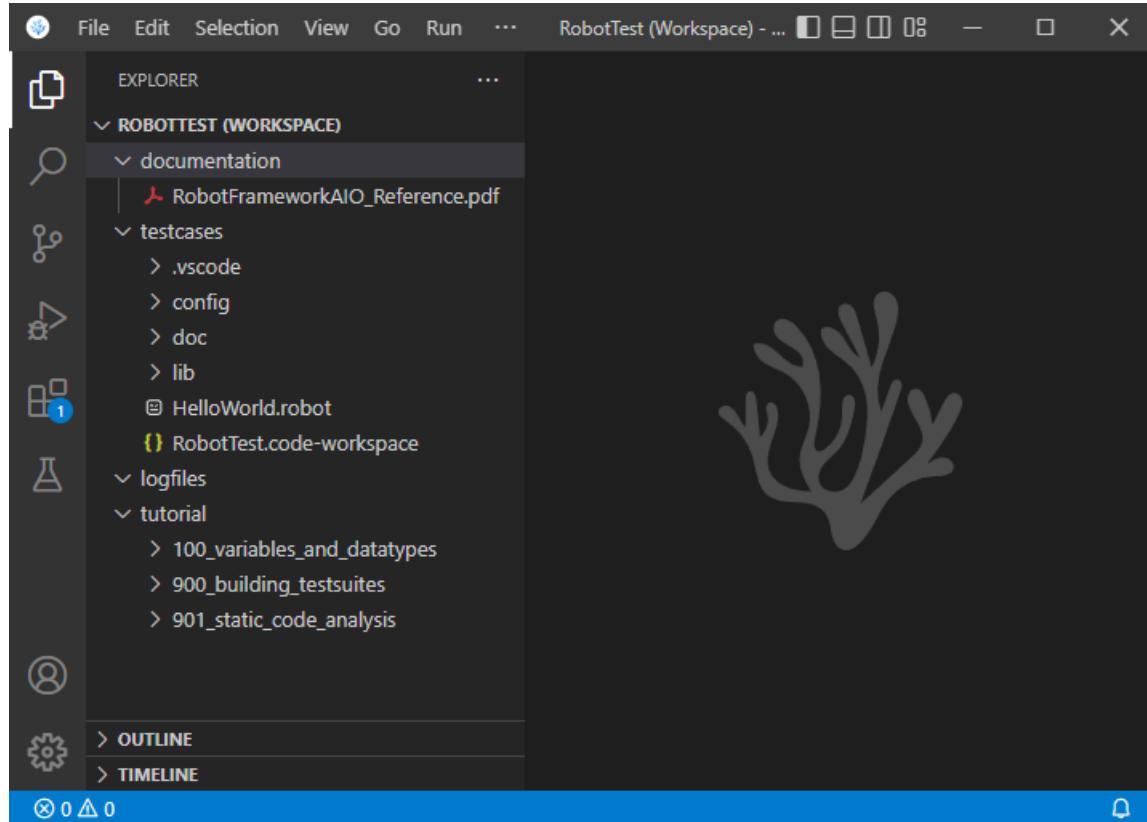
(together with a link to the example robot file `HelloWorld.robot`)

- **Linux**



A double-click starts the application.

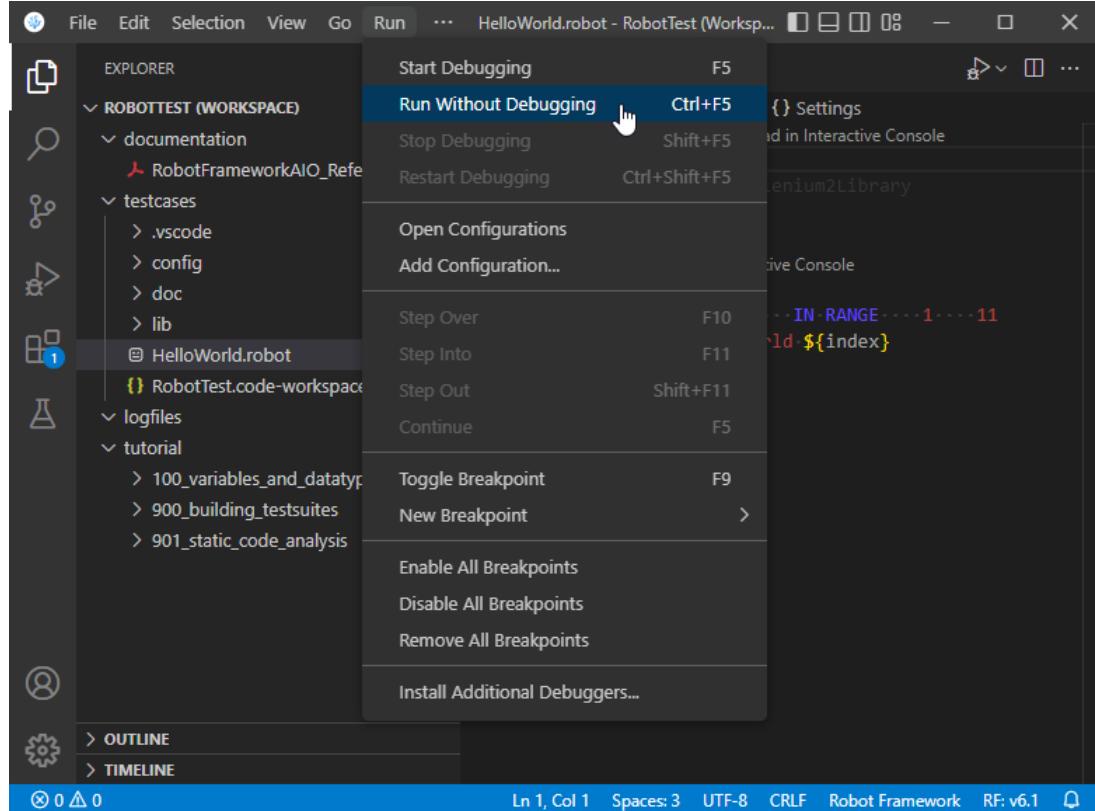
On first start the **VSCodium** main window appears like this:



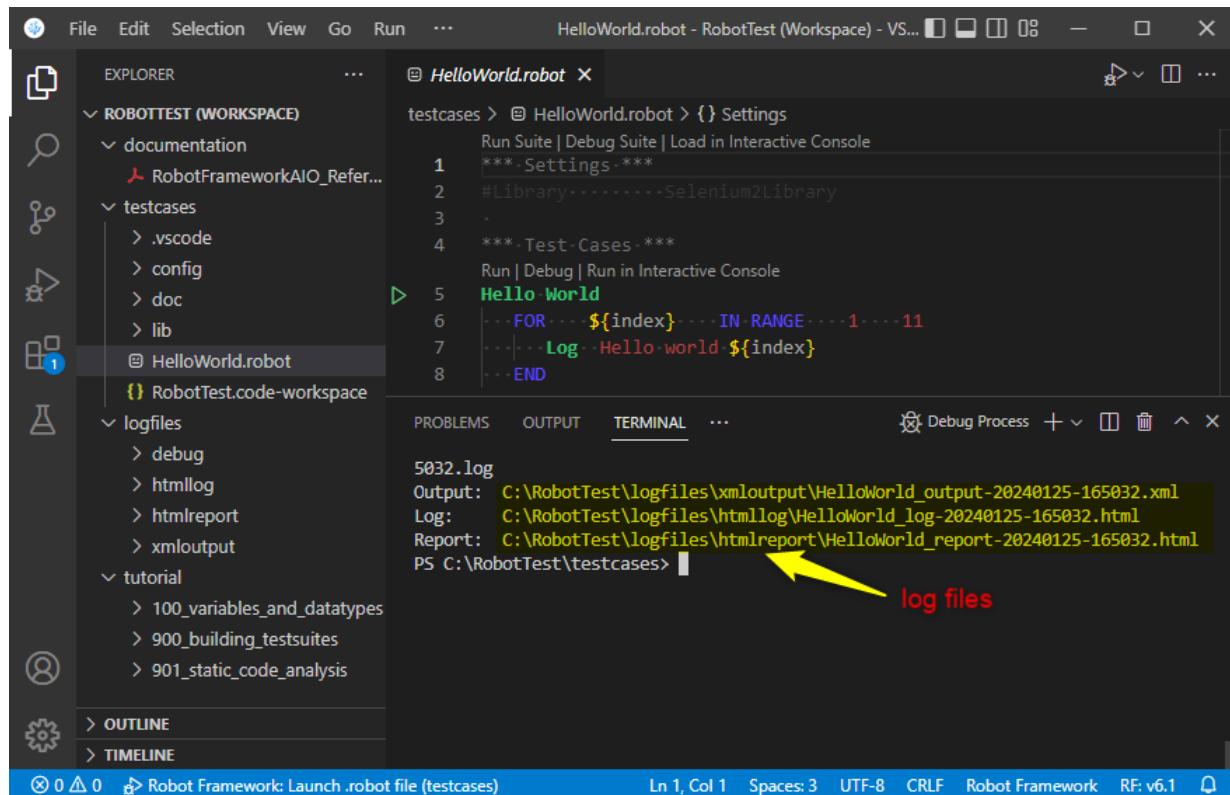
The preconfigured workspace contains the following parts:

- The main documentation `RobotFrameworkAIO_Reference.pdf` (including the documentation of all components that are part of the **RobotFramework AIO**)
- A `testcases` folder containing (beneath other elements) the example robot file `HelloWorld.robot`
- A `logfiles` folder in which all log files and reports will be placed (from tests that are executed with **VSCodium**)
- A `tutorial` folder containing a large bunch of testfiles that can be used for self-dependent learning

For the usage of the predefined infrastructure (e.g. the `logfiles` folder), the **VSCodium** provides an adapted configuration of the **Run** button. This button executes the robot file selected in project explorer of **VSCodium**.



After test execution, the links to log files and reports can be found in the **TERMINAL** window.



Every file there can be opened by CTRL+click on the selected file.

Chapter 3

Components

Compared with the core Robot Framework, the RobotFramework AIO contains some additional components, that extend the functionality by useful features. All components are hosted on GitHub. Most of the components can also be used stand-alone (= outside the context of RobotFramework AIO). But in case of a stand-alone usage, the user is responsible for the installation and configuration. Further hints can be found in the readme files of the corresponding GitHub repository.

The following is an overview about all additional components, that are part of the RobotFramework AIO:

1. RobotFramework_TestsuitesManagement

The **RobotFramework_TestsuitesManagement** enables users to define dynamic configuration values within separate configuration files in JSON format. These configuration values are available during test execution - but under certain conditions that can be defined by the user (e.g. to realize a variant handling or to define parameters that are test bench specific).

The **RobotFramework_TestsuitesManagement** also provides a version control mechanism to ensure that the developed tests fit to the test environment.

Homepage: [RobotFramework_TestsuitesManagement](#)

2. JsonPreprocessor

The **JsonPreprocessor** provides additional features in JSON files. These features extend the standard JSON format and make it easier to use JSON files for e.g. configuring tests (see also **RobotFramework_TestsuitesManagement**). The additional features are:

- Add comments
- Let a JSON file import other JSON files (nested imports; e.g. to avoid redundancy)
- Allow also pure Python keywords like `True`, `False` and `None`
- Define new parameters and overwrite already existing ones

Homepage: [JsonPreprocessor](#)

3. QConnectBase

QConnectBase is a connection testing library for the Robot Framework. It provides a mechanism to handle trace log continuously 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 overflowed response trace log from asynchronous trace systems.

Homepage: [QConnectBase](#)

4. RobotLog2RQM

RobotLog2RQM writes Robot Framework test results (available in XML format) to the IBM® Rational® Quality Manager (RQM). This includes:

- Create all required resources (Test Case Execution Record, Test Case Execution Result, ...) for new testcase on RQM
- Link all testcases to provided testplan

- Add new test result for existing testcase on RQM
- Update existing testcase on RQM

Homepage: [RobotLog2RQM](#)

5. RobotLog2DB

RobotLog2DB writes Robot Framework test results (available in XML format) to a database. This database is provided by another component called **TestResultWebApp**. Based on the content of the database the **TestResultWebApp** presents an overview about the whole test execution and details about each test result.

Homepage: [RobotLog2DB](#)

6. PyTestLog2DB

PyTestLog2DB writes test results of the Python module pytest (available in XML format) to a database. This database is provided by another component called **TestResultWebApp**. Based on the content of the database the **TestResultWebApp** presents an overview about the whole test execution and details about each test result.

Homepage: [PyTestLog2DB](#)

7. TestResultWebApp

TestResultWebApp is a web-based application and 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

Homepage: [TestResultWebApp](#)

8. GenPackageDoc

GenPackageDoc provides a toolchain to generate a documentation in PDF format out of Python sources that are stored within a repository. 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 RST format or in LaTeX format. **GenPackageDoc** is also designed to consider setup informations of a repository.

Homepage: [GenPackageDoc](#)

9. PythonExtensionsCollection

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. Also a file comparison feature is available.

Homepage: [PythonExtensionsCollection](#)

10. RobotframeworkExtensions

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 keyword (especially for composite Python data types). The **RobotframeworkExtensions** keywords are implemented in Python (see **PythonExtensionsCollection**).

Homepage: [RobotframeworkExtensions](#)

11. Tutorial

The RobotFramework AIO tutorial contains robot code examples together with the documentation explaining how to use these examples and how they work. The basic idea behind this tutorial is not to have theory only but additionally to this theory also the possibility to experience what is explained inside. The main focus of this tutorial is on some feature extensions available for the Robot Framework.

Homepage: [Tutorial](#)

12. Development environment

The RobotFramework AIO installer also installs VSCode, that is an open source editor. VSCode is preconfigured especially for the usage together with the RobotFramework AIO.

Features:

- Workspace with included testcases folder, documentation and tutorial
- Previews for certain formats like, MD, RST, PDF, PlantUML
- Support of the extended JSON syntax of **RobotFramework_TestsuitesManagement**
- Static code analysis by Robocop, that is an also included static code checker

Chapter 4

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 bundled 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 5

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 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:

```

	suite02.robot 3 ×
	robotframework-tutorial > 901_static_code_analysis > suite02.robot > ...
31  Test · Case · 0201
32  ····[Documentation]····Documentation·of·test·case
33  ····Log····I·am·test·'${TEST·NAME}'·of·suite·'${SUITE·NAME}'····console=yes
34  ····#·TODO···Add·return·value·to·keyword·implementation;·check·return·value·here
35  ····test_keyword_1
36
Run | Debug | Run in Interactive Console
37  Test · Case · 0201
38  ····[Documentation]····Documentation·of·test·case
39  ····Log····I·am·test·'${TEST·NAME}'·of·suite·'${SUITE·NAME}'····console=yes
40
41
42
43

```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, **/*.ts, !**/node_modules/**)

- suite02.robot robotframework-tutorial • 901_static_code_analysis 3
 - ⊗ Multiple test cases with name 'Test Case 0201' (first occurrence in line 31) robocop(0801) [37, 1]
 - ⚠ Found TODO in comment robocop(0701) [34, 7]
 - ⚠ Too many blank lines at the end of file robocop(1010) [43, 1]

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 6

Logging

6.1 Background

The Robot Framework provides several log levels (also known as trace levels) to give users the possibility to control the amount of output in log files. The build in keyword `log` writes content to the log files. The optional parameter `level` of this keyword defines under which conditions (under which log level) the `log` keyword shall write the log message.

With command line parameter `--loglevel` a user defines this condition. If the log level of an executed `log` keyword matches to the log level defined for test execution, the log message is logged, otherwise not.

Available log levels in Robot Framework are: `ERROR`, `WARN`, `INFO`, `DEBUG` and `TRACE`. The default log level in Robot Framework is `INFO`. This level becomes active, if nothing else is specified by a user.

The log level `INFO` is also used by the Robot Framework itself, e.g. for the logging of informations about executed keywords. The impact is that every log file contains both the users output and the Robot Framework output. For larger tests this might cause unfavorable large log files.

Therefore the RobotFramework AIO provides an additional log level `USER`. Tests that are executed under this log level, do not contain the Robot Framework `INFO` messages any more (but for sure still contain errors and warnings).

Example code:

```
log    my log message    USER
```

Example execution:

```
python -m robot --loglevel USER -b logfile.log ./testfile.robot
```

6.2 Summary

In opposite to the Robot Framework (core), the RobotFramework AIO provides the following log levels to control the amount of logged content:

		log level in <code>log</code> keyword					
		ERROR	WARN	USER	INFO	DEBUG	TRACE
log level in command line <code>--loglevel</code>	ERROR	logged	-----	-----	-----	-----	-----
	WARN	logged	logged	-----	-----	-----	-----
	USER	logged	logged	logged	-----	-----	-----
	INFO	logged	logged	logged	logged	-----	-----
	DEBUG	logged	logged	logged	logged	logged	-----
	TRACE	logged	logged	logged	logged	logged	logged

The log level `USER` has been added to give users the possibility to log own content without all `INFO` messages logged by the Robot Framework (core).

Chapter 7

Threading in Robot Framework

7.1 Background

The Robot Framework core (Robot Framework) now includes advanced threading features. This enhancement enables the simulation of concurrent processes and interactions within test environments.

7.2 Threading Implementation

The threading functionality in Robot Framework introduces the THREAD keyword, which allows for the execution of parallel processes within tests. Here are some key aspects of how THREAD works:

Keyword Usage: The THREAD keyword functions similarly to FOR, WHILE, or TRY keywords. It can be utilized within a test case or included in a User Defined Keyword. This flexibility allows for a wide range of multi-threaded test scenarios.

Immediate Thread Start: A thread starts executing immediately after its declaration. This means that as soon as the line THREAD *ThreadNameIsDaemon* is executed, the thread begins its operation. This immediate start is crucial for real-time testing and synchronization.

Variable Scope in Threads: Each thread has its own variable scope. Variables declared or modified within a thread do not affect the variables in other threads or the main test flow. This separation ensures that threads operate independently and reduces the risk of data conflicts.

Threading Syntax:

```
THREAD      ${ThreadName}      ${IsDaemon}
           [Thread's actions and keywords]
END
```

This syntax is used to define a thread in Robot Framework tests. Here's a breakdown of the parameters:

`${ThreadName}` : This is a required parameter that specifies the name of the thread. It's used for identification and logging purposes. The thread name should be unique within the context of the test suite to avoid confusion. Thread name is case-sensitive.

`${IsDaemon}` : This is an optional parameter that determines the nature of the thread. When set to true, the thread is treated as a daemon thread. Daemon threads are typically background tasks that exit when all non-daemon threads have completed. If not specified, the thread is considered a non-daemon thread by default.

The use of named threads and the option to set them as daemon threads allow for precise control over the threading behavior in test cases, facilitating complex test scenarios.

7.3 New Threading Keywords

7.3.1 Send Thread Notification Keyword

This keyword is utilized to send a notification from one thread to another, either to a specific thread or broadcasted to all threads.

Syntax:

```
Send Thread Notification    notification_name    params=None    dst_thread=None
```

Parameters:

`notification_name` : Mandatory. The unique name of the notification to be sent. This name is used by receiving threads to identify the notification.

`params` : Optional. The payload of the notification. It can be a string or any structured data and is passed to the receiving thread along with the notification. Default is None, which means no payload is sent.

`dst_thread` : Optional. The specific target thread to which the notification should be sent. If not specified or set to None, the notification is broadcasted to all threads.

7.3.2 Wait Thread Notification Keyword

Allows a thread to wait for a specific notification, providing a mechanism for synchronization and response to inter-thread communication.

Syntax:

```
Wait Thread Notification    notification_name    condition=None    timeout=5
```

Parameters:

`notification_name` : Mandatory. The name of the notification the thread is waiting for. This should match the name given in Send Thread Notification.

`condition` : Optional. This parameter allows specifying a Python expression as a condition that the notification's payload must satisfy for the notification to be acknowledged. The condition is written as a string and evaluated dynamically. Inside the condition, the variable payloads refers to the payload of the received notification.

For example, setting `condition="$payloads=='test'"` means that the thread will continue execution only if it receives a notification named `notification_name` whose payload is equal to the string "test". This allows for selective synchronization based on the content of notifications.

Example:

In this example, the thread waits for a notification named "DataUpdate" and proceeds only if the payload of this notification is "test".

```
Wait Thread Notification    DataUpdate    condition=$payloads=='test'    timeout=10
```

`timeout` : Optional. The maximum time in seconds to wait for the notification. If the notification is not received within this period, the keyword will fail. Default is 5 seconds.

7.4 Summary

With the introduction of threading and the new keywords

```
Send Thread Notification
```

and

```
Wait Thread Notification
```

RobotFramework AIO (Robot Framework) now offers advanced multi-threaded testing capabilities. These enhancements allow for more sophisticated, parallel testing scenarios and improve synchronization and communication between threads.

Chapter 8

Library documentation

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

RobotFramework AIO bundle

Version 0.11.0.2 (from 04.2024)

Underlying Robot Framework (core)

Robot Framework 7.0 (Python 3.9.2 on linux)

Included libraries

GenPackageDoc

Version 0.41.1 (from 11.10.2023)

https://github.com/test-fullautomation/python-genpackagedoc

<i>Documentation builder for Python packages</i>
--

PythonExtensionsCollection

Version 0.15.1 (from 19.10.2023)

https://github.com/test-fullautomation/python-extensions-collection

<i>Additional Python functions</i>

RobotframeworkExtensions

Version 0.10.0 (from 06.04.2023)

https://github.com/test-fullautomation/robotframework-extensions-collection

<i>Additional Robot Framework keywords</i>
--

JsonPreprocessor

Version 0.4.0 (from 15.03.2024)

https://github.com/test-fullautomation/python-jsonpreprocessor

<i>Preprocessor for json files</i>

RobotFramework_TestsuitesManagement

Version 0.7.6 (from 01.04.2024)

<https://github.com/test-fullautomation/robotframework-testsuitesmanagement>*Functionality to manage RobotFramework testsuites***QConnectBase**

Version 1.1.3 (from 06.06.2023)

<https://github.com/test-fullautomation/robotframework-qconnect-base>*Robot Framework test library for TCP, SSH, serial connection***QConnectWinapp**

Version 1.0.3 (from 19.10.2023)

<https://github.com/test-fullautomation/robotframework-qconnect-winapp>*Robot Framework QConnect library extension for Winapp GUI testing***RobotFramework_DBus**

Version 0.1.3 (from 19.10.2023)

<https://github.com/test-fullautomation/robotframework-dbus>*Robot Framework QConnect library extension for dbus testing***RobotLog2RQM**

Version 1.2.3 (from 14.03.2024)

<https://github.com/test-fullautomation/robotframework-robotlog2rqm>*Imports robot result(s) to IBM Rational Quality Manager (RQM)***RobotLog2DB**

Version 1.4.1 (from 15.03.2024)

<https://github.com/test-fullautomation/robotframework-robotlog2db>*Imports robot result(s) to TestResult WebApp database***RobotFramework_DoIP**

Version 0.1.2 (from 08.04.2024)

<https://github.com/test-fullautomation/robotframework-doip>*RobotFramework for DoIP Client***PyTestLog2DB**

Version 0.2.8 (from 14.12.2023)

<https://github.com/test-fullautomation/python-pytestlog2db>*Imports pytest result(s) to TestResult WebApp database***TestResultWebApp**

Version 0.1.3 (from 18.10.2022)

<https://github.com/test-fullautomation/testresultwebapp>*Web based display of test results*

8.1 GenPackageDoc

GenPackageDoc

v. 0.41.1

Holger Queckenstedt

11.10.2023

Contents

1	Introduction	1
2	Description	2
2.1	Repository content	2
2.2	Documentation build process	3
2.3	PDF document structure	5
2.4	Examples	6
2.4.1	Example 1: RST file	6
2.4.2	Example 2: Python module	6
2.5	Interface and module descriptions	7
2.6	Runtime variables	8
2.7	Syntax aspects	9
2.7.1	Common rules	9
2.7.2	Syntax extensions	9
2.8	Diagrams	10
2.8.1	Example 1: Sequence diagram	10
2.8.2	Example 2: JSON diagram	11
2.8.3	Picture import	11
3	CDocBuilder.py	12
3.1	Class: CDocBuilder	12
3.1.1	Method: Build	12
4	CInterface.py	13
4.1	Class: CInterface	13
4.1.1	Method: GetLaTeXStyles	13
5	CPackageDocConfig.py	14
5.1	Function: printerror	14
5.2	Class: CPackageDocConfig	14
5.2.1	Method: PrintConfig	14
5.2.2	Method: PrintConfigKeys	14
5.2.3	Method: Get	14
5.2.4	Method: GetConfig	14
6	CPatterns.py	15
6.1	Class: CPatterns	15
6.1.1	Method: GetHeader	15
6.1.2	Method: GetChapter	16

<u>CONTENTS</u>	<u>CONTENTS</u>
6.1.3 Method: GetFooter	16
6.1.4 Method: GetAutodefinedHeader	16
7 CSourceParser.py	17
7.1 Class: CSourceParser	17
7.1.1 Method: ParseSourceFile	17
8 Appendix	18
9 History	19

CHAPTER 1. INTRODUCTION

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 its 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

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).

CHAPTER 2. DESCRIPTION2.2. DOCUMENTATION BUILD PROCESS

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 subsection 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 Description	2
2.1 Repository content	2
2.2 Documentation build process	3
2.3 PDF document structure	5
2.4 Examples	6
2.4.1 Example 1: RST file	6
2.4.2 Example 2: Python module	6

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:

3 CDocBuilder.py	10
3.1 Class: CDocBuilder	10
3.1.1 Method: Build	10

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.

2.8 Diagrams

A *diagram* in this context is a picture that is rendered out of source code. **GenPackageDoc** supports **PlantUML** that supports a wide range of diagrams.

To use the **PlantUML** functionality with **GenPackageDoc**, some preconditions have to be fulfilled:

1. **PlantUML** is installed (either as stand-alone installation or as VSCodium extension)
2. **GenPackageDoc** is configured (`packagedoc_config.json`):
 - a. In section "`DIAGRAMS`" a path to a diagrams folder is defined (containing the diagrams source code).
 - b. In section "`JAVA`" path and name of the java interpreter is defined (because **PlantUML** is a java application).
 - c. In section "`PLANT_UML`" path and name of the **PlantUML** application is defined.

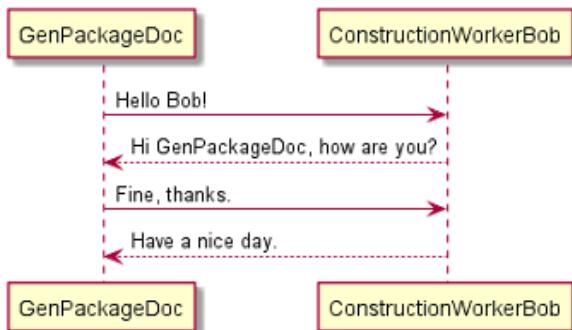
All **PlantUML** source code files within the "`DIAGRAMS`" folder need to have the extension `puml`.

2.8.1 Example 1: Sequence diagram

The following code of a `puml` file produces a sequence diagram:

```
@startuml
GenPackageDoc -> ConstructionWorkerBob: Hello Bob!
ConstructionWorkerBob --> GenPackageDoc: Hi GenPackageDoc, how are you?
GenPackageDoc -> ConstructionWorkerBob: Fine, thanks.
ConstructionWorkerBob --> GenPackageDoc: Have a nice day.
@enduml
```

Result:



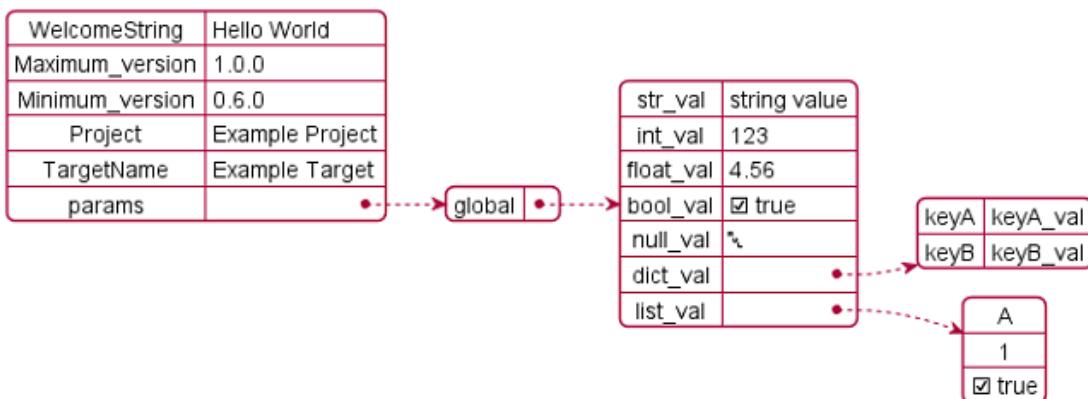
2.8.2 Example 2: JSON diagram

The following code of a puml file produces a sequence diagram:

```
@startjson
{
    "WelcomeString"      : "Hello World",
    "Maximum_version"   : "1.0.0",
    "Minimum_version"   : "0.6.0",
    "Project"           : "Example Project",
    "TargetName"         : "Example Target",

    "params" : {
        "global" : {
            "str_val"      : "string value",
            "int_val"      : 123,
            "float_val"    : 4.56,
            "bool_val"     : true,
            "null_val"     : null,
            "dict_val"     : {"keyA" : "keyA_val", "keyB" : "keyB_val"},
            "list_val"     : ["A", 1, true]
        }
    }
}
@endjson
```

Result:



2.8.3 Picture import

In case of **PlantUML** is configured in the **GenPackageDoc** configuration and in case of puml files are available within the diagrams folder, **GenPackageDoc** automatically calls **PlantUML** to render the diagrams. They can be imported in the following way:

1. Import in RST code:

```
.. image:: ./diagrams/SequenceDiagram.png
```

2. Import in LaTeX code:

```
\includegraphics[scale=0.7]{./diagrams/SequenceDiagram.png}
```

The user needs to adapt the scaling to make the rendered diagrams fit to a page of a PDF document in best way. But this scaling only works in LaTeX code, but not in RST code.

CHAPTER 3. CDOCBUILDER.PY

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

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

Chapter 5

CPackageDocConfig.py

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

5.1 Function: printerror

5.2 Class: CPackageDocConfig

Imported by:

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

5.2.1 Method: PrintConfig

Prints all configuration values to console.

5.2.2 Method: PrintConfigKeys

Prints all configuration key names to console.

5.2.3 Method: Get

Returns the configuration value belonging to a key name.

5.2.4 Method: GetConfig

Returns the complete configuration dictionary.

CHAPTER 6. CPATTERNS.PY

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

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

Chapter 8

Appendix

About this package:

Table 8.1: Package setup

Setup parameter	Value
Name	GenPackageDoc
Version	0.41.1
Date	11.10.2023
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

Chapter 9

History

History of **GenPackageDoc** (hosted in repository [python-genpackagedoc](#)).

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

CHAPTER 9. HISTORY

<i>LaTeX style definitions moved to separate folder</i>	
0.14.0	27.05.2022
- <i>LaTeX compiler check added</i>	
- <i>Control parameter STRICT added to packagedoc_config</i>	
0.15.0	31.05.2022
- <i>Command line added</i>	
- <i>Separate GenPackageDoc configuration class added</i>	
0.16.0	01.06.2022
<i>Path computation reworked</i>	
0.17.0	17.06.2022
- <i>Configuration dump added</i>	
- <i>Code maintenance</i>	
- <i>Error handling extended</i>	
0.18.0	20.06.2022
<i>Added parameter to define an output folder for a dump of final configuration</i>	
0.19.0	28.06.2022
- <i>Method GetLaTeXStyles added</i>	
- <i>PythonExtensionsCollection updated to version 0.8.0</i>	
0.20.0	29.06.2022
<i>Document title bugfix: Added missing masking of underlines (required for LaTeX)</i>	
0.21.0	12.07.2022
<i>Separated file preamble.tex</i>	
0.22.0	13.07.2022
- <i>Maintenance of preamble.tex and styles folder</i>	
- <i>setup.py fix (install tex files also)</i>	
0.23.0	13.07.2022
<i>Maintenance of preamble.tex</i>	
0.24.0	25.07.2022
<i>Maintenance of robotframeworkkai0.sty (line breaks in listings)</i>	
0.25.0	27.07.2022
<i>Layout maintenance of RobotFramework AIO syntax highlighting (.sty files)</i>	
0.26.0	27.07.2022
<i>History reworked; common.sty introduced</i>	
0.27.0	17.08.2022
<i>Added LaTeX style definition for Python syntax highlighting</i>	
0.28.0	23.08.2022
- <i>Introduced new LaTeX environment variable GENDOC_LATEXPATH</i>	
- <i>robotframeworkkai0.sty aligned to version in GenMainDoc</i>	
0.29.0	24.08.2022
<i>Changed the way the import path of a module is printed out to PDF file</i>	
0.30.0	31.08.2022
<i>Introduced simulateonly mode (command line switch to skip the PDF generation)</i>	
0.31.0	12.09.2022
<i>Fix of import path of a module in PDF file</i>	
0.32.0	16.09.2022

CHAPTER 9. HISTORY

<ul style="list-style-type: none"> - Added labels at chapter level - partial rework of label mechanisms 	
0.33.0	19.09.2022
<i>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)</i>	
0.34.0	07.11.2022
<i>Introduced auto defined LaTeX style file containing mnemotechnical commands to type the repository name and the package name</i>	
0.35.0	16.11.2022
<i>Layout settings of some LaTeX commands adapted:</i> <ul style="list-style-type: none"> - Repository name and package name in bold - Inline code and inline listings in clearer colors 	
0.36.0	17.11.2022
<i>Brightness of all listings colors reduced to 45% (both text boxes and inline)</i>	
0.37.0	21.11.2022
<ul style="list-style-type: none"> - LaTeX style adaptions and bugfixes - Introduced LaTeX commands <code>Python_log</code> and <code>pylog</code> - Added keyword decorator detection - Feature 'INCLUDEPRIVATE' temporarily switched off (requires bugfixes) 	
0.38.0	30.11.2022
<i>Removed harming ligatures that were added by Pandoc automatically to LaTeX code in case of multiple minus characters in names</i>	
0.39.0	06.01.2023
<i>Added masking of underlines in case of the content of \repo or \pkg contain underlines (masking required by LaTeX)</i>	
0.40.0	09.05.2023
<i>Added PlantUML support to render diagrams out of source code in text format</i>	
0.41.0	11.07.2023
<i>Added text macro (\q{}) to set "double quotes"</i>	

GenPackageDoc.pdf

*Created at 10.04.2024 - 12:33:35
by GenPackageDoc v. 0.41.1*

8.2 PythonExtensionsCollection

PythonExtensionsCollection

v. 0.15.1

Holger Queckenstedt

19.10.2023

Contents

1	Introduction	1
2	Description	2
2.1	Modules	2
2.2	Methods	2
2.2.1	String operations with CString	2
2.2.2	File access with CFile	11
2.2.3	Utilities	15
3	CComparison.py	16
3.1	Class: CComparison	16
3.1.1	Method: Compare	16
4	CFile.py	18
4.1	Class: enFileStatiType	18
4.2	Class: CFile	18
4.2.1	Method: Close	18
4.2.2	Method: Delete	19
4.2.3	Method: Write	19
4.2.4	Method: Append	20
4.2.5	Method: ReadLines	20
4.2.6	Method: GetFileInfo	22
4.2.7	Method: CopyTo	22
4.2.8	Method: MoveTo	23
5	CFolder.py	24
5.1	Function: rm_dir_readonly	24
5.2	Class: CFolder	24
5.2.1	Method: Delete	24
5.2.2	Method: Create	25
5.2.3	Method: CopyTo	25
6	CString.py	27
6.1	Class: CString	27
6.1.1	Method: NormalizePath	27
6.1.2	Method: DetectParentPath	28
6.1.3	Method: StringFilter	29
6.1.4	Method: FormatResult	31

<u>CONTENTS</u>	<u>CONTENTS</u>
7 CUtils.py	32
7.1 Function: PrettyPrint	32
7.2 Class: CTypePrint	33
7.2.1 Method: TypePrint	33
7.3 Class: CUtils	33
7.3.1 Method: GetInstalledPackages	34
8 Appendix	35
9 History	36

CHAPTER 1. INTRODUCTION

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

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

CHAPTER 2. DESCRIPTION2.2. METHODS

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
```

CHAPTER 2. DESCRIPTION2.2. METHODS**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)
```

CHAPTER 2. DESCRIPTION2.2. METHODS**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 semicolon 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 accidentally.
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.

CHAPTER 2. DESCRIPTION2.2. METHODS

- `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 returns 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

CHAPTER 2. DESCRIPTION2.2. METHODS

This can be implemented in the following way.

Read a subset of file content:

```
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 explicitly.

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 implicitly:

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

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 explicitly:

```
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. **Explicitly 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.

CHAPTER 2. DESCRIPTION2.2. METHODS

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 if the file exists or not, it has to be coded explicitly:

```
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 its 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

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).
- **sIgnorePatternFile**
/ Condition: optional / Type: str / Default: None /
Pattern file containing a set of strings (**not** regular expressions; line by line). Every line containing one of the strings, is skipped.

Returns:

- **bIdentical**
/ Type: bool /
Indicates if the two input files (or their extracts) have the same content or not.
- **bSuccess**
/ Type: bool /
Indicates if the computation of the method was successful or not.

CHAPTER 3. CCOMPARISON.PY3.1. CLASS: CCOMPARISON

• sResult

/ Type: str /

The result of the computation of the method.

CHAPTER 4. CFILE.PY

Chapter 4

CFile.py

4.1 Class: enFileType

Imported by:

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

The class `enFileType` defines the following 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:

CHAPTER 4. CFFILE.PY4.2. CLASS: CFFILE

- 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.

CHAPTER 4. CFILE.PY4.2. CLASS: CFILE

- `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

CHAPTER 4. CFILE.PY4.2. CLASS: CFILE

- **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:

CHAPTER 4. CFILE.PY4.2. CLASS: CFFILE

- **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

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:

CHAPTER 5. CFOLDER.PY5.2. CLASS: CFOLDER

- `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 its 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 /

CHAPTER 5. CFOLDER.PY5.2. CLASS: CFOLDER

- 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

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. Paths can start with environment variables. Accepted are notations for both Windows (%ENVVAR%) and Linux (\${ENVVAR}). Under Windows also the Linux notation will be resolved.

- `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.

CHAPTER 6. CSTRING.PY6.1. CLASS: CSTRING

– 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 sFolderName 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 fulfills 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

CHAPTER 6. CSTRING.PY6.1. CLASS: CSTRING

- 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 code 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. CUTILLS.PY

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:

CHAPTER 7. CUTILLS.PY7.2. CLASS: CTYPEPRINT

- `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`.

7.3 Class: CUtils

Imported by:

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

The class `CUtils` contains useful methods.

7.3.1 Method: GetInstalledPackages

The method `GetInstalledPackages` computes a list of all installed Python packages. The list is returned as list of tuples containing the name and the version of the package.

It is also possible to let the method dump the list to a text file.

Arguments:

- `sOutputFile`
/ *Condition*: optional / *Type*: string / *Default*: None /
Path and name of a file to dump the package list to.

Returns:

- `listofTuplesPackages`
/ *Type*: list /
List of tuples containing the name and the version of the package.
- `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 8. APPENDIX

Chapter 8

Appendix

About this package:

Table 8.1: Package setup

Setup parameter	Value
Name	PythonExtensionsCollection
Version	0.15.1
Date	19.10.2023
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

Chapter 9

History

0.1.0	08/2021
<i>Initial version</i>	
0.2.0	02/2022
<i>Code maintenance</i>	
0.3.0	20.05.2022
<i>Documentation tool chain switched to GenPackageDoc</i>	
0.4.0	24.05.2022
- Documentation rebuild with GenPackageDoc v. 0.13.0 - Code maintenance	
0.5.0	31.05.2022
<i>Adapted to GenPackageDoc v. 0.15.0</i>	
0.6.0	02.06.2022
- Documentation rebuild with GenPackageDoc v. 0.16.0 - Code maintenance	
0.7.0	10.06.2022
<i>Module CFolder added (with methods: Create and Delete)</i>	
0.8.0	28.06.2022
<i>Module CFolder: Method: CopyTo added</i>	
0.9.0	27.07.2022
<i>History reworked (requires GenPackageDoc v. 0.26.0 at least)</i>	
0.10.0	25.10.2022
<i>Added CComparison module to compare two files (either based on original content or based on pattern)</i>	
0.11.0	15.11.2022
<i>Converted parts of the documentation from RST format to LaTeX format (to improve the layout)</i>	
0.12.0	29.03.2023
<i>Type RobotFramework_TestsuitesManagement.Config.CConfig.dotdict added to CTypePrint</i>	
0.13.0	03.04.2023
<i>Ignore pattern file added to CComparison</i>	
0.14.0	12.05.2023
<i>Removed computation of obsolete dotdict class from method CUtils::TypePrint</i>	

CHAPTER 9. HISTORY

PythonExtensionsCollection.pdf

Created at 10.04.2024 - 12:33:37

by GenPackageDoc v. 0.41.1

8.3 RobotframeworkExtensions

RobotframeworkExtensions

v. 0.10.0

Holger Queckenstedt

06.04.2023

CONTENTSCONTENTS

Contents

1	Introduction	1
2	Description	2
2.1	Keywords	2
2.1.1	pretty_print	2
2.1.2	normalize_path	5
3	Collection.py	7
3.1	Class: Collection	7
3.1.1	Keyword: pretty_print	7
3.1.2	Keyword: normalize_path	8
4	Appendix	9
5	History	10

CHAPTER 1. INTRODUCTION

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

Chapter 2

Description

2.1 Keywords

The `Collection` module of the **RobotframeworkExtensions** is the interface between the **PythonExtensionCollection** 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.

CHAPTER 2. DESCRIPTION2.1. KEYWORDS**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.

[CHAPTER 2. DESCRIPTION](#)[2.1. KEYWORDS](#)**Prefix strings**

Prefix strings (`sPrefix`) can be used to give the lines of output a meaning, or they are used just to print also the name of the pretty printed variable (`oData`).

Example

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

Result

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

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}
```

CHAPTER 2. DESCRIPTION2.1. KEYWORDS

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

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
- `sPrefix`
/ *Condition:* optional / *Type:* str / *Default:* None /
If not None, this prefix string is added to every output line.

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

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	RobotframeworkExtensions
Version	0.10.0
Date	06.04.2023
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

Chapter 5

History

0.1.0	01/2022
<i>Initial version</i>	
0.2.0	03/2022
<i>Setup maintenance</i>	
0.3.0	05/2022
<i>Documentation tool chain switched to GenPackageDoc</i>	
0.4.0	24.05.2022
- Documentation rebuild with GenPackageDoc v. 0.13.0	
- Code maintenance	
0.5.0	31.05.2022
<i>Adapted to GenPackageDoc v. 0.15.0</i>	
0.6.0	02.06.2022
- Documentation rebuild with GenPackageDoc v. 0.16.0	
- Code maintenance	
0.7.0	28.06.2022
<i>PythonExtensionsCollection updated to version 0.8.0</i>	
0.8.0	27.07.2022
<i>History reworked (requires GenPackageDoc v. 0.26.0 at least)</i>	
0.9.0	29.03.2023
<i>Prefix parameter added to keyword pretty_print</i>	
0.10.0	06.04.2023
<i>Added logging of pretty_print output to console</i>	

RobotframeworkExtensions.pdf*Created at 10.04.2024 - 12:33:40**by GenPackageDoc v. 0.41.1*

8.4 JsonPreprocessor

JsonPreprocessor

v. 0.4.0

Mai Dinh Nam Son

15.03.2024

CONTENTS

CONTENTS

Contents

1	Introduction	1
2	Description	3
2.1	How to execute	3
2.2	VSCodium support	4
3	The JSONP format	5
3.1	Standard JSON format	5
3.2	Boolean and null values	6
3.3	Comments	7
3.4	Import of JSON files	8
3.5	Overwrite parameters	9
3.6	dotdict notation	18
3.7	Substitution of dollar operator expressions	20
3.8	Overwriting vs. substitution	22
3.9	Implicite creation of dictionaries	23
4	CJsonPreprocessor.py	25
4.1	Class: CSyntaxType	25
4.2	Class: CNameMangling	25
4.3	Class: CPythonJSONDecoder	25
4.3.1	Method: custom_scan_once	25
4.4	Class: CJsonPreprocessor	25
4.4.1	Method: getVersion	26
4.4.2	Method: getVersionDate	26
4.4.3	Method: jsonLoad	26
4.4.4	Method: jsonDump	26
5	Appendix	27
6	History	28

CHAPTER 1. INTRODUCTION

Chapter 1

Introduction

JavaScript Object Notation (**JSON**) is a text-based format for storing any user defined data and can also be used for data interchange between different applications.

But this format has some limitations and the **JsonPreprocessor** has been introduced to fill the gaps.

The **JsonPreprocessor** extends the JSON format by the following features:

1. Parts of a JSON file can be commented out
2. A JSON file can import other JSON files (nested imports)
3. Parameter can be defined, referenced and overwritten (follow up definitions in configuration files overwrite previous definitions of the same parameter)
4. Also Python specific keywords like `True` , `False` and `None` can be used (additionally to the corresponding JSON keywords `true` , `false` and `null`)

The main goal of the **JsonPreprocessor** is to support huge sets of parameters for complex projects. And the features of the **JsonPreprocessor** support this complexity:

1. Like in usual programming languages code comments are useful to explain the meaning of the defined parameters.
2. Splitting all required parameters into several JSON files - that can import each other - enables to distinguish e.g. between local and global parameters or between specific and common parameters. Another advantage of a file split is: Smaller files with a more specific content are easier to maintain than a huge single file that contains all.
3. A possible use case for a file split would be to have a software containing several different components with each component requires an individual set of parameters - and therefore an own JSON file. Additionally all components also require a common set of parameters. In this case all common parameters can be defined within an own JSON file that is imported into all other JSON files containing the specific values. This procedure avoids redundancy in parameter definitions.
4. Parameters can be initialized in common JSON files and overwritten in specific JSON files that import the common ones.

But this has consequences: The new features cause some deviations from JSON standard.

These deviations harm the syntax highlighting of editors and also cause invalid findings of JSON format related static code checkers.

To avoid conflicts between the standard JSON format and the extended JSON format described here, the **JsonPreprocessor** uses the alternative file extension `.jsonp` for all JSON files.

CHAPTER 1. INTRODUCTION

References:

The **JsonPreprocessor** is hosted in PyPi (recommended for users) and in GitHub (recommended for developers):

- [JsonPreprocessor in PyPi](#)
- [JsonPreprocessor in GitHub](#)

Details about how to get the **JsonPreprocessor** can be found in the [README](#).

For the development environment **VSCodium** an extension is available to support the extended JSON format of the **JsonPreprocessor**: [vscode-jsonp](#)

CHAPTER 2. DESCRIPTION

Chapter 2

Description

2.1 How to execute

The **JsonPreprocessor** is implemented in Python3 and therefore requires a Python3 installation.

A basic Python script to use the **JsonPreprocessor** can look like this:

```
from JsonPreprocessor.CJsonPreprocessor import CJsonPreprocessor
import pprint

json_preprocessor = CJsonPreprocessor()
try:
    values = json_preprocessor.jsonLoad("./file.jsonp")
    pprint.pprint(values)
except Exception as reason:
    print(f'{reason}')
```

The main method of the **JsonPreprocessor** is: `jsonLoad`. Input is the path and the name of a JSON file. Output is a dictionary containing all values parsed from this JSON file.

In case of any errors while computing the JSON file, the **JsonPreprocessor** throws an exception. Therefore it is required to call the method `jsonLoad` inside a `try/except` block.

`pprint` is used in this example to give the output a better readability in console.

In chapter [The JSONP format](#) the format of JSON files used by the **JsonPreprocessor**, is described in detail. All discussed JSON files can be tested with the example script listed above.

2.2 VSCodium support

In the introduction we mentioned that the JSON syntax extensions introduced by the **JsonPreprocessor**, harm the syntax highlighting of editors.

Either we give the JSON files the extension `.json`, then an editor expects a JSON file in standard syntax, or we change the extension to `.jsonp`, but in this case an editor usually does not know how to display a file of such type.

In case you use **VSCodium**, you can install a [jsonp extension](#).

With this extension the VSCodium editor will be able to display `.jsonp` files properly.

Some Impressions:

```
{
    //---initialization
    "project_values": {},
    //-
    //---add some common values
    ${project_values}['common_project_param_1'] : "common-project-value-1",
    ${project_values}['common_project_param_2'] : "common-project-value-2",
    //
    //---import feature parameters
    "[import]": "./featureA.jsonp",
    "[import]": "./featureB.jsonp",
    "[import]": "./featureC.jsonp"
}
```

```
//--a) standard notation
"dict_1_key_2_value_2_standard": ${params}[0]['dict_1_key_2'][1],
//--b) dotdict notation
"dict_1_key_2_value_2_dotdict": ${params.0.dict_1_key_2.1},
//--> 'The variable '${params}[0]['dict_1_key_2'][1]' is not available'
//
//--c) standard notation
"dict_2_A_key_2_value_2_standard": ${params}[1]['dict_2_key_2'][1][1]
```

Chapter 3

The JSONP format

This chapter explains the format of JSON files used by the **JsonPreprocessor** in detail. We concentrate here on the content of the JSON files and the corresponding results, available in Python dictionary format.

3.1 Standard JSON format

The **JsonPreprocessor** supports JSON files with standard extension `.json` and standard content.

- JSON file:

```
{
    "param1" : "value1",
    "param2" : "value2"
}
```

Outcome:

```
{'param1': 'value1', 'param2': 'value2'}
```

A JSON file with extension `.jsonp` and same content will produce the same output.

We recommend to give every JSON file the extension `.jsonp` to have a strict separation between the standard and the extended JSON format.

The following example still contains standard JSON content, but with parameters of several different data types (simple and composite).

```
{
    "param_01" : "string",
    "param_02" : 123,
    "param_03" : 4.56,
    "param_04" : [ "A", "B", "C" ],
    "param_05" : { "A" : 1, "B" : 2, "C" : 3 }
}
```

This content produces the following output:

```
{'param_01': 'string',
'param_02': 123,
'param_03': 4.56,
'param_04': ['A', 'B', 'C'],
'param_05': {'A': 1, 'B': 2, 'C': 3}}
```

3.2 Boolean and null values

JSON supports the boolean values `true` and `false`, and also the null value `null`.

In Python the corresponding values are different: `True`, `False` and `None`.

Because the **JsonPreprocessor** is a Python application and therefore the returned content is required to be formatted Python compatible, the **JsonPreprocessor** does a conversion automatically.

Accepted in JSON files are both styles:

```
{  
    "param_06" : true,  
    "param_07" : false,  
    "param_08" : null,  
    "param_09" : True,  
    "param_10" : False,  
    "param_11" : None  
}
```

The output contains all keywords in Python style only:

```
{'param_06': True,  
'param_07': False,  
'param_08': None,  
'param_09': True,  
'param_10': False,  
'param_11': None}
```

3.3 Comments

Comments can be added to JSON files with `//`:

```
{  
    // JSON keywords  
    "param_06" : true,  
    "param_07" : false,  
    "param_08" : null,  
    // Python keywords  
    "param_09" : True,  
    "param_10" : False,  
    "param_11" : None  
}
```

All lines starting with `//`, are ignored by the **JsonPreprocessor**. The output of this example is the same than in the previous example.

Also block comments and inline comments are possible, realized by a pair of `/* */`:

```
{  
    /*  
        "param1" : 1,  
        "param2" : "A",  
    */  
  
    "testlist" : ["A1", /*"B2", "C3",*/ "D4"]  
}
```

Outcome:

```
{"testlist": ["A1", 'D4']}
```

3.4 Import of JSON files

We assume the following scenario:

A software component *A* requires a set of configuration parameters. A software component *B* that belongs to the same main software or to the same project, requires another set of configuration parameters. Additionally both components require a common set of parameters (with the same values).

The outcome is that at least we need two JSON configuration files:

1. A file `componentA.jsonp` containing all parameters required for component *A*
2. A file `componentB.jsonp` containing all parameters required for component *B*

But with this solution both JSON files would contain also the common set of parameters. This is unfavorable, because the corresponding values need to be maintained at two different positions.

Therefore we extend the list of JSON files by a file containing the common part only:

1. A file `common.jsonp` containing all parameters that are the same for component *A* and component *B*
2. A file `componentA.jsonp` containing remaining parameters (with specific values) required for component *A*
3. A file `componentB.jsonp` containing remaining parameters (with specific values) required for component *B*

Finally we use the import mechanism of the **JsonPreprocessor** to import the file `common.jsonp` in file `componentA.jsonp` and also in file `componentB.jsonp`.

This can be the content of the JSON files:

```

• common.jsonp
{
    // common parameters
    "common_param_1" : "common value 1",
    "common_param_2" : "common value 2"
}

• componentA.jsonp
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component A parameters
    "componentA_param_1" : "componentA value 1",
    "componentA_param_2" : "componentA value 2"
}

• componentB.jsonp
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component B parameters
    "componentB_param_1" : "componentB value 1",
    "componentB_param_2" : "componentB value 2"
}

```

Explanation:

JSON files are imported with the key `"[import]"`. The value of this key is the path and name of the JSON file to be imported.

A JSON file can contain more than one import. Imports can be nested: An imported JSON file can import further JSON files also.

Outcome:

The file `componentA.jsonp` produces the following output:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common value 2',
 'componentA_param_1': 'componentA value 1',
 'componentA_param_2': 'componentA value 2'}
```

The file `componentB.jsonp` produces the following output:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common value 2',
 'componentB_param_1': 'componentB value 1',
 'componentB_param_2': 'componentB value 2'}
```

It can be seen that the returned dictionary contains both the parameters from the loaded JSON file and the parameters imported by the loaded JSON file.

3.5 Overwrite parameters

We take over the scenario from the previous section: We still have a JSON file `componentA.jsonp` containing the parameters for component *A*, a JSON file `componentB.jsonp` for component *B* and a JSON file `common.jsonp` for both components.

But now component *B* requires a different value of a common parameter: Within a JSON file we need to change the value of a parameter that is initialized within an imported file. That is possible.

This is now the content of the JSON files:

- `common.jsonp`

```
{
    // common parameters
    "common_param_1": "common value 1",
    "common_param_2": "common value 2"
}
```

- `componentA.jsonp`

```
{
    // common parameters
    "[import]": "./common.jsonp",
    //
    // component A parameters
    "componentA_param_1": "componentA value 1",
    "componentA_param_2": "componentA value 2"
}
```

- `componentB.jsonp`

```
{
    // common parameters
    "[import]": "./common.jsonp",
    //
    // component B parameters
    "componentB_param_1": "componentB value 1",
    "componentB_param_2": "componentB value 2",
    // overwrite parameter initialized by imported file
    "common_param_2": "common componentB value 2"
}
```

Explanation:

With

```
"common_param_2" : "common componentB value 2"
```

in `componentB.jsonp`, the initial definition

```
"common_param_2" : "common value 2"
```

in `common.jsonp` is overwritten.

Outcome:

The file `componentB.jsonp` produces the following output:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common componentB value 2',
 'componentB_param_1': 'componentB value 1',
 'componentB_param_2': 'componentB value 2'}
```

Important: *The value a parameter has finally, depends on the order of definitions, redefinitions and imports!*

In file `componentB.jsonp` we move the import of `common.jsonp` to the bottom:

```
{
    // component B parameters
    "componentB_param_1" : "componentB value 1",
    "componentB_param_2" : "componentB value 2",
    "common_param_2" : "common componentB value 2"
    //
    // common parameters
    "[import]" : "./common.jsonp",
}
```

Now the imported file overwrites the value initialized in the importing file.

Outcome:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common value 2',
 'componentB_param_1': 'componentB value 1',
 'componentB_param_2': 'componentB value 2'}
```

Up to now we considered simple data types only. In case we want to overwrite a parameter that is part of a composite data type, we need to extend the syntax. This is explained in the next examples.

Again we take over the scenario from the previous section: We still have a JSON file `componentA.jsonp` containing the parameters for component A, a JSON file `componentB.jsonp` for component B and a JSON file `common.jsonp` for both components.

But now all values are part of composite data types like lists and dictionaries.

This is the content of the JSON files:

- `common.jsonp`

```
{
    // common parameters
    "common_param_1" : ["common value 1.1", "common value 1.2"],
    "common_param_2" : {"common_key_2_1" : "common value 2.1",
                      "common_key_2_2" : "common value 2.2"}
}
```

- componentA.jsonp

```
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component A parameters
    "componentA_param_1" : ["componentA value 1.1", "componentA value 1.2"],
    "componentA_param_2" : {"componentA_key_2_1" : "componentA value 2.1",
                           "componentA_key_2_2" : "componentA value 2.2"}
}
```

- componentB.jsonp

```
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component B parameters
    "componentB_param_1" : ["componentB value 1.1", "componentB value 1.2"],
    "componentB_param_2" : {"componentB_key_2_1" : "componentB value 2.1",
                           "componentB_key_2_2" : "componentB value 2.2"}
}
```

Like in previous examples, the outcome is a merge of the imported JSON file and the importing JSON file, e.g. for componentA.jsonp :

```
{"common_param_1": ['common value 1.1', 'common value 1.2'],
 "common_param_2": {"common_key_2_1": 'common value 2.1',
                   'common_key_2_2': 'common value 2.2'},
 'componentA_param_1': ['componentA value 1.1', 'componentA value 1.2'],
 'componentA_param_2': {"componentA_key_2_1": 'componentA value 2.1',
                       'componentA_key_2_2': 'componentA value 2.2'}}}
```

Now the following questions need to be answered:

1. How to get the value of an already existing parameter?
2. How to get the value of a single element of a parameter of nested data type (list, dictionary)?
3. How to overwrite the value of a single element of a parameter of nested data type?
4. How to add an element to a parameter of nested data type?

We introduce another JSON file componentB.2.jsonp in which we import the JSON file componentB.jsonp . In this file we also add content to work with simple and composite data types to answer the questions above.

CHAPTER 3. THE JSONP FORMAT3.5. OVERWRITE PARAMETERS

This is the initial content of `componentB.2.jsonp`:

```
{
    // import of componentB parameters
    "[import]" : "./componentB.jsonp",
    //
    // some additional parameters of simple data type
    "string_val" : "ABC",
    "int_val" : 123,
    "float_val" : 4.56,
    "bool_val" : true,
    "null_val" : null,

    // access to existing parameters
    "string_val_b" : ${string_val},
    "int_val_b" : ${int_val},
    "float_val_b" : ${float_val},
    "bool_val_b" : ${bool_val},
    "null_val_b" : ${null_val},
    "common_param_1_b" : ${common_param_1},
    "componentB_param_2_b" : ${componentB_param_2}
}
```

The rules for accessing parameters are:

- Existing parameters are accessed by a dollar operator and a pair of curly brackets (`${...}`) with the parameter name inside.
- If the entire expression of the right hand side of the colon is such a dollar operator expression, it is not required any more to encapsulate this expression in quotes.
- Without quotes, the dollar operator keeps the data type of the referenced parameter. If you use quotes, the value of the used parameter will be of type `str`.

Outcome:

```
{
    'bool_val': True,
    'bool_val_b': True,
    'common_param_1': ['common value 1.1', 'common value 1.2'],
    'common_param_1_b': ['common value 1.1', 'common value 1.2'],
    'common_param_2': {'common_key_2_1': 'common value 2.1',
                      'common_key_2_2': 'common value 2.2'},
    'componentB_param_1': ['componentB value 1.1', 'componentB value 1.2'],
    'componentB_param_2': {'componentB_key_2_1': 'componentB value 2.1',
                          'componentB_key_2_2': 'componentB value 2.2'},
    'componentB_param_2_b': {'componentB_key_2_1': 'componentB value 2.1',
                            'componentB_key_2_2': 'componentB value 2.2'},
    'float_val': 4.56,
    'float_val_b': 4.56,
    'int_val': 123,
    'int_val_b': 123,
    'null_val': None,
    'null_val_b': None,
    'string_val': 'ABC',
    'string_val_b': 'ABC'}
```

Let's take a deeper look at the following line:

```
"int_val_b" : ${int_val},
```

Like mentioned in the rules above, the dollar operator keeps the data type of the referenced parameter: In case of `int_val` is of type `int`, also `int_val_b` will be of type `int`.

Like mentioned in the rules above, it is not required any more to encapsulate dollar operator expressions at the right hand side of the colon in quotes. But nevertheless it is possible to use quotes. In case of:

```
"int_val_b" : "${int_val}",
```

the parameter `int_val_b` is of type `string`.

Value of a single element of a parameter of nested data type

To access an element of a list and a key of a dictionary, we change the content of file `componentB.2.jsonp` to:

```
{
  // import of componentB parameters
  "[import]" : "./componentB.jsonp",
  //
  "list_element_0" : ${componentB_param_1}[0],
  "dict_key_2_2" : ${common_param_2}['common_key_2_2']
}
```

Outcome:

```
{"common_param_1": ["common value 1.1", "common value 1.2"],
 "common_param_2": {"common_key_2_1": "common value 2.1",
                   "common_key_2_2": "common value 2.2"},
 "componentB_param_1": ["componentB value 1.1", "componentB value 1.2"],
 "componentB_param_2": {"componentB_key_2_1": "componentB value 2.1",
                       "componentB_key_2_2": "componentB value 2.2"},
 "dict_key_2_2": "common value 2.2",
 "list_element_0": "componentB value 1.1"}
```

Overwrite the value of a single element of a parameter of nested data type

In the next example we overwrite the value of a list element and the value of a dictionary key.

Again we change the content of file `componentB.2.jsonp`:

```
{
  // import of componentB parameters
  "[import]" : "./componentB.jsonp",
  //
  ${componentB_param_1}[0] : "componentB value 1.1 (new)",
  ${common_param_2}['common_key_2_1'] : "common value 2.1 (new)"
}
```

The dollar operator syntax at the left hand side of the colon is the same than previously used on the right hand side. The entire expression at the left hand side of the colon must *not* be encapsulated in quotes in this case.

Outcome:

The single elements of the list and the dictionary are updated, all other elements are unchanged.

```
{"common_param_1": ["common value 1.1", "common value 1.2"],
 "common_param_2": {"common_key_2_1": "common value 2.1 (new)",
                   "common_key_2_2": "common value 2.2"},
 "componentB_param_1": ["componentB value 1.1 (new)", "componentB value 1.2"],
 "componentB_param_2": {"componentB_key_2_1": "componentB value 2.1",
                       "componentB_key_2_2": "componentB value 2.2"}}
```

Add an element to a parameter of nested data type

Adding further elements to an already existing list is not possible in JSON! But it is possible to add keys to an already existing dictionary.

The following example extends the dictionary `common_param_2` by an additional key `common_key_2_3` :

```
{
    // import of componentB parameters
    "[import]" : "./componentB.jsonp",
    //
    ${common_param_2}['common_key_2_3'] : "common value 2.3"
}
```

Outcome:

```
{'common_param_1': ['common value 1.1', 'common value 1.2'],
'common_param_2': {'common_key_2_1': 'common value 2.1',
                  'common_key_2_2': 'common value 2.2',
                  'common_key_2_3': 'common value 2.3'},
'componentB_param_1': ['componentB value 1.1', 'componentB value 1.2'],
'componentB_param_2': {'componentB_key_2_1': 'componentB value 2.1',
                      'componentB_key_2_2': 'componentB value 2.2'}}
```

Dictionary keys and indices as parameter

In all code examples above the indices of lists and the key names of dictionaries have been hard coded strings. It is also possible to use parameters:

```
{
    "index1" : 0,
    "index2" : 1,
    "key1" : "keyA",
    "key2" : "keyB",
    "testlist" : ["A", "B"],
    "testdict" : {"keyA" : "A", "keyB" : "B"},
    "tmp1" : ${testlist}[${index1}],
    "tmp2" : ${testdict}[${key1}],
    ${testlist}[${index1}] : ${testlist}[${index2}],
    ${testdict}[${key1}] : ${testdict}[${key2}],
    ${testlist}[${index2}] : ${tmp1},
    ${testdict}[${key2}] : ${tmp2}
}
```

Outcome:

```
{'index1': 0,
'index2': 1,
'key1': 'keyA',
'key2': 'keyB',
'testdict': {'keyA': 'B', 'keyB': 'A'},
'testlist': ['B', 'A'],
'tmp1': 'A',
'tmp2': 'A'}
```

Meaning of single quotes in square brackets

Single quotes are used to convert the content inside to a string.

- In case of the parameter `param` is of type string, the expressions `[$param]` and `['$param']` have the same outcome: The content inside the square brackets is a string. The single quotes have no meaning in this case (because the parameter is already of type string).
- In case of the parameter `param` is of type integer, the quotes in `['$param']` convert the integer value to a string. Without the quotes (`[$param]`), the content inside the square brackets is an integer.

In the context of **JsonPreprocessor** JSON files, only strings and integers are expected to be inside square brackets (except the brackets are used to define a list). Other data types are not supported here.

Whether a string or an integer is expected, depends on the data type of the parameter, the square bracket expression belongs to. Dictionaries require a string (a key name), lists require an integer (an index). Deviations will cause an error.

Summarized the following combinations are valid (on both the left hand side of the colon and the right hand side of the colon):

```

${listparam}[$intparam]
${listparam}[1]
${dictparam}['${intparam}']
${dictparam}[$stringparam]
${dictparam}['${stringparam}']
${dictparam}['keyname']

```

Use of a common dictionary

The last example in this section covers the following use case:

- We have several JSON files, each for a certain purpose within a project (e.g. for every feature of this project a separate JSON file).
- They belong together and therefore they are all imported into a main JSON file that is the file that is handed over to the **JsonPreprocessor**.
- Every imported JSON file introduces a certain bunch of parameters. All parameters need to be a part of a common dictionary.
- Outcome is that finally only one single dictionary is used to access the parameters from all JSON files imported in the main JSON file.

To realize this, it is necessary to separate the initialization of the dictionary from all positions where keys are added to this dictionary.

These are the JSON files:

- `project.jsonp`

```
{  
    // initialization  
    "project_values" : {},  
    //  
    // add some common values  
    ${project_values}['common_project_param_1'] : "common project value 1",  
    ${project_values}['common_project_param_2'] : "common project value 2",  
    //  
    // import feature parameters  
    "[import]" : "./featureA.jsonp",  
    "[import]" : "./featureB.jsonp",  
    "[import]" : "./featureC.jsonp"  
}
```

- `featureA.jsonp`

```
{  
    // parameters required for feature A  
    ${project_values}['featureA_params'] : {},  
    ${project_values}['featureA_params']['featureA_param_1'] : "featureA param 1 value",  
    ${project_values}['featureA_params']['featureA_param_2'] : "featureA param 2 value"  
}
```

- `featureB.jsonp`

```
{  
    // parameters required for feature B  
    ${project_values}['featureB_params'] : {},  
    ${project_values}['featureB_params']['featureB_param_1'] : "featureB param 1 value",  
    ${project_values}['featureB_params']['featureB_param_2'] : "featureB param 2 value"  
}
```

- `featureC.jsonp`

```
{  
    // parameters required for feature C  
    ${project_values}['featureC_params'] : {},  
    ${project_values}['featureC_params']['featureC_param_1'] : "featureC param 1 value",  
    ${project_values}['featureC_params']['featureC_param_2'] : "featureC param 2 value"  
}
```

Explanation:

Every `feature*.jsonp` file refer to the dictionary initialized within `project.jsonp`.

Important is that the initialization `"project_values" : {}`, happens at top level (`project.jsonp`), and not within the imported files (`feature*.jsonp`), otherwise follow up initializations would delete previously added keys of this dictionary!

Outcome:

```
{'project_values': {'common_project_param_1': 'common project value 1',
                    'common_project_param_2': 'common project value 2',
                    'featureA_params': {'featureA_param_1': 'featureA param 1 value',
                                         'featureA_param_2': 'featureA param 2 value'},
                    'featureB_params': {'featureB_param_1': 'featureB param 1 value',
                                         'featureB_param_2': 'featureB param 2 value'},
                    'featureC_params': {'featureC_param_1': 'featureC param 1 value',
                                         'featureC_param_2': 'featureC param 2 value'}}}
```

3.6 dotdict notation

Up to now we have accessed dictionary keys in this way (standard notation):

```
 ${dictionary}['key']['sub_key']
```

Additionally to this standard notation, the **JsonPreprocessor** supports the so called *dotdict* notation where keys are handled as attributes:

```
 ${dictionary.key.sub_key}
```

In standard notation keys are encapsulated in square brackets and all together is placed *outside* the curly brackets. In dotdict notation the dictionary name and the keys are separated by dots from each other. All together is placed *inside* the curly brackets.

In standard notation key names are allowed to contain dots:

```
 ${dictionary['key']['sub.key']}
```

In dotdict notation this would cause ambiguities:

```
 ${dictionary.key.sub.key}
```

Therefore it is not possible to implement in this way! In case you need to have dots inside key names, you must use the standard notation. We recommend to prefer underlines as separator - like done in the examples in this document.

Do you really need dots inside key names?

Please keep in mind: The dotdict notation is a reduced one. Because of parts are missing (e.g. the single quotes around key names), the outcome can be code that is really hard to capture.

In the following example we create a composite data structure and demonstrate how to access single elements in both notations.

- JSON file:

```
{
    // composite data structure
    "params" : [{"dict_1_key_1" : "dict_1_key_1 value",
                 "dict_1_key_2" : ["dict_1_key_2 value 1", "dict_1_key_2 value 2"]},
                //
                {"dict_2_key_1" : "dict_2_key_1 value",
                 "dict_2_key_2" : {"dict_2_A_key_1" : "dict_2_A_key_1 value",
                                  "dict_2_A_key_2" : ["dict_2_A_key_2 value 1", ↪
                                         "dict_2_A_key_2 value 2"]}}],
    //
    // access to single elements of composite data structure
    //
    // a) standard notation
    "dict_1_key_2_value_2_standard" : ${params}[0]['dict_1_key_2'][1],
    // b) dotdict notation
    "dict_1_key_2_value_2_dotdict" : ${params.0.dict_1_key_2.1},
    //
    // c) standard notation
    "dict_2_A_key_2_value_2_standard" : ${params}[1]['dict_2_key_2']['dict_2_A_key_2'][1]
    // d) dotdict notation
    "dict_2_A_key_2_value_2_dotdict" : ${params.1.dict_2_key_2.dict_2_A_key_2.1}
}
```

Outcome:

In case of the composite data structure becomes more and more nested (and if also the key names contain numbers), understanding the expressions (like `${params.1.dict_2_key_2.dict_2_A_key_2.1}`) becomes more and more challenging!

```
{'dict_1_key_2_value_2_standard': 'dict_1_key_2 value 2',
 'dict_2_A_key_2_value_2_standard': 'dict_2_A_key_2 value 2',
 'params': [{{'dict_1_key_1': 'dict_1_key_1 value',
   'dict_1_key_2': ['dict_1_key_2 value 1', 'dict_1_key_2 value 2']},
   {'dict_2_key_1': 'dict_2_key_1 value',
   'dict_2_key_2': {'dict_2_A_key_1': 'dict_2_A_key_1 value',
     'dict_2_A_key_2': ['dict_2_A_key_2 value 1',
       'dict_2_A_key_2 value 2']}}}]}
```

3.7 Substitution of dollar operator expressions

Like shown in previous examples, existing parameters are accessed by a dollar operator and a pair of curly brackets (`${...}`) with the parameter name inside.

We discussed use cases, where the entire expression of the left hand side of the colon and also the entire expression of the right hand side of the colon have been either hard coded strings, encapsulated in quotes, or dollar operator expressions, not encapsulated in quotes.

Also a mix of both is possible!

Outcome is a hard coded string, encapsulated in quotes, with parts represented by one or more than one dollar operator expression. This can be used to create new content very dynamically: On the right hand side of the colon new string values can be created; on the left hand side of the colon this mechanism generates dynamic parameter names.

- JSON file:

```
{
    "project"      : "Test",
    "version"      : 1.23,
    "item_number"  : 1,
    "component"    : "componentA",
    //
    "${project}_message_${item_number}" : "Component '${component}' has version ${version}"
}
```

Outcome:

```
{'component': 'componentA',
'item_number': 1,
'project': 'Test',
'Test_message_1': "Component 'componentA' has version 1.23",
'version': 1.23}
```

We recommend to use simple data types for this kind of substitution only!

But nevertheless, on the right hand side of the colon also composite data types are possible. Here it might makes sense to use them. But on the left hand side of the colon only simple data types are allowed for substitution. Because it makes no sense to create parameter names based on composite data types. Most probably this would cause invalid key names.

To demonstrate this, we change the JSON file to:

```
{
    "component"      : "componentA",
    "composite_data" : ["AB", 12, True, null, {"kA" : "kAval", "kB" : "kBval"}],
    //
    "test_parameter" : "Key values of component '${component}' are: ${composite_data}"
}
```

Outcome:

```
{'component': 'componentA',
'composite_data': ['AB', 12, True, None, {'kA': 'kAval', 'kB': 'kBval'}],
'test_parameter': "Key values of component 'componentA' are: ['AB', 12, True, "
                  "None, {'kA': 'kAval', 'kB': 'kBval'}]"}
```

CHAPTER 3. THE JSONP FORMAT3.7. SUBSTITUTION OF DOLLAR OPERATOR EXPRESSIONS

Now we try to do the same on the left hand side of the colon:

```
{  
    "param" : "string value",  
    "composite_data" : ["AB", 12, True, null, {"kA" : "kAval", "kB" : "kBval"}],  
    //  
    "test_parameter_{$composite_data}" : ${param}  
}
```

If this would be allowed, the last line would cause a parameter with this name:

```
"test_parameter_['AB', 12, True, None, {'kA': 'kAval', 'kB': 'kBval'}]"
```

And this is absolutely not a valid name!

To prevent the users from generating invalid parameter names, the **JsonPreprocessor** throws an error:

```
Error: 'Found expression '....' with at least one parameter of composite data type ...  
Because of this expression is the name of a parameter, only simple data types are ↵  
↳ allowed to be substituted inside.'!"
```

3.8 Overwriting vs. substitution

In this section we take a deeper look at the syntax difference between overwriting and substitution explained above.

1. Overwriting:

With `${param2} : ${param1}` the initial value `"XYZ"` of parameter `param2` is overwritten with the value `"ABC"` of parameter `param1`.

```
{
  "param1" : "ABC",
  "param2" : "XYZ",
  //
  ${param2} : ${param1},
}
```

Result:

```
{"param1" : "ABC",
 "param2" : "ABC"}
```

2. Substitution:

With `"${param2}" : ${param1}` a new parameter with name `XYZ` (the value of `param2`) and value `"ABC"` is created.

```
{
  "param1" : "ABC",
  "param2" : "XYZ",
  //
  "${param2}" : ${param1},
}
```

Result:

```
{"param1" : "ABC",
 "param2" : "XYZ",
 "XYZ" : "ABC"}
```

3.9 Implicite creation of dictionaries

Up to now we have discussed two different ways of creating nested dictionaries.

The first one is “on the fly”, like:

```
{
    "project_values" : {"keyA" : "keyA value",
                        "keyB" : {"keyB1" : "keyB1 value",
                                  "keyB2" : {"keyB21" : "keyB21 value",
                                             "keyB22" : "keyB22 value"}{}}
}
```

In case of it is required to split the definition into several files, we have to add keys (and also the initialization) line by line:

```
{
    "project_values" : {},
    ${project_values}['keyA'] : "keyA value",
    ${project_values}['keyB'] : {},
    ${project_values}['keyB']['keyB1'] : "keyB1 value",
    ${project_values}['keyB']['keyB2'] : {},
    ${project_values}['keyB']['keyB2']['keyB21'] : "keyB21 value",
    ${project_values}['keyB']['keyB2']['keyB22'] : "keyB22 value"
}
```

The result will be the same as in the previous example.

It can be seen now that this way of creating nested dictionaries is rather long winded, because every initialization of a dictionary requires a separate line of code (at every level).

To shorten the code, the **JsonPreprocessor** supports an implicite creation of dictionaries.

This is the resulting code in standard notation:

```
{
    ${project_values}['keyA'] : "keyA value",
    ${project_values}['keyB']['keyB1'] : "keyB1 value",
    ${project_values}['keyB']['keyB2']['keyB21'] : "keyB21 value",
    ${project_values}['keyB']['keyB2']['keyB22'] : "keyB22 value"
}
```

And the same in dotdict notation (with precondition, that no key name contains a dot):

```
{
    ${project_values.keyA} : "keyA value",
    ${project_values.keyB.keyB1} : "keyB1 value",
    ${project_values.keyB.keyB2.keyB21} : "keyB21 value",
    ${project_values.keyB.keyB2.keyB22} : "keyB22 value"
}
```

Caution:

We urgently recommend *not* to mixup both styles in one line of code. In case of keys contain a list and also numerical indices are involved, we recommend to prefer the standard notation.

Please be aware of: In case of a missing level in between an expression like

```
{
    ${project_values.keyB.keyB2.keyB22} : "keyB22 value"
}
```

you will *not* get an error message! The entire data structure will be created implicitly. The impact is that this method is very susceptible to typing mistakes.

The implicite creation of data structures does not work with lists! In case you use a list index out of range, you will get a corresponding error message.

Key names

The implicit creation of data structures is only possible with *hard coded* key names. Parameters are not supported.

Example:

```
{
    "paramA" : "ABC",
    "subKey" : "ABC",
    ${testdict.subKey.subKey.paramA} : "DEF"
}
```

All sub key levels within the expression `${testdict.subKey.subKey.paramA}` are interpreted as hard coded strings, even in case of parameters with the same name do exist.

For example: The name of the implicitly created key at bottom level is `"paramA"`, and not the value `"ABC"` of the parameter with the same name (`"paramA"`).

Therefore the outcome is:

```
{"paramA": "ABC", "subKey": "ABC", "testdict": {"subKey": {"subKey": {"paramA": "DEF"}}}}
```

Reference to existing keys

It is possible to use parameters to refer to *already existing* keys.

```
{
    // data structure created implicitly
    ${testdict.subKey_1.subKey_2.subKey_3} : "ABC",

    // string parameter with name of an existing key
    "keyName_3" : "subKey_3",

    // parameter used to refer to an existing key
    ${testdict.subKey_1.subKey_2.${keyName_3}} : "XYZ"
}
```

Outcome:

```
{"keyName_3": "subKey_3",
 "testdict": {"subKey_1": {"subKey_2": {"subKey_3": "XYZ"}}}}
```

Parameters cannot be used to create new keys.

```
{
    // data structure created implicitly
    ${testdict.subKey_1.subKey_2.subKey_3} : "ABC",

    // string parameter with name of a not existing key
    "keyName_4" : "subKey_4",

    // usage of keyName_4 is not possible here
    ${testdict.subKey_1.subKey_2.subKey_3.${keyName_4}} : "XYZ"
}
```

Outcome is the following error:

```
"The implicit creation of data structures based on nested parameter is not supported ..."
```

The same error will happen in case of the standard notation is used:

```
{
    // usage of keyName_4 is not possible here
    ${testdict}['subKey_1']['subKey_2']['subKey_3'][$keyName_4] : "XYZ"
}
```

CHAPTER 4. CJSONPREPROCESSOR.PY

Chapter 4

CJsonPreprocessor.py

4.1 Class: CSyntaxType

Imported by:

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

4.2 Class: CNameMangling

Imported by:

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

4.3 Class: CPythonJSONDecoder

Imported by:

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

Extends the JSON syntax by the Python keywords True, False and None.

Arguments:

- json.JSONDecoder
 - / *Type:* object /
 - Decoder object provided by json.loads

4.3.1 Method: custom_scan_once

4.4 Class: CJsonPreprocessor

Imported by:

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

CJsonPreprocessor extends the JSON syntax by the following features:

- Allow c/c++-style comments within JSON files
- Allow to import JSON files into JSON files
- Allow to define and use parameters within JSON files
- Allow Python keywords True, False and None

4.4.1 Method: getVersion

Returns the version of JsonPreprocessor as string.

4.4.2 Method: getVersionDate

Returns the version date of JsonPreprocessor as string.

4.4.3 Method: jsonLoad

This method is the entry point of JsonPreprocessor.

`jsonLoad` loads the JSON file, preprocesses it and returns the preprocessed result as Python dictionary.

Arguments:

- `jFile`
/ *Condition*: required / *Type*: str /
Path and name of main JSON file. The path can be absolute or relative and is also allowed to contain environment variables.

Returns:

- `oJson`
/ *Type*: dict /
Preprocessed JSON file(s) as Python dictionary

4.4.4 Method: jsonDump

This method writes the content of a Python dictionary to a file in JSON format and returns a normalized path to this JSON file.

Arguments:

- `oJson`
/ *Condition*: required / *Type*: dict /
- `outFile (string)`
/ *Condition*: required / *Type*: str /
Path and name of the JSON output file. The path can be absolute or relative and is also allowed to contain environment variables.

Returns:

- `outFile (string)`
/ *Type*: str /
Normalized path and name of the JSON output file.

CHAPTER 5. APPENDIX

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	JsonPreprocessor
Version	0.4.0
Date	15.03.2024
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 6. HISTORY

Chapter 6

History

0.1.0	01/2022
<i>Initial version</i>	
0.1.4	09/2022
<i>Documentation updated</i>	
0.2.3	05/2023
<i>dotdict format added</i>	
0.2.4	06/2023
<i>Maintenance of dotdict format and log output</i>	
0.3.0	09/2023
<ul style="list-style-type: none"> - Implicit creation of data structures - Dotdict feature bug fixing - Update nested parameters handling in key name and value - Nested parameter feature bug fixing - Nested parameters substitution and overwriting improvement - Jsonp file path computation improvement 	
0.3.1	12/2023
<ul style="list-style-type: none"> - Add jsonDump method to write a file in JSON format - Improve nested parameter format - Improve error message log - Fix bugs of data structures implicitly - Improve index handling together with nested parameters 	
0.3.3	01/2024
<ul style="list-style-type: none"> - Some bugs fixed in implicitly created data structures - Improved index handling together with nested parameters - Improved format of nested parameters; improved error messages - Added getVersion and getVersionDate methods to get current version and the date of the version 	
0.4.0	03/2024
<ul style="list-style-type: none"> - Optimized regular expression patterns - Improved duplicated parameters handling - Added mechanism to prevent Python application freeze - Removed globals scope out of all exec method executions - Optimized errors handling while loading nested parameters - Fixed bugs 	

JsonPreprocessor.pdf*Created at 10.04.2024 - 12:33:41**by GenPackageDoc v. 0.41.1*

8.5 RobotFramework_TestsuitesManagement

RobotFramework_TestsuitesManagement

v. 0.7.6

Mai Dinh Nam Son

01.04.2024

Contents

1	Introduction	1
2	Description	2
2.1	Meaning of "Test Suites Management"	2
2.2	Content of configuration files	3
2.3	Access to configuration files	6
2.4	Activation of "Test Suites Management"	7
2.5	Variants selection	8
2.6	Local configuration	9
2.7	Priority of configuration parameters	10
2.8	Nested configuration files	11
2.9	Overwritten parameters	12
3	The JSONP format	13
3.1	Standard JSON format	13
3.2	Boolean and null values	14
3.3	Comments	15
3.4	Import of JSON files	16
3.5	Overwrite parameters	17
3.6	dotdict notation	26
3.7	Substitution of dollar operator expressions	28
3.8	Overwriting vs. substitution	30
3.9	Implicite creation of dictionaries	31
4	CConfig.py	33
4.1	Function: bundle_version	33
4.2	Class: CConfig	33
4.2.1	Method: loadCfg	34
4.2.2	Method: verifyVersion	34
4.2.3	Method: bValidateMinVersion	34
4.2.4	Method: bValidateMaxVersion	35
4.2.5	Method: bValidateSubVersion	35
4.2.6	Method: tupleVersion	35
4.2.7	Method: versioncontrol_error	36
5	COnFailureHandle.py	37
5.1	Class: COnFailureHandle	37
5.1.1	Method: is_noney	37

CONTENTS	CONTENTS
6 CSetup.py	38
6.1 Class: CSetupKeywords	38
6.1.1 Keyword: testsuite_setup	38
6.1.2 Keyword: testsuite_teardown	38
6.1.3 Keyword: testcase_setup	38
6.1.4 Keyword: testcase_teardown	39
6.2 Class: CGeneralKeywords	39
6.2.1 Keyword: get_config	39
6.2.2 Keyword: load_json	39
7 CStruct.py	40
7.1 Class: CStruct	40
8 Event.py	41
8.1 Class: Event	41
8.1.1 Method: trigger	41
9 ScopeEvent.py	42
9.1 Class: ScopeEvent	42
9.1.1 Method: trigger	42
9.2 Class: ScopeStart	42
9.3 Class: ScopeEnd	42
10 __init__.py	43
10.1 Function: on	43
10.2 Function: dispatch	43
10.3 Function: register_event	43
11 LibListener.py	44
11.1 Class: LibListener	44
12 __init__.py	45
12.1 Class: RobotFramework_TestsuitesManagement	45
12.1.1 Method: run_keyword	45
12.1.2 Method: get_keyword_tags	45
12.1.3 Method: get_keyword_documentation	45
12.1.4 Method: failure_occurred	45
12.2 Class: CTestsuitesCfg	45
13 Appendix	46
14 History	47

CHAPTER 1. INTRODUCTION

Chapter 1

Introduction

The **RobotFramework_TestsuitesManagement** enables users to define dynamic configuration values within separate configuration files in JSON format.

These configuration values are available during test execution - but under certain conditions that can be defined by the user (e.g. to realize a variant handling). This means: Not all parameter values are available during test execution - only the ones that belong to the current test scenario.

To realize this, the **RobotFramework_TestsuitesManagement** provides the following features:

1. Split all possible configuration values into several JSON configuration files, with every configuration file contains a specific set of values for configuration parameter
2. Use nested imports of JSON configuration files
3. Follow up definitions in configuration files overwrite previous definitions (of the same parameter)
4. Select between several criteria to let the Robot Framework use a certain JSON configuration file

How to install

The **RobotFramework_TestsuitesManagement** can be installed in two different ways: via PyPi (recommended for users) and via GitHub (recommended for developers).

Installation details can be found in the [README](#).

Further links

For self-study a tutorial is available containing lots of examples. Here you find the rendered [tutorial documentation](#).

For the development environment **VSCodium** an extension is available to support the features of the **RobotFramework_TestsuitesManagement**: [vscode-jsonp](#). This extension adapts e.g. the syntax highlighting of the editor.

CHAPTER 2. DESCRIPTION

Chapter 2

Description

2.1 Meaning of "Test Suites Management"

In the scope of the Robot Framework a test suite is either a single robot file containing one or more test cases, or a set of several robot files.

Usually all test cases of a test suite run under the same conditions - but these conditions may be different. For example the same test case is used to test several different variants of a system under test. Every variant requires individual values for certain configuration parameters.

Tests are carried out at several test benches. All test benches have different hardware configurations. Also the different test benches may require individual values for configuration parameters used in the tests.

Therefore the same tests have to run under different conditions!

The Robot Framework provides several places to define parameters: robot files, resource files, parameter files. But these parameters are fixed. Therefore we need a more dynamic way of accessing parameters. And we postulate the following: When switching between tests of several variants and test executions on several test benches, no changes shall be required within the test code.

The outcome is that another position has to be introduced to store values for variant and test bench specific parameters. And a possibility has to be provided to dynamically make either the one or the other set of values available during the execution of tests - depending on outer circumstances like "*which variant?*" and "*which test bench?*". Those dynamic configuration values are stored within separate configuration files in JSON format and the **RobotFramework.TestsuitesManagement** makes the values available globally during the test execution.

Two different kinds of JSON configuration files are involved:

1. *parameter configuration files*

These configuration files contain all parameter definitions (can be more than one configuration file in a project)

2. *variant configuration file*

This is a single configuration file containing the mapping between the several parameter configuration files and a name (usually the name of a variant). This name can be used in command line to select a certain parameter configuration file containing the values for this variant.

Background: It's easier simply to use a name for referencing a certain variant instead of having the need always to mention the path and name of a configuration file.

To realize a concrete test suites management for your project, you need to

- identify the parameters that are variant specific, depending on the number of variants in your project,
- identify the parameters that are test bench specific, depending on the number of test benches in your project,
- identify the parameters that are both: variant specific and test bench specific,
- identify the parameters that have the same value in all variants and test benches.

CHAPTER 2. DESCRIPTION2.2. CONTENT OF CONFIGURATION FILES

After this

- for every set of parameters (variant specific and bench specific) you have to introduce a certain parameter configuration file,
- in the variant configuration file you have to define for every variant a variant name together with the path to the corresponding parameter configuration file.

Basically all configuration files of the **RobotFramework_TestsuitesManagement** are implemented in JSON format. This format is extended by some useful features like code comments and imports (nested configuration files). This is explained in more detail in the following chapters. These features cause deviations from standard JSON format. To give applications like editors or syntax checkers a chance to handle these deviations (without invalid findings), all JSON configurations files of the **RobotFramework_TestsuitesManagement** have the extension `.jsonp`, instead of `.json`.

The content of the configuration files is described in the next section.

2.2 Content of configuration files

1. variant configuration file

This file configures the access to all variant dependent `robot_config*.jsonp` files.

```
{
  "default": {
    "name": "robot_execution_config.jsonp",
    "path": ".../config/"
  },
  "variant_1": {
    "name": "robot_config_variant_1.jsonp",
    "path": ".../config/"
  },
  "variant_2": {
    "name": "robot_config_variant_2.jsonp",
    "path": ".../config/"
  },
  "variant_3": {
    "name": "robot_config_variant_3.jsonp",
    "path": ".../config/"
  }
}
```

The example above contains definitions for three variants with names:

`variant_1`, `variant_2` and `variant_3`. Additionally a variant named `default` is defined. This default configuration becomes active in case of no certain variant name is provided when the test suite is being executed.

Another aspect is important: the **three dots**. The path to the `robot_config*.jsonp` files depends on the test file location. A different number of `../` is required dependent on the directory depth of the test case location.

Therefore we use here three dots to tell the **RobotFramework_TestsuitesManagement** to search from the test file location up till the `robot_config*.jsonp` files are found:

```
./config/robot_config.jsonp
../config/robot_config.jsonp
../../config/robot_config.jsonp
../../../config/robot_config.jsonp
```

and so on.

Hint: The paths to the `robot_config*.jsonp` files are relative to the position of the test suite - **and not relative to the position of the mapping file in which they are defined!** You are free to move your test suites one or more level up or down in the file system, but using the three dots notation enables you to let the position of the `config` folder unchanged.

It is of course still possible to use the standard notation for relative paths:

```
"path": "./config/"
```

CHAPTER 2. DESCRIPTION2.2. CONTENT OF CONFIGURATION FILES**2. parameter configuration files**

In these configuration files all parameters are defined, that shall be available globally during test execution.

Some parameters are required. Optionally the user can add own ones. The following example shows the smallest version of a parameter configuration file containing only the most important parameters. This version is a default version and part of the **RobotFramework_TestsuitesManagement** installation.

```
{
    "WelcomeString" : "Hello... Robot Framework is running now!",
    "Maximum_version" : "1.0.0",
    "Minimum_version" : "0.6.0",
    "Project" : "RobotFramework Testsuites",
    "TargetName" : "Device_01"
}
```

`Project`, `WelcomeString` and `TargetName` are simple strings that can be used anyhow. `Maximum_version` and `Minimum_version` are part of a version control mechanism: In case of the version of the currently installed software is outside the range between `Minimum_version` and `Maximum_version`, the test execution stops with an error message.

What is the meaning of "currently installed software"?

- The first possibility is that the **RobotFramework_TestsuitesManagement** runs stand-alone, that means, it is not part of a larger bundle (like the RobotFramework AIO). The installation from PyPi or GitHub causes such a stand-alone installation. In this case the component version of the **RobotFramework_TestsuitesManagement** itself is used for a version control against `Minimum_version` and `Maximum_version`.
- The second possibility is that the **RobotFramework_TestsuitesManagement** runs as part of the RobotFramework AIO. In this case the version of the entire RobotFramework AIO is used for a version control instead.

The version control mechanism is optional. In case you do not need to have your tests under version control, you can set the versions to the value `null`.

```
"Maximum_version" : null,
"Minimum_version" : null,
```

As an alternative it is also possible to remove `Minimum_version` and `Maximum_version` completely.

In case you define only one single version number, only this version number is considered. The following combination makes sure, that the installed software at least is of version 0.6.0, but there is no upper version limit:

```
"Maximum_version" : null,
"Minimum_version" : "0.6.0",
```

Hint: The parameters are keys of an internal configuration dictionary. They have to be accessed in the following way:

```
Log    Maximum_version : ${CONFIG}[Maximum_version]
Log    Project : ${CONFIG}[Project]
```

The following example is an extended version of a configuration file containing also some user defined parameters.

```
{
    "WelcomeString" : "Hello... Robot Framework is running now!",
    "Maximum_version" : "1.0.0",
    "Minimum_version" : "0.6.0",
    "Project" : "RobotFramework Testsuites",
    "TargetName" : "Device_01",
    "params": {
        // global parameters
        "global" : {
            "param1" : "ABC",
            "param2" : 25
        }
    }
}
```

CHAPTER 2. DESCRIPTION2.2. CONTENT OF CONFIGURATION FILES

User defined parameters have to be placed inside `params:global`. The intermediate level `global` is introduced to enable further parameter scopes than `global` in future.

All user defined parameters have the scope `params:global` per default. Therefore they can be accessed directly:

```
Log    param1 : ${param1}
```

And another feature can be seen in the example above:

In the context of the **RobotFramework_TestsuitesManagement** the JSON format is an extended one. Deviating from JSON standard it is possible to comment out lines with starting them with a double slash `//`. This allows to add explanations about the meaning of the defined parameters already within the JSON file.

2.3 Access to configuration files

With an installed **RobotFramework.TestsuitesManagement** every test execution requires a configuration - that is the accessibility of a configuration file in JSON format. The **RobotFramework.TestsuitesManagement** provides four different possibilities - also called *level* - to realize such an access. These possibilities are sorted and the **RobotFramework.TestsuitesManagement** tries to access the configuration file in a certain order: Level 1 has the highest priority and level 4 has the lowest priority.

Level 1

Path and name of a parameter configuration file is provided in command line of the Robot Framework.

Level 2 (recommended)

The name of the variant is provided in command line of the Robot Framework.

This level requires that a variant configuration file is passed to the suite setup of the **RobotFramework.TestsuitesManagement**.

Level 2 includes the automated selection of a default variant (in case of no variant name is provided in command line). Also this default variant has to be defined within the variant configuration file.

Level 3

The **RobotFramework.TestsuitesManagement** searches for parameter configuration files within a folder `config` in current test suite folder. In case of such a folder exists and parameter configuration files are inside, they will be used.

Level 4 (unwanted, fallback solution only)

The **RobotFramework.TestsuitesManagement** uses the default configuration file that is part of the installation.

Summary

- With highest priority a parameter configuration file provided in command line, is considered - even in case of also other configuration files (level 2 - level 4) are available.
- If a parameter configuration file is not provided in command line, but a variant name, then the configuration belonging to this variant, is loaded - even in case of also other configuration files (level 3 - level 4) are available.
- If nothing is specified in command line, then the **RobotFramework.TestsuitesManagement** tries to find parameter configuration files within a `config` folder and take them if available - even in case of also the level 4 configuration file is available.
- In case of the user does not provide any information about parameter configuration files to use, the **RobotFramework.TestsuitesManagement** loads the default configuration from installation folder (fallback solution; level 4).

In this context two aspects are important to know for users:

- Which parameter configuration file is selected for the test execution?*
To answer this question the log file contains the path and the name of the selected parameter configuration file.
- For which reason is this parameter configuration file selected?*
To answer this question the log file also contains the level number. The level number indicates the reason.

With these log file entries the test execution is clearly understandable, traceable and scales for huge test suites.

Why is level 2 the recommended one?

Level 2 is the most flexible and extensible solution. Because the robot files contain a link to a variants configuration file, the possible sets of parameter values can already be taken out of the code.

The values selected by level 1, you only see in the log files, but not in the code, because the selection happens in command line only.

Level 3 has a rather strong binding between robot files and configuration files. If you start the test implementation based on level 3 and after this want to have a variant handling, then you have to switch from level 3 to level 2 - and this causes effort in implementation.

Whereas if you start with level 2 immediately and need to consider another set of configuration values for the same tests, then you only have to add another parameter configuration file and another entry in the variants configuration file, without changing any test implementation.

CHAPTER 2. DESCRIPTION2.4. ACTIVATION OF "TEST SUITES MANAGEMENT"

We strongly recommend not to mix up several different configuration levels in one project!

2.4 Activation of "Test Suites Management"

To activate the test suites management you have to import the **RobotFramework_TestsuitesManagement** library in the following way:

```
Library    RobotFramework_TestsuitesManagement    WITH NAME    tm
```

We recommend to use the `WITH NAME` option to shorten the robot code a little bit.

The next step is to call the `testsuite_setup` of the **RobotFramework_TestsuitesManagement** within the `Suite Setup` of your test:

```
Suite Setup    tm.testsuite_setup
```

As long as you

- do not provide a parameter configuration file in command line when executing the test suite (level 1),
- do not provide a variants configuration file as parameter of the `testsuite_setup` (level 2),
- do not have a `config` folder containing parameter configuration files in your test suites folder (level 3),

the **RobotFramework_TestsuitesManagement** falls back to the default configuration (level 4).

In case you want to realize a variant handling you have to provide the path and the name of a variants configuration file to the `testsuite_setup`:

```
Suite Setup    tm.testsuite_setup    ./config/exercise_variants.jsonp
```

To ease the analysis of a test execution, the log file contains informations about the selected level and the path and the name of the used configuration file, for example:

```
Running with configuration level: 2
CfgFile Path: ./config/exercise_config.jsonp
```

Please consider: The `testsuite_setup` requires a variants configuration file (in the example above: `exercise_variants.jsonp`) - whereas the log file contains the resulting parameter configuration file (in the example above: `exercise_config.jsonp`), that is selected depending on the name of the variant provided in command line of the Robot Framework.

2.5 Variants selection

In a previous section the level concept for configuration files has been explained. This section contains corresponding code examples.

1. Selection of a certain parameter configuration file in command line

```
--variable config_file:"(path to parameter configuration file)"
```

2. Selection of a certain variant per name in command line

```
--variable variant:"(variant name)"
```

3. Parameter configuration taken from `config` folder

This `config` folder has to be placed in the same folder than the test suites.

Parameter configuration files within this folder are considered under two different conditions:

- The configuration file has the name `robot_config.jsonp`. That is a fix name predefined by the **Robot-Framework.TestsuitesManagement**.
- The configuration file has the same name than a robot file inside the test suites folder, e.g.:
 - Name of test suite file: `example.robot`
 - Path and name of corresponding parameter configuration file: `./config/example.jsonp`

With this rule it is possible to give every test suite in a certain folder an own individual configuration.

2.6 Local configuration

It might be required to execute tests on several different test benches with every test bench has it's own individual hardware that might require configuration parameter values that are test bench specific. This can be related to common configuration parameters and also to parameters that are variant specific. In the second case a configuration parameter is both variant specific *and* test bench specific.

The *local configuration* feature of the **RobotFramework_TestsuitesManagement** provides the possibility to define test bench specific configuration parameter values.

The meaning of *local* in this context is: placed on a certain test bench - and valid for this bench only.

Also this local configuration is based on configuration files in JSON format. These files are the last ones that are considered when the configuration is loaded. The outcome is that it is possible to define default values for test bench specific parameters in other configuration files - to be also test bench independent. And it is possible to use the local configuration to overwrite these default values with values that are specific for a certain test bench.

Important:

- Local configuration files are fragments only - and not a full configuration! Even so they need to follow the JSON syntax rules. This means, at least they have to start with an opening curly bracket and they have to end with a closing curly bracket.
- Local configuration files must not contain the mandatory top level parameters like the `WelcomeString` and others.

Using the local configuration feature is an option and the **RobotFramework_TestsuitesManagement** provides two ways to realize it:

1. per command line

Path and name of the local parameter configuration file is provided in command line of the Robot Framework with the following syntax:

```
--variable local_config:"(path to local configuration file)"
```

2. per environment variable

An environment variable named `ROBOT_LOCAL_CONFIG` exists and contains path and name of a local parameter configuration file.

The user has to create this environment variable!

This mechanism allows a user - without any command line extensions - automatically to refer on every test bench to an individual local configuration, simply by giving on every test bench this environment variable an individual value.

The command line has a higher priority than the environment variable. If both is available the local configuration is taken from command line.

Recommendation: *To avoid an accidental overwriting of local configuration files in version control systems we recommend to give those files names that are test bench specific.*

2.7 Priority of configuration parameters

In previous sections the level concept has been explained. This concept introduces four levels of priority that define, which of the possible sources of configuration parameters are processed. But there are other rules involved that influence the priority:

- The local configuration has higher priority than other parameter configurations
- The command line has higher priority than definitions within configuration files

Already in command line we have several possibilities to make settings:

- Set a parameter configuration file (with **RobotFramework_TestsuitesManagement** command line variable `config_file`, *level 1*)
- Set a variant name (with **RobotFramework_TestsuitesManagement** command line variable `variant`, *level 2*)
- Set a local configuration (with **RobotFramework_TestsuitesManagement** command line variable `local_config`)
- Set any other variables (directly with Robot Framework command line variable `--variable`)

And it is possible that in all four use cases the same parameters are used. Or in other words: It is possible to use the `--variable` mechanism to define a parameter that is also defined within a parameter configuration or within a local configuration - or in both together.

Finally this is the order of processing (with highest priority first):

1. Single command line variable (`--variable`)
2. Local configuration (`local_config`)
3. Variant specific configuration (`config_file` or `variant`)

Meaning:

1. Variant specific configuration is overwritten by local configuration
2. Local configuration is overwritten by single command line variable

What happens in case of a command line contains both a `config_file` and a `variant`?

`config_file` is level 1 and `variant` is level 2. Level 1 has higher priority than level 2. Therefore `config_file` is the valid one. This does **not** mean that `config_file` overwrites `variant`! In case of a certain level is identified (here: level 1), all other levels are ignored. The outcome is that - in this example - the `variant` has no meaning. Between different levels there is an *either or* relationship. And that is the reason for that it makes no sense to define both in command line, a `config_file` and a `variant`. The **RobotFramework_TestsuitesManagement** throws an error in this case.

But when additionally `--variable` is used to define a new value for a parameter that is already defined in one of the involved configuration files, then the configuration file value is overwritten by the command line value.

And even this is not all. The Robot Framework provides further possibilities to define parameters in command line, e.g. by `--variablefile`. `--variable` and `--variablefile` are Robot Framework mechanisms to define parameters, whereas `config_file` and `local_config` are corresponding **RobotFramework_TestsuitesManagement** mechanisms.

The rules behind all are: `--variable` overrules `--variablefile`. Robot Framework mechanisms overrule **RobotFramework_TestsuitesManagement** mechanisms.

To avoid the things becoming too much complicated, we urgently recommend not to mixup both mechanisms to define *different values for the same parameters* (but to overwrite only a single variable with `--variable` might be OK).

2.8 Nested configuration files

In case of a project requires more and more parameters, it makes sense to split the growing configuration file into smaller ones.

This means, at first we have to split all configuration parameters in

1. parameters that are specific for a certain variant,
2. common parameters that have the same value for all variants

Placing those common parameters in every single variant specific parameter configuration file would create a lot of redundancy. This would also complicate the maintenance.

The solution is to use the variant specific configuration files only for variant specific parameters and to put all common parameters in a separate configuration file. This common parameter file has to be imported in every variant specific parameter file.

The outcome is that still with the selection of a certain variant specific parameter file both types of parameters are available: the variant specific ones and the common ones.

This can be done in the following way:

For example we have the following variant specific configuration files:

```
config/config_variant1.jsonp
config/config_variant2.jsonp
```

Additionaly we have a configuration file with common parameters:

```
config/config_common.jsonp
```

The import of `config_common.jsonp` into `config_variant1.jsonp` and into `config_variant2.jsonp` is possible in the following way:

```
"params" : {
    "global": {
        "[import]" : "./config_common.jsonp",
        "teststring" : "variant specific value"
    }
}
```

The key `[import]` indicates the import of another configuration file. The value of the key is the path and name of this file.

Imports can be nested. An imported configuration file is allowed to contain imports also.

The content of the importing file and the content of all imported files are merged. In case of duplicate parameter names follow up definitions overwrite previous definitions of the same parameter!

Important:

- All imported configuration files are fragments only - and not a full configuration! Even so they need to follow the JSON syntax rules. This means, at least they have to start with an opening curly bracket and they have to end with a closing curly bracket.
- Imported configuration files must not contain the mandatory top level parameters like the `WelcomeString` and others.

2.9 Overwritten parameters

Summarized the **RobotFramework_TestsuitesManagement** provides three different types of parameter configuration files to define parameters:

1. A full standard parameter configuration file containing at least the mandatory parameters and - as option - also user defined parameters
2. A parameter configuration file fragment that is imported in other configuration files by the `[import]` key
3. A local parameter configuration file that is also a fragment only, and accessed either by command line or environment variable

All types of configuration file can be used

1. to define new parameters
2. to overwrite already existing parameters

This possibility only belongs to user defined parameters with scope `params:global`!

Example:

1. Define a new parameter:

```
"params" : {
    "global": {
        "teststring" : "initial value"
    }
}
```

2. Overwrite an already existing parameter:

To overwrite a parameter is - after the initial definition - possible at any follow up position

- in the same configuration file or
- in other configuration files like the imported ones or
- in a local configuration file

With the following syntax:

```
 ${params}['global']['teststring'] : "new value"
```

The resulting value of a parameter at the end depends on the priority (computation order) described in previous sections of this description.

Chapter 3

The JSONP format

This chapter explains the format of JSON files used by the **RobotFramework_TestsuitesManagement** in detail. We concentrate here on the content of the JSON files and the corresponding results, available in Python dictionary format.

3.1 Standard JSON format

The **RobotFramework_TestsuitesManagement** supports JSON files with standard extension `.json` and standard content.

- JSON file:

```
{  
    "param1" : "value1",  
    "param2" : "value2"  
}
```

Outcome:

```
{"param1": "value1", "param2": "value2"}
```

A JSON file with extension `.jsonp` and same content will produce the same output.

We recommend to give every JSON file the extension `.jsonp` to have a strict separation between the standard and the extended JSON format.

The following example still contains standard JSON content, but with parameters of several different data types (simple and composite).

```
{  
    "param_01" : "string",  
    "param_02" : 123,  
    "param_03" : 4.56,  
    "param_04" : ["A", "B", "C"],  
    "param_05" : {"A" : 1, "B" : 2, "C" : 3}  
}
```

This content produces the following output:

```
{"param_01": "string",  
 "param_02": 123,  
 "param_03": 4.56,  
 "param_04": ["A", "B", "C"],  
 "param_05": {"A": 1, "B": 2, "C": 3}}
```

3.2 Boolean and null values

JSON supports the boolean values `true` and `false`, and also the null value `null`.

In Python the corresponding values are different: `True`, `False` and `None`.

Because the `RobotFramework_TestsuitesManagement` is a Python application and therefore the returned content is required to be formatted Python compatible, the `RobotFramework_TestsuitesManagement` does a conversion automatically.

Accepted in JSON files are both styles:

```
{  
    "param_06" : true,  
    "param_07" : false,  
    "param_08" : null,  
    "param_09" : True,  
    "param_10" : False,  
    "param_11" : None  
}
```

The output contains all keywords in Python style only:

```
{'param_06': True,  
'param_07': False,  
'param_08': None,  
'param_09': True,  
'param_10': False,  
'param_11': None}
```

3.3 Comments

Comments can be added to JSON files with `//`:

```
{  
    // JSON keywords  
    "param_06" : true,  
    "param_07" : false,  
    "param_08" : null,  
    // Python keywords  
    "param_09" : True,  
    "param_10" : False,  
    "param_11" : None  
}
```

All lines starting with `//`, are ignored by the **RobotFramework.TestsuitesManagement**. The output of this example is the same than in the previous example.

Also block comments and inline comments are possible, realized by a pair of `/* */`:

```
{  
    /*  
        param1" : 1,  
        "param2" : "A",  
    */  
  
    "testlist" : ["A1", /*"B2", "C3",*/ "D4"]  
}
```

Outcome:

```
{"testlist": ['A1', 'D4']}
```

3.4 Import of JSON files

We assume the following scenario:

A software component *A* requires a set of configuration parameters. A software component *B* that belongs to the same main software or to the same project, requires another set of configuration parameters. Additionally both components require a common set of parameters (with the same values).

The outcome is that at least we need two JSON configuration files:

1. A file `componentA.jsonp` containing all parameters required for component *A*
2. A file `componentB.jsonp` containing all parameters required for component *B*

But with this solution both JSON files would contain also the common set of parameters. This is unfavorable, because the corresponding values need to be maintained at two different positions.

Therefore we extend the list of JSON files by a file containing the common part only:

1. A file `common.jsonp` containing all parameters that are the same for component *A* and component *B*
2. A file `componentA.jsonp` containing remaining parameters (with specific values) required for component *A*
3. A file `componentB.jsonp` containing remaining parameters (with specific values) required for component *B*

Finally we use the import mechanism of the **RobotFramework_TestsuitesManagement** to import the file `common.jsonp` in file `componentA.jsonp` and also in file `componentB.jsonp`.

This can be the content of the JSON files:

```

• common.jsonp
{
    // common parameters
    "common_param_1" : "common value 1",
    "common_param_2" : "common value 2"
}

• componentA.jsonp
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component A parameters
    "componentA_param_1" : "componentA value 1",
    "componentA_param_2" : "componentA value 2"
}

• componentB.jsonp
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component B parameters
    "componentB_param_1" : "componentB value 1",
    "componentB_param_2" : "componentB value 2"
}

```

Explanation:

JSON files are imported with the key `"[import]"`. The value of this key is the path and name of the JSON file to be imported.

A JSON file can contain more than one import. Imports can be nested: An imported JSON file can import further JSON files also.

Outcome:

The file `componentA.jsonp` produces the following output:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common value 2',
 'componentA_param_1': 'componentA value 1',
 'componentA_param_2': 'componentA value 2'}
```

The file `componentB.jsonp` produces the following output:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common value 2',
 'componentB_param_1': 'componentB value 1',
 'componentB_param_2': 'componentB value 2'}
```

It can be seen that the returned dictionary contains both the parameters from the loaded JSON file and the parameters imported by the loaded JSON file.

3.5 Overwrite parameters

We take over the scenario from the previous section: We still have a JSON file `componentA.jsonp` containing the parameters for component *A*, a JSON file `componentB.jsonp` for component *B* and a JSON file `common.jsonp` for both components.

But now component *B* requires a different value of a common parameter: Within a JSON file we need to change the value of a parameter that is initialized within an imported file. That is possible.

This is now the content of the JSON files:

- `common.jsonp`

```
{
    // common parameters
    "common_param_1": "common value 1",
    "common_param_2": "common value 2"
}
```

- `componentA.jsonp`

```
{
    // common parameters
    "[import]": "./common.jsonp",
    //
    // component A parameters
    "componentA_param_1": "componentA value 1",
    "componentA_param_2": "componentA value 2"
}
```

- `componentB.jsonp`

```
{
    // common parameters
    "[import]": "./common.jsonp",
    //
    // component B parameters
    "componentB_param_1": "componentB value 1",
    "componentB_param_2": "componentB value 2",
    // overwrite parameter initialized by imported file
    "common_param_2": "common componentB value 2"
}
```

Explanation:

With

```
"common_param_2" : "common componentB value 2"
```

in `componentB.jsonp`, the initial definition

```
"common_param_2" : "common value 2"
```

in `common.jsonp` is overwritten.

Outcome:

The file `componentB.jsonp` produces the following output:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common componentB value 2',
 'componentB_param_1': 'componentB value 1',
 'componentB_param_2': 'componentB value 2'}
```

Important: *The value a parameter has finally, depends on the order of definitions, redefinitions and imports!*

In file `componentB.jsonp` we move the import of `common.jsonp` to the bottom:

```
{
    // component B parameters
    "componentB_param_1" : "componentB value 1",
    "componentB_param_2" : "componentB value 2",
    "common_param_2" : "common componentB value 2"
    //
    // common parameters
    "[import]" : "./common.jsonp",
}
```

Now the imported file overwrites the value initialized in the importing file.

Outcome:

```
{'common_param_1': 'common value 1',
 'common_param_2': 'common value 2',
 'componentB_param_1': 'componentB value 1',
 'componentB_param_2': 'componentB value 2'}
```

Up to now we considered simple data types only. In case we want to overwrite a parameter that is part of a composite data type, we need to extend the syntax. This is explained in the next examples.

Again we take over the scenario from the previous section: We still have a JSON file `componentA.jsonp` containing the parameters for component A, a JSON file `componentB.jsonp` for component B and a JSON file `common.jsonp` for both components.

But now all values are part of composite data types like lists and dictionaries.

This is the content of the JSON files:

- `common.jsonp`

```
{
    // common parameters
    "common_param_1" : ["common value 1.1", "common value 1.2"],
    "common_param_2" : {"common_key_2_1" : "common value 2.1",
                      "common_key_2_2" : "common value 2.2"}
}
```

CHAPTER 3. THE JSONP FORMAT3.5. OVERWRITE PARAMETERS

• componentA.jsonp

```
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component A parameters
    "componentA_param_1" : ["componentA value 1.1", "componentA value 1.2"],
    "componentA_param_2" : {"componentA_key_2_1" : "componentA value 2.1",
                           "componentA_key_2_2" : "componentA value 2.2"}
}
```

• componentB.jsonp

```
{
    // common parameters
    "[import]" : "./common.jsonp",
    //
    // component B parameters
    "componentB_param_1" : ["componentB value 1.1", "componentB value 1.2"],
    "componentB_param_2" : {"componentB_key_2_1" : "componentB value 2.1",
                           "componentB_key_2_2" : "componentB value 2.2"}
}
```

Like in previous examples, the outcome is a merge of the imported JSON file and the importing JSON file, e.g. for componentA.jsonp :

```
{"common_param_1": ['common value 1.1', 'common value 1.2'],
 "common_param_2": {"common_key_2_1": 'common value 2.1',
                   'common_key_2_2': 'common value 2.2'},
 'componentA_param_1': ['componentA value 1.1', 'componentA value 1.2'],
 'componentA_param_2': {"componentA_key_2_1": 'componentA value 2.1',
                       'componentA_key_2_2': 'componentA value 2.2'}}}
```

Now the following questions need to be answered:

1. How to get the value of an already existing parameter?
2. How to get the value of a single element of a parameter of nested data type (list, dictionary)?
3. How to overwrite the value of a single element of a parameter of nested data type?
4. How to add an element to a parameter of nested data type?

We introduce another JSON file componentB.2.jsonp in which we import the JSON file componentB.jsonp . In this file we also add content to work with simple and composite data types to answer the questions above.

CHAPTER 3. THE JSONP FORMAT3.5. OVERWRITE PARAMETERS

This is the initial content of `componentB.2.jsonp`:

```
{
    // import of componentB parameters
    "[import]" : "./componentB.jsonp",
    //
    // some additional parameters of simple data type
    "string_val" : "ABC",
    "int_val" : 123,
    "float_val" : 4.56,
    "bool_val" : true,
    "null_val" : null,

    // access to existing parameters
    "string_val_b" : ${string_val},
    "int_val_b" : ${int_val},
    "float_val_b" : ${float_val},
    "bool_val_b" : ${bool_val},
    "null_val_b" : ${null_val},
    "common_param_1_b" : ${common_param_1},
    "componentB_param_2_b" : ${componentB_param_2}
}
```

The rules for accessing parameters are:

- Existing parameters are accessed by a dollar operator and a pair of curly brackets (`${...}`) with the parameter name inside.
- If the entire expression of the right hand side of the colon is such a dollar operator expression, it is not required any more to encapsulate this expression in quotes.
- Without quotes, the dollar operator keeps the data type of the referenced parameter. If you use quotes, the value of the used parameter will be of type `str` .

Outcome:

```
{
    'bool_val': True,
    'bool_val_b': True,
    'common_param_1': ['common value 1.1', 'common value 1.2'],
    'common_param_1_b': ['common value 1.1', 'common value 1.2'],
    'common_param_2': {'common_key_2_1': 'common value 2.1',
                      'common_key_2_2': 'common value 2.2'},
    'componentB_param_1': ['componentB value 1.1', 'componentB value 1.2'],
    'componentB_param_2': {'componentB_key_2_1': 'componentB value 2.1',
                           'componentB_key_2_2': 'componentB value 2.2'},
    'componentB_param_2_b': {'componentB_key_2_1': 'componentB value 2.1',
                            'componentB_key_2_2': 'componentB value 2.2'},
    'float_val': 4.56,
    'float_val_b': 4.56,
    'int_val': 123,
    'int_val_b': 123,
    'null_val': None,
    'null_val_b': None,
    'string_val': 'ABC',
    'string_val_b': 'ABC'}
```

Let's take a deeper look at the following line:

```
"int_val_b" : ${int_val},
```

Like mentioned in the rules above, the dollar operator keeps the data type of the referenced parameter: In case of `int_val` is of type `int`, also `int_val_b` will be of type `int`.

CHAPTER 3. THE JSONP FORMAT3.5. OVERWRITE PARAMETERS

Like mentioned in the rules above, it is not required any more to encapsulate dollar operator expressions at the right hand side of the colon in quotes. But nevertheless it is possible to use quotes. In case of:

```
"int_val_b" : "${int_val}",
```

the parameter `int_val_b` is of type `string`.

Value of a single element of a parameter of nested data type

To access an element of a list and a key of a dictionary, we change the content of file `componentB.2.jsonp` to:

```
{
  // import of componentB parameters
  "[import]" : "./componentB.jsonp",
  //
  "list_element_0" : ${componentB_param_1}[0],
  "dict_key_2_2" : ${common_param_2}['common_key_2_2']
}
```

Outcome:

```
{"common_param_1": ["common value 1.1", "common value 1.2"],
 "common_param_2": {"common_key_2_1": "common value 2.1",
                   "common_key_2_2": "common value 2.2"},
 "componentB_param_1": ["componentB value 1.1", "componentB value 1.2"],
 "componentB_param_2": {"componentB_key_2_1": "componentB value 2.1",
                       "componentB_key_2_2": "componentB value 2.2"},
 "dict_key_2_2": "common value 2.2",
 "list_element_0": "componentB value 1.1"}
```

Overwrite the value of a single element of a parameter of nested data type

In the next example we overwrite the value of a list element and the value of a dictionary key.

Again we change the content of file `componentB.2.jsonp`:

```
{
  // import of componentB parameters
  "[import]" : "./componentB.jsonp",
  //
  ${componentB_param_1}[0] : "componentB value 1.1 (new)",
  ${common_param_2}['common_key_2_1'] : "common value 2.1 (new)"
}
```

The dollar operator syntax at the left hand side of the colon is the same than previously used on the right hand side. The entire expression at the left hand side of the colon must *not* be encapsulated in quotes in this case.

Outcome:

The single elements of the list and the dictionary are updated, all other elements are unchanged.

```
{"common_param_1": ["common value 1.1", "common value 1.2"],
 "common_param_2": {"common_key_2_1": "common value 2.1 (new)",
                   "common_key_2_2": "common value 2.2"},
 "componentB_param_1": ["componentB value 1.1 (new)", "componentB value 1.2"],
 "componentB_param_2": {"componentB_key_2_1": "componentB value 2.1",
                       "componentB_key_2_2": "componentB value 2.2"}}
```

Add an element to a parameter of nested data type

Adding further elements to an already existing list is not possible in JSON! But it is possible to add keys to an already existing dictionary.

The following example extends the dictionary `common_param_2` by an additional key `common_key_2_3` :

```
{
    // import of componentB parameters
    "[import]" : "./componentB.jsonp",
    //
    ${common_param_2}['common_key_2_3'] : "common value 2.3"
}
```

Outcome:

```
{'common_param_1': ['common value 1.1', 'common value 1.2'],
'common_param_2': {'common_key_2_1': 'common value 2.1',
                  'common_key_2_2': 'common value 2.2',
                  'common_key_2_3': 'common value 2.3'},
'componentB_param_1': ['componentB value 1.1', 'componentB value 1.2'],
'componentB_param_2': {'componentB_key_2_1': 'componentB value 2.1',
                      'componentB_key_2_2': 'componentB value 2.2'}}
```

Dictionary keys and indices as parameter

In all code examples above the indices of lists and the key names of dictionaries have been hard coded strings. It is also possible to use parameters:

```
{
    "index1" : 0,
    "index2" : 1,
    "key1" : "keyA",
    "key2" : "keyB",
    "testlist" : ["A", "B"],
    "testdict" : {"keyA" : "A", "keyB" : "B"},
    "tmp1" : ${testlist}[${index1}],
    "tmp2" : ${testdict}[${key1}],
    ${testlist}[${index1}] : ${testlist}[${index2}],
    ${testdict}[${key1}] : ${testdict}[${key2}],
    ${testlist}[${index2}] : ${tmp1},
    ${testdict}[${key2}] : ${tmp2}
}
```

Outcome:

```
{'index1': 0,
'index2': 1,
'key1': 'keyA',
'key2': 'keyB',
'testdict': {'keyA': 'B', 'keyB': 'A'},
'testlist': ['B', 'A'],
'tmp1': 'A',
'tmp2': 'A'}
```

Meaning of single quotes in square brackets

Single quotes are used to convert the content inside to a string.

- In case of the parameter `param` is of type string, the expressions `[$param]` and `['$param']` have the same outcome: The content inside the square brackets is a string. The single quotes have no meaning in this case (because the parameter is already of type string).
- In case of the parameter `param` is of type integer, the quotes in `['$param']` convert the integer value to a string. Without the quotes (`[$param]`), the content inside the square brackets is an integer.

In the context of **RobotFramework_TestsuitesManagement** JSON files, only strings and integers are expected to be inside square brackets (except the brackets are used to define a list). Other data types are not supported here.

Whether a string or an integer is expected, depends on the data type of the parameter, the square bracket expression belongs to. Dictionaries require a string (a key name), lists require an integer (an index). Deviations will cause an error.

Summarized the following combinations are valid (on both the left hand side of the colon and the right hand side of the colon):

```

${listparam}[$intparam]
${listparam}[1]
${dictparam}['${intparam}']
${dictparam}[$stringparam]
${dictparam}['${stringparam}']
${dictparam}['keyname']

```

Use of a common dictionary

The last example in this section covers the following use case:

- We have several JSON files, each for a certain purpose within a project (e.g. for every feature of this project a separate JSON file).
- They belong together and therefore they are all imported into a main JSON file that is the file that is handed over to the **RobotFramework_TestsuitesManagement**.
- Every imported JSON file introduces a certain bunch of parameters. All parameters need to be a part of a common dictionary.
- Outcome is that finally only one single dictionary is used to access the parameters from all JSON files imported in the main JSON file.

To realize this, it is necessary to separate the initialization of the dictionary from all positions where keys are added to this dictionary.

CHAPTER 3. THE JSONP FORMAT3.5. OVERWRITE PARAMETERS

These are the JSON files:

- `project.jsonp`

```
{
    // initialization
    "project_values" : {},
    //
    // add some common values
    ${project_values}['common_project_param_1'] : "common project value 1",
    ${project_values}['common_project_param_2'] : "common project value 2",
    //
    // import feature parameters
    "[import]" : "./featureA.jsonp",
    "[import]" : "./featureB.jsonp",
    "[import]" : "./featureC.jsonp"
}
```

- `featureA.jsonp`

```
{
    // parameters required for feature A
    ${project_values}['featureA_params'] : {},
    ${project_values}['featureA_params']['featureA_param_1'] : "featureA param 1 value",
    ${project_values}['featureA_params']['featureA_param_2'] : "featureA param 2 value"
}
```

- `featureB.jsonp`

```
{
    // parameters required for feature B
    ${project_values}['featureB_params'] : {},
    ${project_values}['featureB_params']['featureB_param_1'] : "featureB param 1 value",
    ${project_values}['featureB_params']['featureB_param_2'] : "featureB param 2 value"
}
```

- `featureC.jsonp`

```
{
    // parameters required for feature C
    ${project_values}['featureC_params'] : {},
    ${project_values}['featureC_params']['featureC_param_1'] : "featureC param 1 value",
    ${project_values}['featureC_params']['featureC_param_2'] : "featureC param 2 value"
}
```

Explanation:

Every `feature*.jsonp` file refer to the dictionary initialized within `project.jsonp`.

Important is that the initialization `"project_values" : {}`, happens at top level (`project.jsonp`), and not within the imported files (`feature*.jsonp`), otherwise follow up initializations would delete previously added keys of this dictionary!

Outcome:

```
{'project_values': {'common_project_param_1': 'common project value 1',
                    'common_project_param_2': 'common project value 2',
                    'featureA_params': {'featureA_param_1': 'featureA param 1 value',
                                         'featureA_param_2': 'featureA param 2 value'},
                    'featureB_params': {'featureB_param_1': 'featureB param 1 value',
                                         'featureB_param_2': 'featureB param 2 value'},
                    'featureC_params': {'featureC_param_1': 'featureC param 1 value',
                                         'featureC_param_2': 'featureC param 2 value'}}}
```

3.6 dotdict notation

Up to now we have accessed dictionary keys in this way (standard notation):

```
 ${dictionary}['key']['sub_key']
```

Additionally to this standard notation, the **RobotFramework_TestsuitesManagement** supports the so called *dotdict* notation where keys are handled as attributes:

```
 ${dictionary.key.sub_key}
```

In standard notation keys are encapsulated in square brackets and all together is placed *outside* the curly brackets. In dotdict notation the dictionary name and the keys are separated by dots from each other. All together is placed *inside* the curly brackets.

In standard notation key names are allowed to contain dots:

```
 ${dictionary['key']['sub.key']}
```

In dotdict notation this would cause ambiguities:

```
 ${dictionary.key.sub.key}
```

Therefore it is not possible to implement in this way! In case you need to have dots inside key names, you must use the standard notation. We recommend to prefer underlines as separator - like done in the examples in this document.

Do you really need dots inside key names?

Please keep in mind: The dotdict notation is a reduced one. Because of parts are missing (e.g. the single quotes around key names), the outcome can be code that is really hard to capture.

In the following example we create a composite data structure and demonstate how to access single elements in both notations.

- JSON file:

```
{
    // composite data structure
    "params" : [{"dict_1_key_1" : "dict_1_key_1 value",
                 "dict_1_key_2" : ["dict_1_key_2 value 1", "dict_1_key_2 value 2"]},
                //
                {"dict_2_key_1" : "dict_2_key_1 value",
                 "dict_2_key_2" : {"dict_2_A_key_1" : "dict_2_A_key_1 value",
                                  "dict_2_A_key_2" : ["dict_2_A_key_2 value 1", ↪
                                         "dict_2_A_key_2 value 2"]}}],
    //
    // access to single elements of composite data structure
    //
    // a) standard notation
    "dict_1_key_2_value_2_standard" : ${params}[0]['dict_1_key_2'][1],
    // b) dotdict notation
    "dict_1_key_2_value_2_dotdict" : ${params.0.dict_1_key_2.1},
    //
    // c) standard notation
    "dict_2_A_key_2_value_2_standard" : ${params}[1]['dict_2_key_2']['dict_2_A_key_2'][1]
    // d) dotdict notation
    "dict_2_A_key_2_value_2_dotdict" : ${params.1.dict_2_key_2.dict_2_A_key_2.1}
}
```

Outcome:

In case of the composite data structure becomes more and more nested (and if also the key names contain numbers), understanding the expressions (like `${params.1.dict_2_key_2.dict_2_A_key_2.1}`) becomes more and more challenging!

```
{'dict_1_key_2_value_2_standard': 'dict_1_key_2 value 2',
 'dict_2_A_key_2_value_2_standard': 'dict_2_A_key_2 value 2',
 'params': [{{'dict_1_key_1': 'dict_1_key_1 value',
   'dict_1_key_2': ['dict_1_key_2 value 1', 'dict_1_key_2 value 2']},
   {'dict_2_key_1': 'dict_2_key_1 value',
   'dict_2_key_2': {'dict_2_A_key_1': 'dict_2_A_key_1 value',
     'dict_2_A_key_2': ['dict_2_A_key_2 value 1',
       'dict_2_A_key_2 value 2']}}}]}
```

3.7 Substitution of dollar operator expressions

Like shown in previous examples, existing parameters are accessed by a dollar operator and a pair of curly brackets (`${...}`) with the parameter name inside.

We discussed use cases, where the entire expression of the left hand side of the colon and also the entire expression of the right hand side of the colon have been either hard coded strings, encapsulated in quotes, or dollar operator expressions, not encapsulated in quotes.

Also a mix of both is possible!

Outcome is a hard coded string, encapsulated in quotes, with parts represented by one or more than one dollar operator expression. This can be used to create new content very dynamically: On the right hand side of the colon new string values can be created; on the left hand side of the colon this mechanism generates dynamic parameter names.

- JSON file:

```
{
    "project"      : "Test",
    "version"      : 1.23,
    "item_number"  : 1,
    "component"    : "componentA",
    //
    "${project}_message_${item_number}" : "Component '${component}' has version ${version}"
}
```

Outcome:

```
{'component': 'componentA',
'item_number': 1,
'project': 'Test',
'Test_message_1': "Component 'componentA' has version 1.23",
'version': 1.23}
```

We recommend to use simple data types for this kind of substitution only!

But nevertheless, on the right hand side of the colon also composite data types are possible. Here it might makes sense to use them. But on the left hand side of the colon only simple data types are allowed for substitution. Because it makes no sense to create parameter names based on composite data types. Most probably this would cause invalid key names.

To demonstrate this, we change the JSON file to:

```
{
    "component"      : "componentA",
    "composite_data" : ["AB", 12, True, null, {"kA" : "kAval", "kB" : "kBval"}],
    //
    "test_parameter" : "Key values of component '${component}' are: ${composite_data}"
}
```

Outcome:

```
{'component': 'componentA',
'composite_data': ['AB', 12, True, None, {'kA': 'kAval', 'kB': 'kBval'}],
'test_parameter': "Key values of component 'componentA' are: ['AB', 12, True, "
                  "None, {'kA': 'kAval', 'kB': 'kBval'}]"}
```

[CHAPTER 3. THE JSONP FORMAT](#)[3.7. SUBSTITUTION OF DOLLAR OPERATOR EXPRESSIONS](#)

Now we try to do the same on the left hand side of the colon:

```
{  
    "param" : "string value",  
    "composite_data" : ["AB", 12, True, null, {"kA" : "kAval", "kB" : "kBval"}],  
    //  
    "test_parameter_`${composite_data}" : ${param}  
}
```

If this would be allowed, the last line would cause a parameter with this name:

```
"test_parameter_['AB', 12, True, None, {'kA': 'kAval', 'kB': 'kBval'}]"
```

And this is absolutely not a valid name!

To prevent the users from generating invalid parameter names, the **RobotFramework_TestsuitesManagement** throws an error:

```
Error: 'Found expression '...' with at least one parameter of composite data type ...  
Because of this expression is the name of a parameter, only simple data types are ↵  
↳ allowed to be substituted inside.'!'
```

3.8 Overwriting vs. substitution

In this section we take a deeper look at the syntax difference between overwriting and substitution explained above.

1. Overwriting:

With `${param2} : ${param1}` the initial value `"XYZ"` of parameter `param2` is overwritten with the value `"ABC"` of parameter `param1`.

```
{
  "param1" : "ABC",
  "param2" : "XYZ",
  //
  ${param2} : ${param1},
}
```

Result:

```
{"param1" : "ABC",
"param2" : "ABC"}
```

2. Substitution:

With `"${param2}" : ${param1}` a new parameter with name `XYZ` (the value of `param2`) and value `"ABC"` is created.

```
{
  "param1" : "ABC",
  "param2" : "XYZ",
  //
  "${param2}" : ${param1},
}
```

Result:

```
{"param1" : "ABC",
"param2" : "XYZ",
"XYZ" : "ABC"}
```

3.9 Implicite creation of dictionaries

Up to now we have discussed two different ways of creating nested dictionaries.

The first one is “on the fly”, like:

```
{
    "project_values" : {"keyA" : "keyA value",
                        "keyB" : {"keyB1" : "keyB1 value",
                                  "keyB2" : {"keyB21" : "keyB21 value",
                                             "keyB22" : "keyB22 value"}{}}
}
```

In case of it is required to split the definition into several files, we have to add keys (and also the initialization) line by line:

```
{
    "project_values" : {},
    ${project_values}['keyA'] : "keyA value",
    ${project_values}['keyB'] : {},
    ${project_values}['keyB']['keyB1'] : "keyB1 value",
    ${project_values}['keyB']['keyB2'] : {},
    ${project_values}['keyB']['keyB2']['keyB21'] : "keyB21 value",
    ${project_values}['keyB']['keyB2']['keyB22'] : "keyB22 value"
}
```

The result will be the same as in the previous example.

It can be seen now that this way of creating nested dictionaries is rather long winded, because every initialization of a dictionary requires a separate line of code (at every level).

To shorten the code, the **RobotFramework_TestsuitesManagement** supports an implicite creation of dictionaries.

This is the resulting code in standard notation:

```
{
    ${project_values}['keyA'] : "keyA value",
    ${project_values}['keyB']['keyB1'] : "keyB1 value",
    ${project_values}['keyB']['keyB2']['keyB21'] : "keyB21 value",
    ${project_values}['keyB']['keyB2']['keyB22'] : "keyB22 value"
}
```

And the same in dotdict notation (with precondition, that no key name contains a dot):

```
{
    ${project_values.keyA} : "keyA value",
    ${project_values.keyB.keyB1} : "keyB1 value",
    ${project_values.keyB.keyB2.keyB21} : "keyB21 value",
    ${project_values.keyB.keyB2.keyB22} : "keyB22 value"
}
```

Caution:

We urgently recommend *not* to mixup both styles in one line of code. In case of keys contain a list and also numerical indices are involved, we recommend to prefer the standard notation.

Please be aware of: In case of a missing level in between an expression like

```
{
    ${project_values.keyB.keyB2.keyB22} : "keyB22 value"
}
```

you will *not* get an error message! The entire data structure will be created implicitly. The impact is that this method is very susceptible to typing mistakes.

The implicite creation of data structures does not work with lists! In case you use a list index out of range, you will get a corresponding error message.

Key names

The implicit creation of data structures is only possible with *hard coded* key names. Parameters are not supported.

Example:

```
{
    "paramA" : "ABC",
    "subKey" : "ABC",
    ${testdict.subKey.subKey.paramA} : "DEF"
}
```

All sub key levels within the expression `${testdict.subKey.subKey.paramA}` are interpreted as hard coded strings, even in case of parameters with the same name do exist.

For example: The name of the implicitly created key at bottom level is `"paramA"`, and not the value `"ABC"` of the parameter with the same name (`"paramA"`).

Therefore the outcome is:

```
{"paramA": "ABC", "subKey": "ABC", "testdict": {"subKey": {"subKey": {"paramA": "DEF"}}}}
```

Reference to existing keys

It is possible to use parameters to refer to *already existing* keys.

```
{
    // data structure created implicitly
    ${testdict.subKey_1.subKey_2.subKey_3} : "ABC",

    // string parameter with name of an existing key
    "keyName_3" : "subKey_3",

    // parameter used to refer to an existing key
    ${testdict.subKey_1.subKey_2.${keyName_3}} : "XYZ"
}
```

Outcome:

```
{"keyName_3": "subKey_3",
 "testdict": {"subKey_1": {"subKey_2": {"subKey_3": "XYZ"}}}}
```

Parameters cannot be used to create new keys.

```
{
    // data structure created implicitly
    ${testdict.subKey_1.subKey_2.subKey_3} : "ABC",

    // string parameter with name of a not existing key
    "keyName_4" : "subKey_4",

    // usage of keyName_4 is not possible here
    ${testdict.subKey_1.subKey_2.subKey_3.${keyName_4}} : "XYZ"
}
```

Outcome is the following error:

```
"The implicit creation of data structures based on nested parameter is not supported ..."
```

The same error will happen in case of the standard notation is used:

```
{
    // usage of keyName_4 is not possible here
    ${testdict}['subKey_1']['subKey_2']['subKey_3'][$keyName_4] : "XYZ"
}
```

CHAPTER 4. CCONFIG.PY

Chapter 4

CConfig.py

4.1 Function: bundle_version

This function prints out the package version which is:

- RobotFramework_TestsuitesManagement version when this module is installed stand-alone (via pip or directly from sourcecode)
- RobotFramework AIO version when this module is bundled with RobotFramework AIO package

Arguments:

- No input parameter is required

Returns:

- No return variable

4.2 Class: CConfig

Imported by:

```
from RobotFramework_TestsuitesManagement.Config.CConfig import CConfig
```

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

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

Level1: Handed over by command line argument

Level2: Read from content of json config file

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

CHAPTER 4. CCONFIG.PY**4.2. CLASS: CCONFIG**

According to the ConfigName, RobotFramework_TestsuitesManagement will choose the corresponding config file. ".../config/" indicates the relative path to json config file, RobotFramework_TestsuitesManagement will recursively find the config folder.

Level3: Read in testsuite folder: /config/robot_config.jsonp

Level4: Read from RobotFramework AIO installation folder:

/RobotFramework/defaultconfig/robot_config.jsonp

4.2.1 Method: loadCfg

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

Arguments:

- No input parameter is required

Returns:

- No return variable

4.2.2 Method: verifyVersion

This verifyVersion validates the current package version with maximum and minimum version (if provided in the configuration file).

The package version is:

- RobotFramework_TestsuitesManagement version when this module is installed stand-alone (via pip or directly from sourcecode)

- RobotFramework AIO version when this module is bundled with RobotFramework AIO package

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

4.2.3 Method: bValidateMinVersion

This bValidateMinVersion validates the current version with required minimum version.

Arguments:

- tCurrentVersion
/ Condition: required / Type: tuple /
Current package version.
- tMinVersion
/ Condition: required / Type: tuple /
The minimum version of package.

Returns:

- True or False

4.2.4 Method: bValidateMaxVersion

This bValidateMaxVersion validates the current version with required minimum version.

Arguments:

- tCurrentVersion
/ Condition: required / Type: tuple /
 Current package version.
- tMinVersion
/ Condition: required / Type: tuple /
 The minimum version of package.

Returns:

- True or False

4.2.5 Method: bValidateSubVersion

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

Arguments:

- sVersion
/ Condition: required / Type: string /
 The version of package.

Returns:

- lSubVersion
/ Type: tuple /

4.2.6 Method: tupleVersion

This tupleVersion returns a tuple which contains the (major, minor, patch) version.

In case minor/patch version is missing, it is set to 0. E.g: "1" is transformed to "1.0.0" and "1.1" is transformed to "1.1.0"

This tupleVersion also support version which contains Alpha (a), Beta (b) or Release candidate (rc): E.g: "1.2rc3", "1.2.1b1", ...

Arguments:

- sVersion
/ Condition: required / Type: string /
 The version of package.

Returns:

- lVersion
/ Type: tuple /
 A tuple which contains the (major, minor, patch) version.

4.2.7 Method: versioncontrol_error

Wrapper version control error log:

Log error message of version control due to reason and set to unknown state.

Arguments:

- `reason`
/ Condition: required / Type: string /
reason can only be conflict_min, conflict_max and wrong_minmax.
- `version1`
/ Condition: required / Type: string /
- `version2`
/ Condition: required / Type: string /

Returns:

- No return variable

CHAPTER 5. CONFAILUREHANDLE.PY

Chapter 5

COnFailureHandle.py

5.1 Class: COnFailureHandle

Imported by:

```
from RobotFramework_TestsuitesManagement.Keywords.COnFailureHandle import  
    COnFailureHandle
```

5.1.1 Method: is_noney

CHAPTER 6. CSETUP.PY

Chapter 6

CSetup.py

6.1 Class: CSetupKeywords

Imported by:

```
from RobotFramework_TestsuitesManagement.Keywords.CSetup import 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.

6.1.1 Keyword: 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 variable `config_file` is not set RobotFramework AIO will check for configuration level 3, and level 4.
`sTestsuiteCfgFile` is set with a `<json_config_file_path>` and variable `config_file` is not set RobotFramework AIO will load configuration level 2.

Returns:

- No return variable

6.1.2 Keyword: testsuite_teardown

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

6.1.3 Keyword: testcase_setup

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

6.1.4 Keyword: testcase_teardown

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

6.2 Class: CGeneralKeywords

Imported by:

```
from RobotFramework_TestsuitesManagement.Keywords.CSetup import 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.

6.2.1 Keyword: 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 /

6.2.2 Keyword: load_json

Loads a json file and returns a 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 7. CSTRUCT.PY

Chapter 7

CStruct.py

7.1 Class: CStruct

Imported by:

```
from RobotFramework_TestsuitesManagement.Utils.CStruct import CStruct
```

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

CHAPTER 8. EVENT.PY

Chapter 8

Event.py

8.1 Class: Event

Imported by:

```
from RobotFramework_TestsuitesManagement.Utils.Events.Event import Event
```

8.1.1 Method: trigger

CHAPTER 9. SCOPEEVENT.PY

Chapter 9

ScopeEvent.py

9.1 Class: ScopeEvent

Imported by:

```
from RobotFramework_TestsuitesManagement.Utils.Events.ScopeEvent import ScopeEvent
```

9.1.1 Method: trigger

9.2 Class: ScopeStart

Imported by:

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

9.3 Class: ScopeEnd

Imported by:

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

CHAPTER 10. __INIT__.PY

Chapter 10

__init__.py

10.1 Function: on

10.2 Function: dispatch

10.3 Function: register_event

CHAPTER 11. LIBLISTENER.PY

Chapter 11

LibListener.py

11.1 Class: LibListener

Imported by:

```
from RobotFramework_TestsuitesManagement.Utils.LibListener import 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 12. __INIT__.PY

Chapter 12

__init__.py

12.1 Class: RobotFramework_TestsuitesManagement

Imported by:

```
from RobotFramework_TestsuitesManagement.__init__ import
    ↳ RobotFramework_TestsuitesManagement
```

12.1.1 Method: run_keyword

12.1.2 Method: get_keyword_tags

12.1.3 Method: get_keyword_documentation

12.1.4 Method: failure_occurred

12.2 Class: CTestsuitesCfg

Imported by:

```
from RobotFramework_TestsuitesManagement.__init__ import CTestsuitesCfg
```

CHAPTER 13. APPENDIX

Chapter 13

Appendix

About this package:

Table 13.1: Package setup

Setup parameter	Value
Name	RobotFramework_TestsuitesManagement
Version	0.7.6
Date	01.04.2024
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

Chapter 14

History

0.1.0	06/2022
<i>Initial version</i>	
0.2.2	07/2022
<i>Created documentation and updated message logs</i>	
0.3.0	07/2022
<i>Added local configuration feature; documentation rework</i>	
0.4.0	03/2023
<i>Maintenance of log output; maintenance of JSON schema validation of configuration file</i>	
0.7.1	05/2023
<i>Introduce package context which allows RobotFramework.TestsuitesManagement work in many contexts:</i> - stand-alone - as part of RobotFramework AIO package	
0.7.3	09/2023
<i>New RobotFramework 6.1 adaptation; Maintenance of log output</i>	
0.7.4	12/2023
<i>- Improve dotdict feature - Update logging - Code maintenance</i>	
0.7.5	03/2024
<i>- Simplify local configuration loading based on the new version of JsonPreprocessor</i>	
0.7.6	04/2024
<i>- Added naming convention check for parameter names. - Updated logging message.</i>	

RobotFramework.TestsuitesManagement.pdf*Created at 10.04.2024 - 12:33:44**by GenPackageDoc v. 0.41.1*

8.6 QConnectBase

QConnectBase

v. 1.1.3

Nguyen Huynh Tri Cuong

06.06.2023

Contents

1	Introduction	1
2	Description	2
2.1	Getting Started	2
2.2	Usage	2
2.2.1	connect	2
2.2.2	disconnect	3
2.2.3	send command	3
2.2.4	transfer file	3
2.2.5	verify	4
2.3	Example	4
2.4	Auxiliary Utilities	5
2.4.1	Introduction to RMQSignal	5
2.4.2	Prerequisites	5
2.4.3	Library Keywords and Usage	6
2.4.4	Example of Inter-Process Communication	7
2.5	Contribution Guidelines	7
2.6	Configure Git and correct EOL handling	8
2.7	Sourcecode Documentation	8
2.8	Feedback	8
2.9	About	9
2.9.1	Maintainers	9
2.9.2	Contributors	9
2.10	License	9
3	<code>__init__.py</code>	10
3.1	Class: ConnectionManager	10
4	<code>connection_base.py</code>	11
4.1	Class: BrokenConnError	11
4.2	Class: ConnectionBase	11
4.2.1	Method: <code>is_supported_platform</code>	11
4.2.2	Method: <code>is_precondition_pass</code>	11
4.2.3	Method: <code>get_connection_type</code>	11
4.2.4	Method: <code>error_instruction</code>	12
4.2.5	Method: <code>quit</code>	12
4.2.6	Method: <code>connect</code>	12
4.2.7	Method: <code>disconnect</code>	12

CONTENTS	CONTENTS
----------	----------

4.2.8 Method: send_obj	13
4.2.9 Method: read_obj	13
4.2.10 Method: wait_4_trace	13
4.2.11 Method: wait_4_trace_continuously	14
4.2.12 Method: create_and_activate_trace_queue	14
4.2.13 Method: deactivate_and_delete_trace_queue	15
4.2.14 Method: activate_trace_queue	15
4.2.15 Method: deactivate_trace_queue	16
4.2.16 Method: check_timeout	16
4.2.17 Method: pre_msg_check	16
4.2.18 Method: post_msg_check	16
5 connection_manager.py	17
5.1 Class: InputParam	17
5.1.1 Method: get_attr_list	17
5.2 Class: ConnectParam	17
5.3 Class: SendCommandParam	17
5.4 Class: VerifyParam	17
5.5 Class: ConnectionManager	17
5.5.1 Method: quit	18
5.5.2 Method: add_connection	18
5.5.3 Method: remove_connection	18
5.5.4 Method: get_connection_by_name	18
5.5.5 Keyword: disconnect	19
5.5.6 Keyword: connect	19
5.5.7 Keyword: send_command	19
5.5.8 Keyword: transfer_file	20
5.5.9 Keyword: verify	20
5.6 Class: TestOption	21
6 constants.py	22
6.1 Class: SocketType	22
6.2 Class: String	22
7 rabbitmq_client.py	23
7.1 Class: RabbitmqClientConfig	23
7.2 Class: RabbitmqClient	23
7.2.1 Method: on_response	23
7.2.2 Method: connect	23
7.2.3 Method: close	23
7.2.4 Method: quit	23
7.3 Class: RMQSignal	23
7.3.1 Method: send_signal	24
7.3.2 Method: unset_signal_receiver_name	24
7.3.3 Method: set_signal_receiver_name	24
7.3.4 Method: consume_channel	24
7.3.5 Method: wait_for_signal	25

CONTENTSCONTENTS

7.3.6 Method: wait_for_signals	25
8 qlogger.py	27
8.1 Class: ColorFormatter	27
8.1.1 Method: format	27
8.2 Class: QFileHandler	27
8.2.1 Method: get_log_path	27
8.2.2 Method: get_config_supported	28
8.3 Class: QDefaultFileHandler	28
8.3.1 Method: get_log_path	28
8.3.2 Method: get_config_supported	28
8.4 Class: QConsoleHandler	29
8.4.1 Method: get_config_supported	29
8.5 Class: QLogger	29
8.5.1 Method: get_logger	29
8.5.2 Method: set_handler	29
9 serial_base.py	31
9.1 Class: SerialConfig	31
9.2 Class: SerialSocket	31
9.2.1 Method: connect	31
9.2.2 Method: disconnect	31
9.2.3 Method: quit	31
9.3 Class: SerialClient	32
9.3.1 Method: connect	32
10 raw_tcp.py	33
10.1 Class: RawTCPBase	33
10.2 Class: RawTCPServer	33
10.3 Class: RawTCPClient	33
11 ssh_client.py	34
11.1 Class: AuthenticationType	34
11.2 Class: SSHConfig	34
11.3 Class: SSHClient	34
11.3.1 Method: connect	34
11.3.2 Method: transfer_file	34
11.3.3 Method: close	35
11.3.4 Method: quit	35
12 tcp_base.py	36
12.1 Class: TCPConfig	36
12.2 Class: TCPBase	36
12.2.1 Method: close	36
12.2.2 Method: quit	36
12.2.3 Method: connect	36
12.2.4 Method: disconnect	37
12.3 Class: TCPBaseServer	37

CONTENTSCONTENTS

12.3.1 Method: accept_connection	37
12.3.2 Method: connect	37
12.3.3 Method: disconnect	37
12.4 Class: TCPBaseClient	37
12.4.1 Method: connect	37
12.4.2 Method: disconnect	37
13 utils.py	38
13.1 Class: Singleton	38
13.2 Class: DictToClass	38
13.2.1 Method: validate	38
13.3 Class: Utils	38
13.3.1 Method: get_all_descendant_classes	38
13.3.2 Method: get_all_sub_classes	39
13.3.3 Method: is_valid_host	39
13.3.4 Method: execute_command	39
13.3.5 Method: kill_process	39
13.3.6 Method: caller_name	39
13.3.7 Method: load_library	39
13.3.8 Method: is_ascii_or_unicode	40
13.4 Class: Job	40
13.4.1 Method: stop	40
13.4.2 Method: run	40
13.5 Class: ResultType	40
13.6 Class: ResponseMessage	40
13.6.1 Method: get_json	40
13.6.2 Method: get_data	40
13.6.3 Method: create_from_string	40
14 Appendix	41
15 History	42

CHAPTER 1. INTRODUCTION

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 continuously 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 overflow 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_name]    conn_type=[conn_type]    conn_mode=[conn_mode]
conn_conf=[conn_conf]
```

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=[conn_name] command=[command] [argument name]=[argument value]

Arguments:

conn_name: Name of the connection.

command: Command to be sent.

2.2.4 transfer file

Use for transferring file from local to remote and vice versa. (Only available for SSHClient connection type)

Syntax:

transfer file conn_name=[conn_name] src=[source] dest=[destination] type=[transfer type]

Arguments:

conn_name: Name of the connection.
src: Your source file's path.
dest: The destination path on the remote side.
type: 'put' to send from local to remote, 'get' to bring it back.

2.2.5 verify

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

Syntax:

```
verify conn_name=[conn_name]    search_pattern=[search_pattern]  timeout=[timeout]
match_try=[match try time]   fetch_block=[is using fetchblock]  eob_pattern=[end
of block pattern] filter_pattern=[filter pattern]  send_cmd=[send
comand] [argument name]=[argument value]
```

Arguments:

conn_name: Name of the connection.
search_pattern: Regular expression for matching with the response.
timeout: Timeout for waiting response matching pattern.
match_try: Number of time for trying to match the 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}
```

CHAPTER 2. DESCRIPTION**2.4. AUXILIARY UTILITIES**

```

${config}=          evaluate      json.loads(''':''${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 ..

${res}=    Verify      conn_name=${CONNECTION_NAME}
...
send_cmd=echo ~~START~~;ls -la;echo ~~END$?~~\r\n
search_pattern=~~START~~\r\n([\\s\\S]*?)~~END(\\d+)~~
...
fetch_block=True
eob_pattern=~~END(\\d*)~~

Log To Console   ${res}[1]

# Disconnect
disconnect test_ssh

```

Listing 2.1: Robot code example

Explanation:

In the example above, we will establish an SSH connection to the remote host 127.0.0.1:8002 and then navigate back one directory level to list all files/folders within that directory.

To achieve this, we utilize the following steps:

1. First, we use the `send command` keyword, which is designed to send commands to the server and immediately return without waiting for a response. In this case, we send the command "cd .." to the remote host.
2. Next, we employ the `verify` keyword, specifically designed for verifying the response from the server after sending a command. The command sent is specified by the `send_cmd` argument (here, it is the "ls -la" command). We use a regular expression defined by the `search_pattern` argument to verify the response and extract the list of files/folders after executing the "ls -la" command.

However, by default, if no additional arguments are provided, only the first line of the response after running the command will be captured. To address this limitation, we introduce two additional arguments: `fetch_block` and `eob_pattern`.

When `fetch_block` is set to "True", the `Verify` keyword will capture all lines returned from the server until it encounters a line that matches the pattern defined by `eob_pattern` (End of Block pattern). This extended functionality ensures that we can effectively capture multi-line returns from the server.

To handle variable-length output, the `send.cmd` parameter encapsulates the command between "START" and "END\$?". By setting `fetch_block` to "True" and specifying the `eob_pattern`, we can accurately capture the entire list returned by the server between "START" and "END\$?".

2.4 Auxiliary Utilities

2.4.1 Introduction to RMQSignal

The `RMQSignal` class serves as an extension to the core functionalities, providing support for Inter-Process Communication (IPC) based on the RabbitMQ infrastructure. This utility module is designed to facilitate the exchange of signals accompanied by payloads between various processes, enhancing the overall communication mechanism.

2.4.2 Prerequisites

For Windows:

To use this library, installing RabbitMQ is required. For RabbitMQ installation on Windows, Erlang needs to be installed beforehand.

CHAPTER 2. DESCRIPTION2.4. AUXILIARY UTILITIES

For user convenience, we have created a .bat file to install all necessary components. Users simply need to run the file QConnectBase/tools/setup_rabbitmq.bat to install the entire infrastructure required for these Auxiliary Utilities.

For Ubuntu:

1. Update the system. First, ensure your system is up to date by running the command:

```
sudo apt update & sudo apt upgrade -y
```

2. Add the required repositories. Add the official RabbitMQ signing key and repository by running the following commands:

```
sudo apt install curl gnupg -y
curl -fsSL https://packages.rabbitmq.com/gpg | sudo apt-key add -
sudo add-apt-repository 'deb https://dl.bintray.com/rabbitmq/debian focal main'
```

3. Install RabbitMQ Server. Now, proceed with installing RabbitMQ:

```
sudo apt update && sudo apt install rabbitmq-server -y
```

4. Enable and start the RabbitMQ Service. After installation, enable and start the RabbitMQ service:

```
sudo systemctl enable rabbitmq-server
sudo systemctl start rabbitmq-server
```

2.4.3 Library Keywords and Usage

The keywords provided by this library offer a range of functionalities designed to streamline various tasks and processes. Below are some of the key functionalities along with their usage:

Send Signal Keyword

Usage: Used to send a signal to other processes.

Syntax: Send Signal [signal_name] [payloads] [Receiver]

Arguments:

- signal_name (string): Defines the name of the signal.
- payloads: Specifies the payloads of the signal.
- Receiver (optional, default is None): Specifies the receiver. If defined, the signal will be sent only to that specified receiver. If not defined, it sends a broadcast to all receivers.

Wait For Signal Keyword

Usage: Used to wait for a specific signal within a timeout.

Syntax: Wait For Signal [signal_name] [timeout]

Arguments:

- signal_name (string): Defines the name of the signal to wait for.
- timeout (optional, default is 10): Specifies the timeout in seconds to wait for the specific signal.

Return: Returns the payloads of the received signal, if received within the timeout.

Wait For Signals Keyword

Usage: Used to wait for multiple specific signals within a timeout.

Syntax: Wait For Signals [signal_names] [timeout]

Arguments:

- signal_names (list of strings): Defines the list of signal names to wait for.
- timeout (optional, default is 10): Specifies the timeout in seconds to wait for the full set of specific signals.

Return: Returns a list of payloads of the signals in the same order if the full set of signals is received within the timeout.

Set Signal Receiver Name Keyword

Usage: Used to set a specific name for the Signal Receiver, which defines a unique name for the current process. Other processes can send signals directly to the current process using this receiver name.

Syntax: Set Signal Receiver Name [receiver] [force]

Arguments:

- **receiver** (string): Specifies the name to set for the current process signal receiver.
- **force** (optional, default is True): Determines whether to force-create the signal receiver. If `force` is set to False, an exception will be raised if any process has already defined this name as a signal receiver.

Unset Signal Receiver Name Keyword

Usage: Used to unset the current Signal Receiver name of the current process.

Syntax: Unset Signal Receiver Name

2.4.4 Example of Inter-Process Communication

Content of ProcessA.robot file:

```
*** Settings ***
Library      QConnectBase.message.RMQSignal

*** Test Cases ***
Test signal
    Log To Console    Process A start
    Send Signal     HelloFromProcA    Hello, I'm A        receiverB
    Sleep   1s
    Send Signal     Hello2FromProcA   Hello 2nd time, I'm A      receiverB
    ${res}=  Wait For Signal   HelloFromProcB   ${20}
    Log To Console    Get resp signal from ProcB: ${res}
```

Content of ProcessB.robot file:

```
*** Settings ***
Library      QConnectBase.message.RMQSignal

*** Test Cases ***
Receiving Multiple Signals
    Log To Console    message
    Set Signal Receiver Name    receiverB
    ${list_of_signal}=  Create List    HelloFromProcA    Hello2FromProcAfull
    ${res}=  Wait For Signals   ${list_of_signal}   ${15}
    Sleep   2s
    Log To Console    Get signal from ProcA: ${res}
    Send Signal     HelloFromProcB    Hello A, I'm B. Got your signal:${res}
```

Explanation: In the above test cases, two separate Robot Framework test files, ProcessA.robot, and ProcessB.robot are used for inter-process communication.

In ProcessB.robot, the test case 'Receiving Multiple Signals' starts by setting up a signal receiver 'receiverB'. It then waits for a set of signals (HelloFromProcA and Hello2FromProcA) from ProcessA. Once both signals are received within 15 seconds, it logs to console the received signals' payloads and waits for 2 seconds before sending a response signal 'HelloFromProcB' to ProcessA.

In ProcessA.robot, the test case 'Sending Multiple Signals' sends HelloFromProcA signal to 'receiverB' in ProcessB. After a 1-second pause, it sends another signal 'Hello2FromProcA' to 'receiverB'. Then, it waits for a response, expecting signal 'HelloFromProcB' from ProcessB within 20 seconds. Upon receiving signal, it logs the received payloads to console.

2.5 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 protocol. 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 for receiving data byte by byte. You should implement the **thrd_llrecv_from_connection_interface()** method. This method is a immediate 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.

2.6 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.

2.7 Sourcecode Documentation

For investigating sourcecode, please refer to [QConnectBase library documentation](#)

2.8 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!

CHAPTER 2. DESCRIPTION2.9. ABOUT

2.9 About

2.9.1 Maintainers

Nguyen Huynh Tri Cuong

2.9.2 Contributors

Nguyen Huynh Tri Cuong

Thomas Pollerspöck

2.10 License

Copyright 2020-2023 Robert Bosch GmbH

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER 3. __INIT__.PY

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

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: get_connection_type

Get the connection type.

Returns:

/ Type: str /

The connection type.

4.2.4 Method: error_instruction

Get the error instruction.

Returns:

/ *Type*: str /

Error instruction string.

4.2.5 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*)

4.2.6 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 /

Determme if it's necessary for testing the connection.

Returns:

(*no returns*)

4.2.7 Method: disconnect

>> This method MUST be overridden in derived class <<

Abstract method for disconnecting connection.

Arguments:

- *device*

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

Device's name.

Returns:

(*no returns*)

4.2.8 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)

4.2.9 Method: read_obj

Wrapper method to get the response from connection.

Returns:

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

4.2.10 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:

CHAPTER 4. CONNECTION_BASE.PY4.2. CLASS: CONNECTIONBASE

- **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'.

4.2.11 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'.

4.2.12 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.

CHAPTER 4. CONNECTION_BASE.PY4.2. CLASS: CONNECTIONBASE

- `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

4.2.13 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)

4.2.14 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.

4.2.15 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.*

4.2.16 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)

4.2.17 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)

4.2.18 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

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

Making a connection.

Arguments:

- `conn_name`
/ Condition: optional / Type: str / Default: 'default_conn' /
Name of connection.
- `conn_type`
/ Condition: optional / Type: str / Default: 'TCPIP' /
Type of connection.
- `conn_mode`
/ Condition: optional / Type: str / Default: '' /
Connection mode.
- `conn_conf`
/ Condition: optional / Type: json / Default: {} /
Configuration for connection.

Returns:

(*no returns*)

5.5.7 Keyword: send_command

Send command to a connection.

Arguments:

- `connection_name`
/ Condition: required / Type: str /
Name of connection.
- `command`
/ Condition: required / Type: str /
Command to be sent.
- `kwargs`
/ Condition: optional / Type: dict / Default: {} /
Keyword Arguments.

Returns:

(*no returns*)

5.5.8 Keyword: transfer_file

Transfer file from local to remote and vice versa.

Arguments:

- `connection_name`
/ Condition: required / Type: str /

Name of connection.

- `src`
/ Condition: required / Type: str /

Source file path.

- `dest`
/ Condition: required / Type: str /

Destination file path.

- `type`
/ Condition: required / Type: str /

Transfer file type.

'get' - Copy a remote file from the SFTP server to the local host.

'put' - Copy a local file to the SFTP server.

Returns:

(no returns)

5.5.9 Keyword: verify

Verify a pattern from connection response after sending a command.

Arguments:

- `conn_name`
/ Condition: required / Type: str /

Name of connection.

- `search_pattern`
/ 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'.

- `timeout`
/ Condition: optional / Type: float / Default: 0 /

Timeout parameter specified as a floating point number in the unit 'seconds'.

- `match_try`
/ Condition: optional / Type: int / Default: 1 /

Number of time for trying to match the pattern.

- `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.

CHAPTER 5. CONNECTION_MANAGER.PY5.6. CLASS: TESTOPTION

- filter_pattern
 - / Condition: optional / Type: str / Default: '.*' /
 - Pattern to filter message line by line.
- send_cmd
 - / Condition: optional / Type: str / Default: '' /
 - Command to be sent.
- kwargs
 - / Condition: optional / Type: Dict / Default: None /
 - The optional arguments depend on the connection type used in the 'connect' keyword. Here are the supported options:

Connection Type		Argument		Explanation	
-----		-----		-----	
Winapp		element_def		Definition for detecting GUI item	
/		Type: str		Default: ''	/

Returns:

- match_res
 - / Type: str /
 - Matched string.

5.6 Class: TestOption

Imported by:

```
from QConnectBase.connection_manager import TestOption
```

CHAPTER 6. CONSTANTS.PY

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. RABBITMQ_CLIENT.PY

Chapter 7

rabbitmq_client.py

7.1 Class: RabbitmqClientConfig

Imported by:

```
from QConnectBase.message.rabbitmq_client import RabbitmqClientConfig
```

Class to store the configuration for SSH connection.

7.2 Class: RabbitmqClient

Imported by:

```
from QConnectBase.message.rabbitmq_client import RabbitmqClient
```

Rabbitmq client connection class.

7.2.1 Method: on_response

7.2.2 Method: connect

Implementation for creating a rabbitmq connection.

Returns:

(*no returns*)

7.2.3 Method: close

Close rabbitmq connection.

Returns:

(*no returns*)

7.2.4 Method: quit

Quit and stop receiver thread.

Returns:

(*no returns*)

7.3 Class: RMQSignal

Imported by:

```
from QConnectBase.message.rabbitmq.client import RMQSignal
```

RMQSignal class.

7.3.1 Method: send_signal

Send signal to other processes.

Arguments:

- **signal_name**
/ Condition: required / Type: str /
Signal to be sent.
- **payload**
/ Condition: required / Type: str /
Payloads for signal.
- **receiver**
/ Condition: optional / Type: str / Default: None /
Specify the signal receiver to send signal to.

Returns:

(no returns)

7.3.2 Method: unset_signal_receiver_name

Unset signal receiver.

Returns:

(no returns)

7.3.3 Method: set_signal_receiver_name

Set the signal receiver to be received signal.

Arguments:

- **receiver**
/ Condition: optional / Type: str / Default: '' /
Name a signal receiver to receive signal.
- **force**
/ Condition: optional / Type: bool / Default: True /
Force create the signal receiver (delete the exist signal receiver with the same name).

Returns:

(no returns)

7.3.4 Method: consume_channel

Consume the message from specific queue.

Arguments:

- **exchange**
/ Condition: required / Type: str /
Name of the exchange.

- `queue_name`
`/ Condition: required / Type: str /`
Name of the queue.
- `routing_key`
`/ Condition: required / Type: str /`
Routing key string.
- `stop_event`
`/ Condition: required / Type: Event /`
Event to notify stopping consumming.
- `signal_name`
`/ Condition: required / Type: str or list /`
Name of the signal to be wait for.
- `messages`
`/ Condition: required / Type: list or dict /`
Storage for received messages.
- `queue_delete`
`/ Condition: optional / Type: bool / Default: False /`
Determine if we should delete the queue at the end.

Returns:*(no returns)***7.3.5 Method: wait_for_signal**

Wait for specific signal in timeout.

Arguments:

- `signal_name`
`/ Condition: optional / Type: str /`
Name of the signal to wait for.
- `timeout`
`/ Condition: optional / Type: int / Default: 10 /`
Timeout for waiting the signal. Default is 10 seconds.

Returns:`/ Type: str /`

Payloads of the waiting signal if received.

7.3.6 Method: wait_for_signals

Wait for multiple specific signals in timeout.

Arguments:

- `signal_name`
`/ Condition: optional / Type: list /`
List of the signals to wait for.

CHAPTER 7. RABBITMQ_CLIENT.PY7.3. CLASS: RMQSIGNAL

- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 10 /
Timeout for waiting the signal. Default is 10 seconds.

Returns:

/ *Type*: str /
List of payloads of the waiting signals if received.

CHAPTER 8. QLOGGER.PY

Chapter 8

qlogger.py

8.1 Class: ColorFormatter

Imported by:

```
from QConnectBase.qlogger import ColorFormatter
```

Custom formatter class for setting log color.

8.1.1 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.

8.2 Class: QFileHandler

Imported by:

```
from QConnectBase.qlogger import QFileHandler
```

Handler class for user defined file in config.

8.2.1 Method: get_log_path

Get the log file path for this handler.

Arguments:

- config
/ *Condition:* required / *Type:* DictToClass /
Connection configurations.

Returns:

CHAPTER 8. QLOGGER.PY8.3. CLASS: QDEFAULTFILEHANDLER

/ *Type*: str /

Log file path.

8.2.2 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.

8.3 Class: QDefaultFileHandler

Imported by:

```
from QConnectBase.qlogger import QDefaultFileHandler
```

Handler class for default log file path.

8.3.1 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.

8.3.2 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.

8.4 Class: QConsoleHandler

Imported by:

```
from QConnectBase.qlogger import QConsoleHandler
```

Handler class for console log.

8.4.1 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.

8.5 Class: QLogger

Imported by:

```
from QConnectBase.qlogger import QLogger
```

Logger class for QConnect Libraries.

8.5.1 Method: get_logger

Get the logger object.

Arguments:

- `logger_name`
/ *Condition:* required / *Type:* str /
Name of the logger.

Returns:

- `logger`
/ *Type:* Logger /
Logger object. .

8.5.2 Method: set_handler

Set handler for logger.

Arguments:

- `config`
/ *Condition:* required / *Type:* DictToClass /
Connection configurations.

CHAPTER 8. QLOGGER.PY8.5. CLASS: QLOGGER**Returns:**

- `handler_ins`
/ *Type*: `logging.handler` /
None if no handler is set.
Handler object.

CHAPTER 9. SERIAL_BASE.PY

Chapter 9

serial_base.py

9.1 Class: SerialConfig

Imported by:

```
from QConnectBase.serialclient.serial_base import SerialConfig
```

Class to store the configuration for Serial connection.

9.2 Class: SerialSocket

Imported by:

```
from QConnectBase.serialclient.serial_base import SerialSocket
```

Class for handling serial connection.

9.2.1 Method: connect

Connect to serial port.

Returns:

(*no returns*)

9.2.2 Method: disconnect

Disconnect serial port.

Arguments:

- `_device`
/ *Condition:* required / *Type:* str /
Unused

Returns:

(*no returns*)

9.2.3 Method: quit

Quit serial connection.

Returns:

(*no returns*)

9.3 Class: SerialClient

Imported by:

```
from QConnectBase.serialclient.serial_base import SerialClient
```

Serial client class.

9.3.1 Method: connect

Connect to the Serial port.

Returns:

(no returns)

CHAPTER 10. RAW_TCP.PY

Chapter 10

raw_tcp.py

10.1 Class: RawTCPBase

Imported by:

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPBase
```

Base class for a raw tcp connection.

10.2 Class: RawTCPServer

Imported by:

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPServer
```

Class for a raw tcp connection server.

10.3 Class: RawTCPClient

Imported by:

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPClient
```

Class for a raw tcp connection client.

CHAPTER 11. SSH.CLIENT.PY

Chapter 11

ssh_client.py

11.1 Class: AuthenticationType

Imported by:

```
from QConnectBase.tcp.ssh.ssh_client import AuthenticationType
```

11.2 Class: SSHConfig

Imported by:

```
from QConnectBase.tcp.ssh.ssh_client import SSHConfig
```

Class to store the configuration for SSH connection.

11.3 Class: SSHClient

Imported by:

```
from QConnectBase.tcp.ssh.ssh_client import SSHClient
```

SSH client connection class.

11.3.1 Method: connect

Implementation for creating a SSH connection.

Returns:

(no returns)

11.3.2 Method: transfer_file

Transfer file from local to remote and vice versa.

Arguments:

- **connection_name**
/ *Condition:* required / *Type:* str /
Name of connection.
- **src**
/ *Condition:* required / *Type:* str /
Source file path.

- dest

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

Destination file path.

- type

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

Transfer file type.

'get' - Copy a remote file from the SFTP server to the local host

'put' - Copy a local file to the SFTP server

Returns:

(*no returns*)

11.3.3 Method: close

Close SSH connection.

Returns:

(*no returns*)

11.3.4 Method: quit

Quit and stop receiver thread.

Returns:

(*no returns*)

CHAPTER 12. TCP_BASE.PY

Chapter 12

tcp_base.py

12.1 Class: TCPConfig

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPConfig
```

Class to store configurations for TCP connection.

12.2 Class: TCPBase

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPBase
```

Base class for a tcp connection.

12.2.1 Method: close

Close connection.

Returns:

(*no returns*)

12.2.2 Method: quit

Quit connection.

Arguments:

- `is_disconnect_all`
/ *Condition:* required / *Type:* bool /
Determine if it's necessary for disconnect all connection.

Returns:

(*no returns*)

12.2.3 Method: connect

>> Should be override in derived class.

Establish the connection.

Returns:

(*no returns*)

12.2.4 Method: disconnect

>> Should be override in derived class.

Disconnect the connection.

Returns:

(*no returns*)

12.3 Class: TCPBaseServer

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPBaseServer
```

Base class for TCP server.

12.3.1 Method: accept_connection

Wrapper method for handling accept action of TCP Server.

Returns:

(*no returns*)

12.3.2 Method: connect

12.3.3 Method: disconnect

12.4 Class: TCPBaseClient

Imported by:

```
from QConnectBase.tcp.tcp_base import TCPBaseClient
```

Base class for TCP client.

12.4.1 Method: connect

12.4.2 Method: disconnect

CHAPTER 13. UTILS.PY

Chapter 13

utils.py

13.1 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.

13.2 Class: DictToClass

Imported by:

```
from QConnectBase.utils import DictToClass
```

Class for converting dictionary to class object.

13.2.1 Method: validate

13.3 Class: Utils

Imported by:

```
from QConnectBase.utils import Utils
```

Class to implement utilities for supporting development.

13.3.1 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.

13.3.2 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.

13.3.3 Method: is_valid_host**13.3.4 Method: execute_command****13.3.5 Method: kill_process****13.3.6 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

13.3.7 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.

13.3.8 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

13.4 Class: Job

Imported by:

```
from QConnectBase.utils import Job
```

13.4.1 Method: stop

13.4.2 Method: run

13.5 Class: ResultType

Imported by:

```
from QConnectBase.utils import ResultType
```

Result Types.

13.6 Class: ResponseMessage

Imported by:

```
from QConnectBase.utils import ResponseMessage
```

Response message class

13.6.1 Method: get_json

Convert response message to json

Returns:

Response message in json format

13.6.2 Method: get_data

Get string data result

Returns:

String result

13.6.3 Method: create_from_string

CHAPTER 14. APPENDIX

Chapter 14

Appendix

About this package:

Table 14.1: Package setup

Setup parameter	Value
Name	QConnectBase
Version	1.1.3
Date	06.06.2023
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 15. HISTORY

Chapter 15

History

1.1.0	07/2022
<i>Initial version</i>	

QConnectBase.pdf

*Created at 10.04.2024 - 12:33:47
by GenPackageDoc v. 0.41.1*

8.7 QConnectWinapp

QConnectWinapp

v. 1.0.3

Nguyen Huynh Tri Cuong

19.10.2023

Contents

1	Introduction	1
2	Description	2
2.1	Prerequisites	2
2.2	Getting Started	2
2.3	Usage	2
2.3.1	Configurations for Winapp connection type	2
2.3.2	Using the Automation Inspector Tool to define a GUI control	3
2.4	Example	7
2.5	Contribution Guidelines	9
2.6	Configure Git and correct EOL handling	9
2.7	Sourcecode Documentation	9
2.8	Feedback	10
2.9	About	10
2.9.1	Maintainers	10
2.9.2	Contributors	10
2.9.3	3rd Party Licenses	10
2.9.4	Used Encryption	10
2.9.5	License	10
3	element_handler.py	11
3.1	Class: ElementActionHandler	11
3.1.1	Method: get_supported_level	11
3.1.2	Method: get_attribute	11
3.1.3	Method: divert_action	11
4	winapp_client.py	13
4.1	Class: CustomWebDriver	13
4.1.1	Method: start_session	13
4.2	Class: WinappConfig	13
4.3	Class: WinAppClient	13
4.3.1	Method: connect	14
4.3.2	Method: perform_action	14
4.3.3	Method: send_obj	14
4.3.4	Method: wait_4_trace	14
4.3.5	Method: disconnect	15
4.3.6	Method: quit	15

CONTENTSCONTENTS

5 Appendix	16
6 History	17

CHAPTER 1. INTRODUCTION

Chapter 1

Introduction

QConnectWinapp is an extension library for [QConnectBase](#) library, designed to simplify and automate Windows application GUI testing. QConnectWinapp is built to provide seamless and efficient GUI testing experiences for Windows applications.

The library uses the powerful backend of WinAppDriver (<https://github.com/microsoft/WinAppDriver>) to drive the testing of your Windows applications, ensuring reliable and accurate results. QConnectWinapp is designed to support Python 3.7+, RobotFramework 3.2+, and QConnectBase 1.0.0+.

With QConnectWinapp, you can easily and quickly automate the testing of your Windows applications, saving time and effort while ensuring that your applications are of the highest quality. Whether you're a seasoned tester or just starting out, QConnectWinapp offers a range of features and tools to make GUI testing simple and efficient.

CHAPTER 2. DESCRIPTION

Chapter 2

Description

QConnectionWinapp

2.1 Prerequisites

To use the QConnectWinapp library, users need to install the following app prerequisites:

WinAppDriver: <https://github.com/Microsoft/WinAppDriver/releases> (version >=1.2.1)

Windows SDK: <https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/>

2.2 Getting Started

You can checkout all [QConnectWinapp](#) sourcecode from the GitHub.

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

```
python setup.py install
```

2.3 Usage

QConnectWinapp is a backend extension for the QConnectBase library that adds support for testing WinApp UI. From the user's perspective, this means they now have an additional connection type for WinApp testing when using the **connect** keyword in the QConnectBase library. With QConnectWinapp, users can now easily automate tests for Windows desktop applications, and take advantage of its integration with WinAppDriver to interact with UI elements and validate their behavior.

Please refer to [QConnectBase](#) for more information on how to use the **connect** keyword. In this section, we will focus on the **Winapp** connection type and how to configure it.

2.3.1 Configurations for Winapp connection type

QConnectBaseLibrary has already 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.
- **DLT**: Create a client connection to Diagnostic Log and Trace(DLT) Module.

QConnectWinapp add one more connection type for Winapp UI testing call **Winapp**

Below is the description of the configuration string for Winapp connection type:

```
{
    "host": [host ip],      # Optional. Default value is "localhost".
    "port": [listening port] # Optional. Default value is 4723.
    "caps": [Desired Capabilities string in JSON format], # E.g. { "app": "C:/Program Files/BOSCH/CMST/CMST.exe" }
    "logfile": [Log file path. Possible values: 'nonlog', 'console', [user define path]]
}
```

Table 2.1: List of Capabilities

Capability	Description
app	The package full name or the executable file's full path of the application to automate (e.g. Microsoft.WindowsCalculator_8wekyb3d8bbwe!App).
deviceName	Mostly optional, but the name of your device
automationName	The name of the automation engine to use (e.g. windows).

2.3.2 Using the Automation Inspector Tool to define a GUI control

The Automation Inspector tool provides a convenient way to interact with controls of an Application Under Test (AUT) and create resource files for testing. Follow the steps below to effectively utilize the tool:

Accessing the Tool

The Automation Inspector tool can be found at the following location: C:/Program Files/RobotFramework/python39/Lib/site-packages/QConnectWinapp/tools/bin/release.

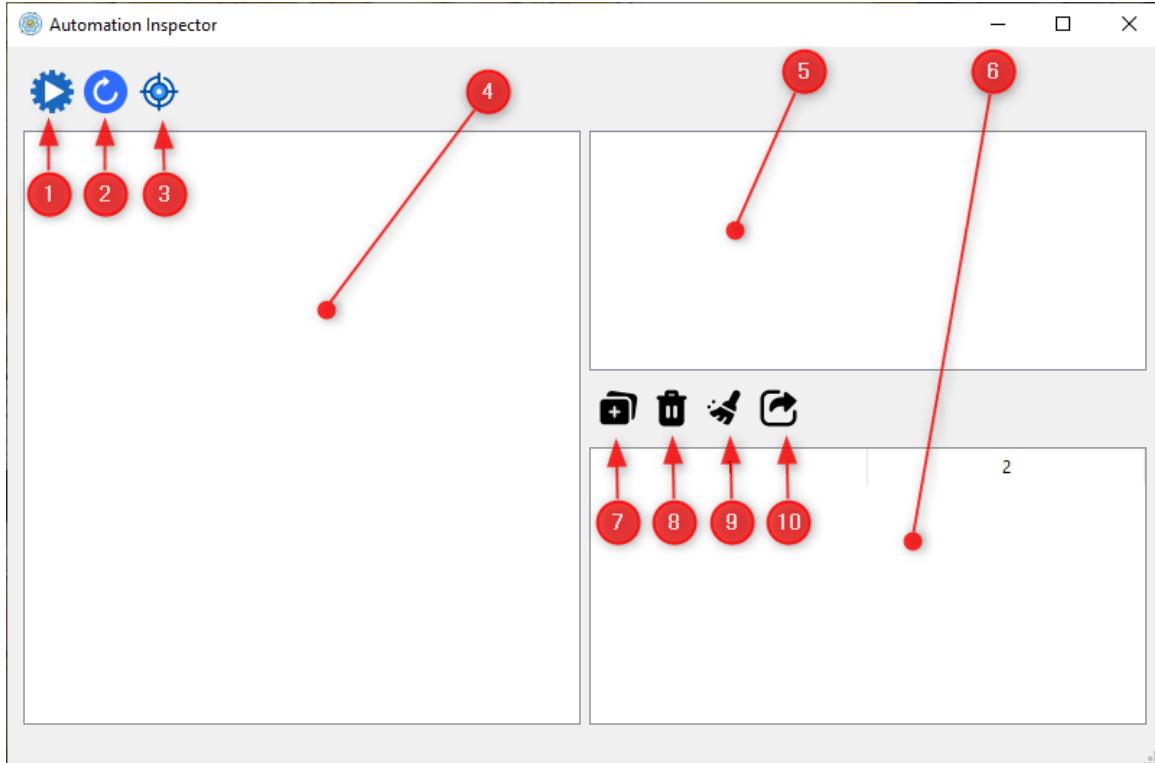


Figure 2.1: Automation Inspector tool.

Tool Overview

Upon launching the Automation Inspector tool, you will be presented with a graphical user interface that offers a range of functionalities. The main features (please refer to the Figure 2.1) of the tool are described below:

1. **Exec AUT button:** Press this button to start an Application Under Test (AUT).
2. **Refresh button:** Press this button to refresh the information of the AUT.
3. **Inspect controls button:** After pressing this button, users can hover their mouse over controls on the AUT. The tool will automatically highlight and display the attributes of the currently highlighted control.
4. **Elements treeview area:** This area displays the list of controls of the AUT in a hierarchical tree structure. Clicking on a node in the tree displays its attributes in the Properties table (5).
5. **Properties table:** This area displays the attributes of the node selected in the Elements treeview area (4) or highlighted using the Inspect controls feature (3). Users can check checkboxes to select attributes for defining the control to be tested and add them to the Definitions table (6) using the Add definition button (7).
6. **Definitions table:** This table contains the definitions of controls to be tested. Users can select attributes in the Properties table (5) and add them to this table using the Add definition button (7). Users can also check checkboxes to select definitions for exporting to a .resource file using the Export button (10).
7. **Add definition button:** Press this button to add the selected attributes from the Properties table (5) as definitions for a control in the Definitions table (6).
8. **Remove Definition button:** Press this button to remove the selected definition from the Definitions table (6).
9. **Clear Definitions button:** Press this button to remove all definitions from the Definitions table (6).
10. **Export button:** Export all selected definitions from the Definitions table (6) into a .resource file.

Creating a Resource File

To create a resource file using the Automation Inspector tool, follow these steps:

1. Launch the Automation Inspector tool from C:/Program Files/RobotFramework/python39/Lib/site-packages/QConnectWinapp/tools/bin/release.
2. Start the Application Under Test (AUT) by clicking the **Exec AUT** button. After clicking, a dialog will open, allowing the user to select the path to the executable file for the AUT. In the following example, I will input the path for the Calculator application.

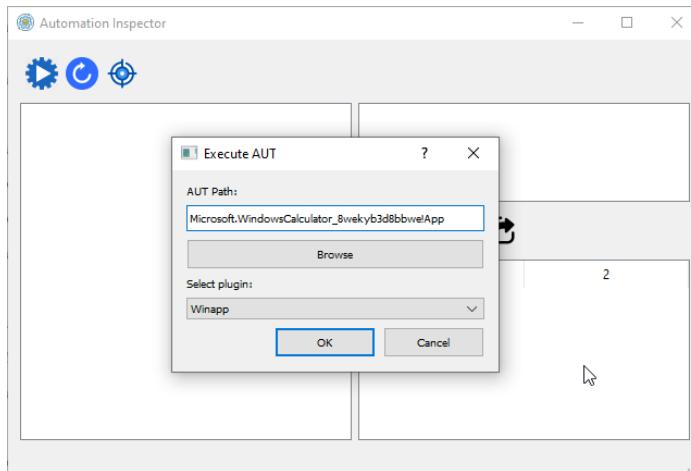


Figure 2.2: Select the AUT path to execute

CHAPTER 2. DESCRIPTION

2.3. USAGE

3. After selecting the AUT executable, click the **OK** button to start the AUT. The **Elements treeview** will be populated, showing a hierarchical tree of controls present in the AUT.
 4. To view the properties of a control, you can either:
 - Click on a node in the **Elements treeview** to display its properties in the **Properties table**.
 - Select the **Inspect controls** button and hover the mouse over a control in the AUT. The corresponding control in the **Elements treeview** will be selected and its properties will be shown in the **Properties table**.

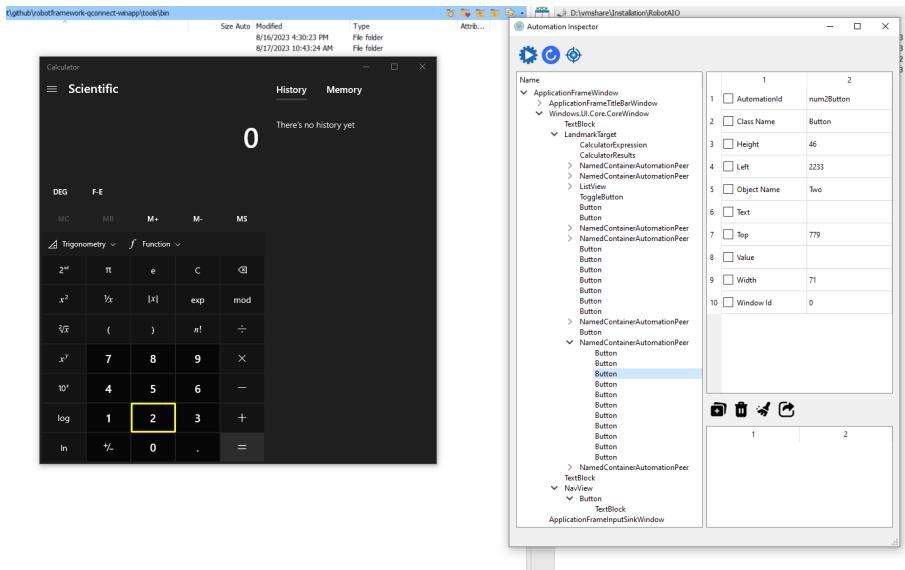


Figure 2.3: View control properties

5. Select the desired attributes for control definitions by checking the checkboxes in the **Properties** table.

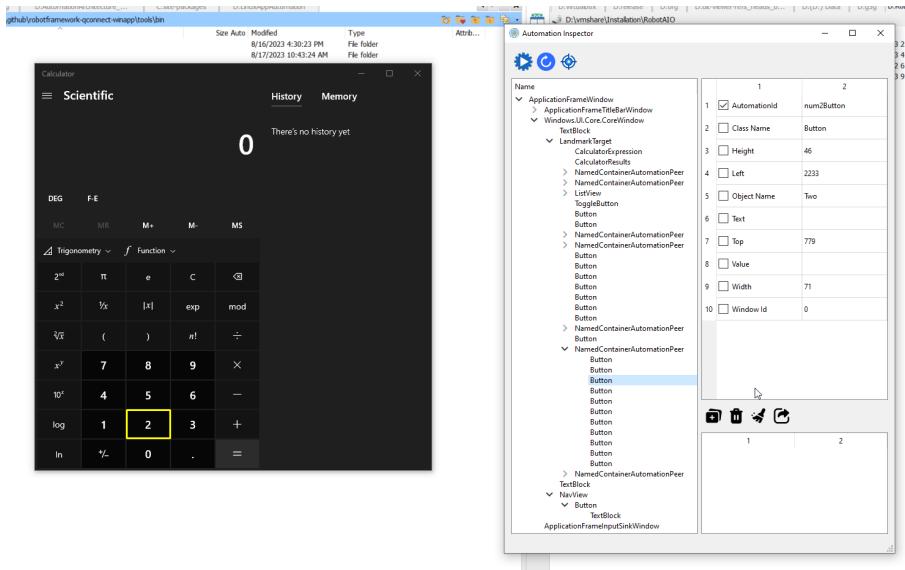


Figure 2.4: Select property for definition

CHAPTER 2. DESCRIPTION2.3. USAGE

6. Click the **Add definition** button to add the selected attributes to the **Definitions table**. After clicking the button, a dialog will appear for the user to declare a name corresponding with the definition. In the following example, I will fill the name as 'btn2Num', and the definition will be `{"AutomationId": "num2Button"}`, which means that the 'btn2Num' is a control with the AutomationId property value of 'num2Button'.

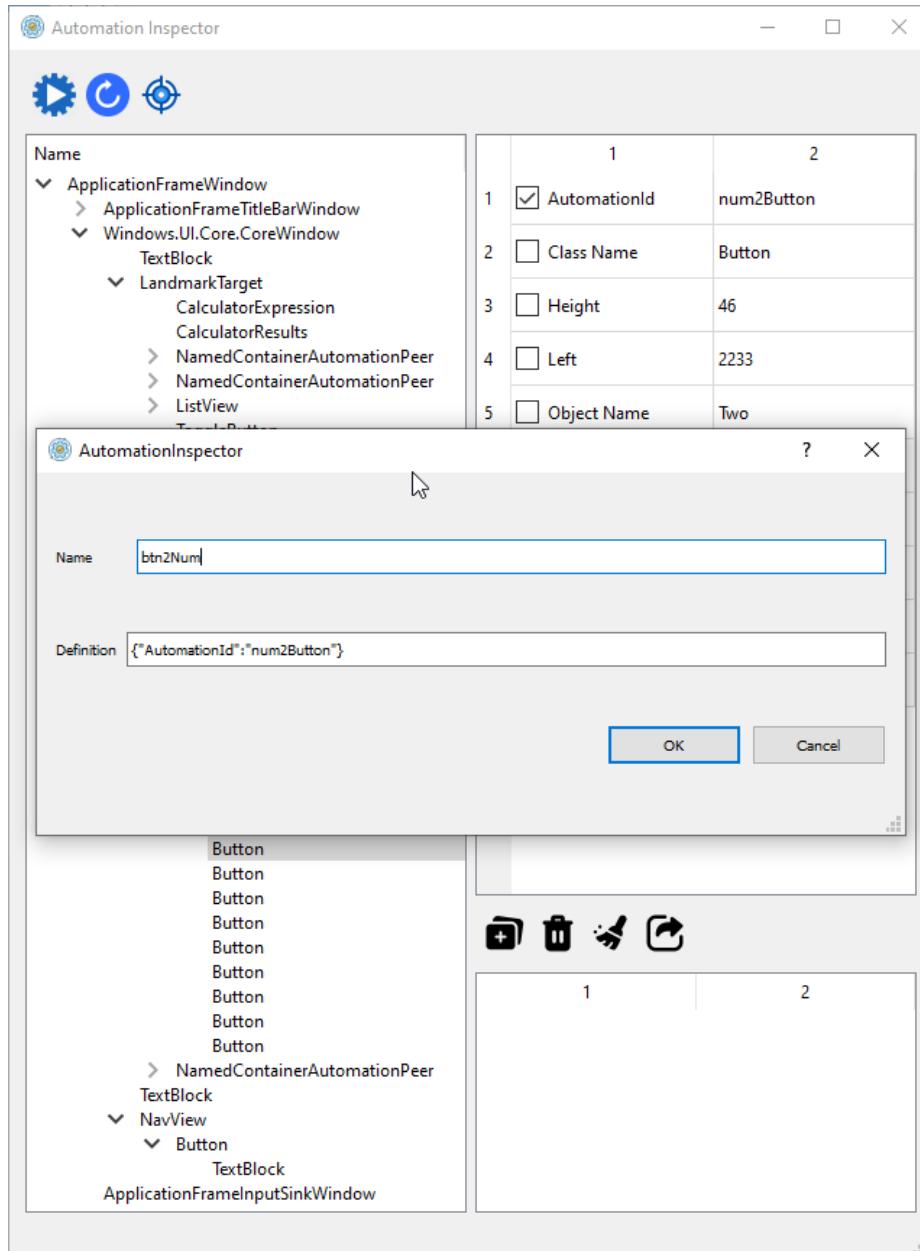


Figure 2.5: Declare a test automation control

7. Repeat the process to define multiple controls and their attributes.

CHAPTER 2. DESCRIPTION**2.4. EXAMPLE**

8. Use the **Export** button to export selected definitions to a .resource file.

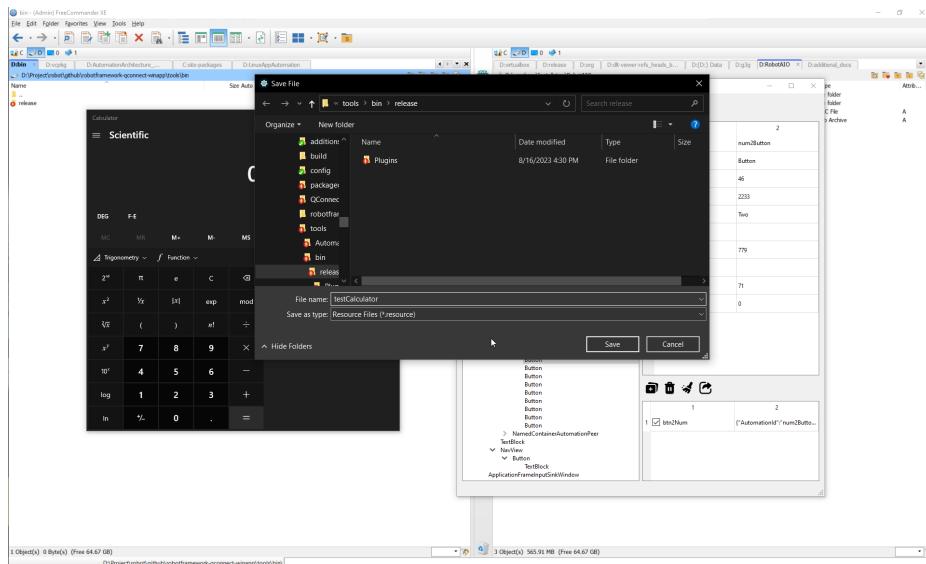


Figure 2.6: Export to resource file

By following these steps, you can effectively utilize the Automation Inspector tool to interact with controls, define attributes, and create resource files for testing purposes.

2.4 Example

In this example, I provide a test scenario that allows adding on the Calculator app as follows: the script will open the Calculator application on Windows, then press the '8' button, press the '+' button, press the '9' button, and then press the '=' button. Then, verify if the result textbox displays the result '17'.

```
*** Settings ***
Documentation    Suite description
Library        QConnectBase.ConnectionManager
Resource       QConnectWinapp/GUIAction.resource

*** Variables ***
${CONNECTION_NAME}  TEST_CONN

&{Number 8}      AutomationId=num8Button
&{Number 9}      AutomationId=num9Button
&{Equal}         AutomationId=equalButton
&{Result}        AutomationId=CalculatorResults
&{Plus}          AutomationId=plusButton

*** Test Cases ***
Test Adding
    ${config_string}=      catenate
    ...  \n
    ...      "host": "localhost", \n
    ...      "port": 4723, \n
    ...      "caps":\n    ...
    ...      "\n"
    ...      "app": "Microsoft.WindowsCalculator_8wekyb3d8bbwe!App"\n
    ...  }\n
    ...

    log to console  \nConnecting with below configurations:\n${config_string}
    ${config}=      evaluate      json.loads('${config_string}')      json
```

CHAPTER 2. DESCRIPTION2.4. EXAMPLE

```

connect           conn_name=${CONNECTION_NAME}
...
...
conn_type=Winapp
conn_conf=${config}

send command     conn_name=${CONNECTION_NAME}
...
...
element_def=${Number 8}
command=${Action.click}

Sleep   1s

send command     conn_name=${CONNECTION_NAME}
...
...
element_def=${Plus}
command=${Action.click}

Sleep   1s

send command     conn_name=${CONNECTION_NAME}
...
...
element_def=${Number 9}
command=${Action.click}

Sleep   1s

send command     conn_name=${CONNECTION_NAME}
...
...
element_def=${Equal}
command=${Action.click}

${res}= verify    conn_name=${CONNECTION_NAME}
...
...
element_def=${Result}
search_pattern=17
send_cmd=${Action.get_text}
timeout=20

log to console \nCalculation result: ${res}[0]

*** Keyword ***
Close Connection
disconnect  ${CONNECTION_NAME}

```

Explanation:

- &{Number 8} AutomationId=num8Button**: This line is used to identify the 'Number 8' button as the control on the application with an Accessible Name (AutomationId) of 'num8Button'.
- "app": "Microsoft.WindowsCalculator_8wekyb3d8bbwe!App"**: This line specifies that the AUT is the application with the package name 'Microsoft.WindowsCalculator_8wekyb3d8bbwe!App'. You can also use the full path to the exe file of the AUT.
- send command conn_name=\${CONNECTION_NAME}...command=\${Action.click}**: This line means to click on the Number 8 button on the AUT.
- `\${Action.click}`** has been defined in the resource file **QConnectWinapp/GUIAction.resource**.

Currently supported in the GUIAction.resource:

Action	Description
`\${Action.click}`	Clicks on a GUI element
`\${Action.get_text}`	Gets the text of a GUI element
`\${Action.get_visible}`	Get the visibility of a GUI element
`\${Action.get_enable}`	Check if a GUI element is enabled

Table 2.2: Actions supported in GUIAction.resource

2.5 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 protocol. 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.

2.6 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.

2.7 Sourcecode Documentation

For investigating sourcecode, please refer to [QConnectWinapp Documentation](#)

2.8 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. Connect with me at cuong.nguyenhuynhtri@vn.bosch.com.

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

2.9 About

2.9.1 Maintainers

[Nguyen Huynh Tri Cuong](#)

2.9.2 Contributors

[Nguyen Huynh Tri Cuong](#)

[Thomas Pollerspoeck](#)

2.9.3 3rd Party Licenses

You must mention all 3rd party licenses (e.g. OSS) licenses used by your project here. Example:

Name	License	Type
Apache Felix	Apache 2.0 License.	Dependency

2.9.4 Used Encryption

Declaration of the usage of any encryption (see BIOS Repository Policy §4.a).

2.9.5 License

[license BIOSL v4](#)

Copyright (c) 2009, 2018 Robert Bosch GmbH and its subsidiaries. This program and the accompanying materials are made available under the terms of the Bosch Internal Open Source License v4 which accompanies this distribution, and is available at <http://bios.intranet.bosch.com/bioslv4.txt>

CHAPTER 3. ELEMENT_HANDLER.PY

Chapter 3

element_handler.py

3.1 Class: ElementActionHandler

Imported by:

```
from QConnectWinapp.ActionHandlers.element_handler import ElementActionHandler
```

3.1.1 Method: get_supported_level

Get the supported level of this handler for a specific element object.

Arguments:

- ele_obj
/ *Condition:* required / *Type:* WebElement /
Winapp GUI element object.

Returns:

/ *Type:* int /
Supported level of this action handler for the element object.

3.1.2 Method: get_attribute

Get element object's property.

Arguments:

- attr
/ *Condition:* required / *Type:* str /
Property's name to be got value.

Returns:

/ *Type:* str /
Property's value.

3.1.3 Method: divert_action

Divert action string to the corresponding method, execute the method and return value.

Arguments:

CHAPTER 3. ELEMENT_HANDLER.PY3.1. CLASS: ELEMENTACTIONHANDLER

- **action**
/ Condition: required / Type: str /
Action string to be diverted.

Returns:

/ Type: str /
Corresponding method's return.

CHAPTER 4. WINAPP.CLIENT.PY

Chapter 4

winapp_client.py

4.1 Class: CustomWebDriver

Imported by:

```
from QConnectWinapp.WinappDriver.winapp_client import CustomWebDriver
```

Customer WebDriver class for Winapp.

4.1.1 Method: start_session

Creates a new session with the desired capabilities.

Override for Winapp

Arguments:

- `capabilities`
/ *Condition:* required / *Type:* Union /
Read <https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md> for more details.
- `browser_profile`
/ *Condition:* optional / *Type:* str / *Default:* None /
Browser profile

4.2 Class: WinappConfig

Imported by:

```
from QConnectWinapp.WinappDriver.winapp_client import WinappConfig
```

Class to store the configuration for SSH connection.

4.3 Class: WinAppClient

Imported by:

```
from QConnectWinapp.WinappDriver.winapp_client import WinAppClient
```

Winapp client connection class.

4.3.1 Method: connect

Connect to WinappDriver which is listening on configured port.

Returns:

(no returns)

4.3.2 Method: perform_action

Perform an action on user defined element.

Arguments:

- obj_defined_dict
/ Condition: required / Type: dict /
User's definition for a GUI element.
- cmd
/ Condition: required / Type: str /
Action to be perform on GUI element.
- time_wait
/ Condition: optional / Type: int / Default: 0 /
Timeout to find a GUI element based on user's definitions.

Returns:

/ Type: WebElement /

GUI element.

4.3.3 Method: send_obj

Action to be send to a GUI element.

Arguments:

- obj
/ Condition: required / Type: str /
Data to be sent.
- element_def
/ Condition: required / Type: dict /
User's definition for a GUI element.
- args
/ Condition: optional / Type: tuple /
Optional arguments.

Returns:

(no returns)

4.3.4 Method: wait_4_trace

Perform an action on GUI element and wait to receive a return which matches to a specified regular expression.

Arguments:

CHAPTER 4. WINAPP_CLIENT.PY4.3. CLASS: WINAPPCLIENT

- **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'.
- **element_def**
/ Condition: required / *Type:* dict /
 User's definition for a GUI element.
- **args**
/ Condition: optional / *Type:* tuple /
 Optional arguments.

Returns:

- **match**
/ Type: re.Match /
 If no return value matched to the specified regular expression and a timeout occurred, return None.
 If a return value 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'.

4.3.5 Method: disconnect

Abstract method for disconnecting connection.

Arguments:

- **_device**
/ Condition: required / *Type:* str /
 Unused.

Returns:

(no returns)

4.3.6 Method: quit

Quiting the connection.

Returns:

(no returns)

CHAPTER 5. APPENDIX

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	QConnectWinapp
Version	1.0.3
Date	19.10.2023
Description	Robot Framework QConnect library extension for Winapp GUI testing
Package URL	robotframework-qconnect-winapp
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 :: Microsoft :: Windows
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

CHAPTER 6. HISTORY

Chapter 6

History

1.0.0	07/2022
<i>Initial version</i>	

QConnectWinapp.pdf

*Created at 10.04.2024 - 12:33:50
by GenPackageDoc v. 0.41.1*

8.8 RobotFramework_DBus

RobotFramework_DBus

v. 0.1.3

Nguyen Huynh Tri Cuong

19.10.2023

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Key Features	1
2	Description	2
2.1	Prerequisites	2
2.2	Getting Started	2
2.3	Usage	2
2.3.1	connect	2
2.3.2	disconnect	3
2.3.3	set signal received handler	3
2.3.4	unset signal received handler	4
2.3.5	register signal	4
2.3.6	call dbus method	4
2.3.7	wait for signal	5
2.4	Remote testing	5
2.5	Example	5
2.5.1	Example 1 - Synchronized DBus Signal Waiting	6
2.5.2	Example 2 - Managing Multiple DBus Signals with 'Register Signal' Keyword	6
2.5.3	Example 3 - Non-blocking DBus Signal Handling with 'Set Signal Received Handler'	7
2.5.4	Example 4 - Invoking DBus Methods: Using 'Call Dbus Method' Keyword	8
2.5.5	Example 5 - Testing a Remote DBus Service: Setup and Steps	8
2.6	Configure Git and correct EOL handling	9
2.7	Feedback	9
2.8	About	9
2.8.1	Maintainers	9
2.8.2	Contributors	9
2.8.3	3rd Party Licenses	9
2.8.4	Used Encryption	9
2.8.5	License	9
3	__init__.py	10
3.1	Class: DBusManager	10
4	priority_queue.py	11
4.1	Class: PriorityQueue	11
4.1.1	Method: put	11

CONTENTS **CONTENTS**

5 register_keyword.py	12
5.1 Class: RegisterKeyword	12
5.1.1 Method: get_kw_name	12
5.1.2 Method: callback_func	12
6 scheduled_job.py	13
6.1 Class: ScheduledJob	13
6.1.1 Method: stop	13
6.1.2 Method: run	13
7 thread_safe_dict.py	14
7.1 Class: ThreadSafeDict	14
7.1.1 Method: clear	14
7.1.2 Method: pop	14
7.1.3 Method: popitem	14
7.1.4 Method: update	14
8 utils.py	16
8.1 Class: Singleton	16
8.2 Class: DictToClass	16
8.2.1 Method: validate	16
8.3 Class: Utils	16
8.3.1 Method: make_unique_token	16
8.3.2 Method: get_all_descendant_classes	17
8.3.3 Method: get_all_sub_classes	17
8.3.4 Method: caller_name	17
8.3.5 Method: load_library	18
8.3.6 Method: is_ascii_or_unicode	18
9 dbus_client_agent.py	19
9.1 Function: run_agent	19
9.2 Class: DBusClientExecutor	19
9.2.1 Method: connect	19
9.2.2 Method: disconnect	19
9.2.3 Method: quit	19
9.2.4 Method: get_monitoring_signal_payloads	20
9.2.5 Method: add_signal_to_captured_dict	20
9.2.6 Method: register_monitored_signal	20
9.2.7 Method: wait_for_signal	21
9.2.8 Method: call_dbus_method	21
9.3 Class: DBusClientAgent	21
9.3.1 Method: get_session_token	21
9.3.2 Method: initialize_dbus_client	22
9.3.3 Method: connect	22
9.3.4 Method: disconnect	22
9.3.5 Method: quit	22
9.3.6 Method: get_monitoring_signal_payloads	22

CONTENTSCONTENTS

9.3.7 Method: register_monitored_signal	23
9.3.8 Method: wait_for_signal	23
9.3.9 Method: call_dbus_method	24
10 dbus_client.py	25
10.1 Class: DBusClient	25
10.1.1 Method: connect	25
10.1.2 Method: disconnect	25
10.1.3 Method: quit	25
10.1.4 Method: set_signal_received_handler	25
10.1.5 Method: unset_signal_received_handler	26
10.1.6 Method: add_signal_to_captured_dict	26
10.1.7 Method: register_monitored_signal	26
10.1.8 Method: wait_for_signal	27
10.1.9 Method: call_dbus_method	27
10.1.10 Method: call_dbus_method_with_keyword_args	27
11 dbus_client_remote.py	28
11.1 Class: DBusClientRemote	28
11.1.1 Method: connect	28
11.1.2 Method: disconnect	28
11.1.3 Method: quit	28
11.1.4 Method: do_signal_check	28
11.1.5 Method: set_signal_received_handler	29
11.1.6 Method: unset_signal_received_handler	29
11.1.7 Method: register_monitored_signal	29
11.1.8 Method: wait_for_signal	29
11.1.9 Method: call_dbus_method	30
12 dbus_manager.py	31
12.1 Class: DBusManager	31
12.1.1 Method: quit	31
12.1.2 Method: add_connection	31
12.1.3 Method: remove_connection	31
12.1.4 Method: get_connection_by_name	32
12.1.5 Keyword: disconnect	32
12.1.6 Keyword: connect	32
12.1.7 Keyword: set_signal_received_handler	33
12.1.8 Keyword: unset_signal_received_handler	33
12.1.9 Keyword: register_signal	34
12.1.10 Keyword: call_dbus_method	34
12.1.11 Keyword: wait_for_signal	34
13 Appendix	36
14 History	37

CHAPTER 1. INTRODUCTION

Chapter 1

Introduction

1.1 Introduction

RobotFramework.DBus is a Python extension library for Robot Framework. It is designed to support the automation testing of DBus services. The library provides a set of keywords that enable testing of DBus methods and signals using both synchronous and asynchronous mechanisms.

Built upon the Python *dasbus* library, **RobotFramework.DBus** simplifies the process of interacting with DBus services and facilitates efficient testing of their functionality. It offers a comprehensive set of features to streamline the testing process and ensure the reliability of DBus-based applications.

1.2 Key Features

- Integration with Robot Framework: **RobotFramework.DBus** seamlessly integrates with Robot Framework, allowing you to leverage its powerful test automation capabilities for DBus services.
- DBus Service Automation: With **RobotFramework.DBus**, you can easily automate the testing of DBus services. It provides a range of keywords specifically designed for this purpose.
- Testing DBus Methods and Signals: The library enables you to test both DBus methods and signals. You can validate the behavior of DBus methods and monitor the emission of signals during the testing process.
- Synchronous and Asynchronous Testing: **RobotFramework.DBus** supports both synchronous and asynchronous testing approaches. You can choose the appropriate mechanism based on your testing requirements.

Whether you are testing a single DBus service or a complex system of interconnected services, **RobotFramework.DBus** offers a reliable and efficient solution for DBus automation testing. It empowers you to ensure the quality and robustness of your DBus-based applications.

Chapter 2

Description

2.1 Prerequisites

Before using **RobotFramework_DBus**, make sure you have the following libraries installed:

- **pycairo**: This library provides Python bindings for the Cairo graphics library. It is used for rendering graphics and creating visual elements in your applications.
- **PyGObject**: PyGObject is a Python package that provides bindings for the GObject library. It allows you to use GObject-based libraries, such as GTK+, in Python applications.
- **dasbus**: **RobotFramework_DBus** is built upon the *dasbus* library, which is a Pythonic D-Bus library. Make sure you have *dasbus* installed before using **RobotFramework_DBus**.
- **pyinstaller**: PyInstaller is a tool used to package Python applications into standalone executables. If you plan to distribute your application as an executable, you will need to have *pyinstaller* installed.

Ensure that the above libraries are installed and properly configured in your Python environment before using **robotframework-dbus**. This will ensure the smooth functioning and compatibility of the library with your system.

2.2 Getting Started

You can checkout all **robotframework-dbus** sourcecode from the GitHub.

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

```
python setup.py install
```

2.3 Usage

RobotFramework_DBus Library support following keywords for testing connection in RobotFramework.

2.3.1 connect

Use for establishing a connection.

Syntax:

```
connect      conn_name=[connection name]
...          namespace=[namespace]
...          object_path=[object path]
...          mode=[test mode]
...          host=[remote host]
...          port=[remote port]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

namespace: The namespace of the DBus service. This identifies the specific service or group of services. It is used to differentiate between different service instances. The namespace should be a string that uniquely identifies the service.

E.g. namespace=org.example.HelloWorld

object_path: The object path of the DBus service. This identifies the specific object within the service that the action will be performed on. The object path should be a string that follows the DBus object path naming convention. It typically consists of a hierarchical structure separated by slashes (/).

E.g. object-path=/org/example/HelloWorld

mode: The mode of testing the DBus service. Possible values are 'local' or 'remote'. 'local' indicates testing on the current system, while 'remote' indicates testing on a remote system. Default is 'local'.

host: The IP address or hostname of the remote system where the DBus agent is running. This parameter is applicable only if 'mode' is set to 'remote'.

Default is 'localhost'.

2.3.2 disconnect

Use for disconnecting from the DBus service by connection name.

Syntax:

disconnect	conn_name
------------	-----------

Arguments:

conn_name: The name or identifier of the connection instance to disconnect from. This parameter is optional and can be used to specify a specific connection to disconnect. If the connection name is 'ALL', all connections will be disconnected.

2.3.3 set signal received handler

Use to set a signal received handler for a specific DBus connection and signal.

Syntax:

set signal received handler	conn_name=[conn_name]
...	signal=[signal_name]
...	handler=[keyword handles signal emitted event]

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal to be set emitted handler.

handler: The robotframework keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

2.3.4 unset signal received handler

Use to unset a signal received handler for a specific signal.

Syntax:

```
unset signal received handler conn name=[conn name]
...
...
signal=[signal name]
handler=[keyword handles signal emitted event]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal to be unset emitted handler.

handler: The robotframework keyword which is handling the signal emitted event.

2.3.5 register signal

Use to register a DBus signal or signals to be monitored for a specific connection.

Syntax:

```
register signal conn name=[conn name]
...
signal=[signal name]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

2.3.6 call dbus method

Use to call a DBus method with the specified method name and input arguments.

Syntax:

```
call dbus method [conn name] [method name] [args]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

method_name: The name of the DBus method to be called.

args: Input arguments to be passed to the method.

Return value:

Return from called method.

2.3.7 wait for signal

Use to wait for a specific DBus signal to be received within a specified timeout period.

Syntax:

```
wait for signal      conn name=[conn name]
...                  signal=[signal name]
...                  timeout=[timeout]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal to wait for.

timeout: The maximum time (in seconds) to wait for the signal.

Return value:

The signal payloads.

2.4 Remote testing

After installing the library, you can find the DBus Client Agent at `/opt/rfwaio/python39/install/bin/`.

For remote testing, follow these steps:

1. Start the DBus Agent on the remote system by the following command:

`dbus_client_agent [-h] [-host HOST] [-port PORT]`

The DBus Client Agent supports the following command-line arguments:

`--host` (str, optional) The host where the agent is running. Default is `0.0.0.0`.
`--port` (int, optional) The port where the agent is listening. Default is `2507`.

2. On the host test PC, using the `connect` keyword with the `remote` mode and specify the correct host using the `host` parameter.
3. Use the other keywords in the same way as local testing.

2.5 Example

Kindly be advised that all the examples presented within this session are designed to interact with the DBus service sample server.py residing in the atest/ directory.

2.5.1 Example 1 - Synchronized DBus Signal Waiting

Scenario: In this example, I will use the `wait for signal` keyword from the RobotFramework_DBus library to wait for a DBus signal to be emitted using the synchronized mechanism. This means that the keyword following `wait for signal` will only be executed after the signal is emitted, and the `wait for signal` keyword will return a value.

```
*** Settings ***
Library    RobotFramework_DBus.DBusManager

*** Test Cases ***
Hello World
  connect    conn_name=test_dbus
  ...
  ...      namespace=org.example.HelloWorld
  ...      mode=local

  ${ret}=  Wait For Signal    conn_name=test_dbus
  ...
  ...      signal=YellowMessage
  ...      timeout=10

  Log To Console    ${ret}

  Disconnect    test_dbus
```

Explanation: In the aforementioned example, we establish a connection to the DBus service `org.example.HelloWorld` using the '`connect`' keyword and name this connection '`test_dbus`'. Subsequently, the '`Wait for signal`' keyword will wait for the DBus service to emit the `YellowMessage` signal within a 10-second timeframe and return the payloads of this signal immediately upon its emission within the specified timeframe (failing if the timeout is exceeded). The content of the payloads will be printed to the console.

2.5.2 Example 2 - Managing Multiple DBus Signals with 'Register Signal' Keyword

Scenario: In Example 1, if we include an additional `Wait for signal` keyword to wait for the `GreenMessage` signal to be emitted after waiting for `YellowMessage`, there is a possibility of missing the occurrence of `GreenMessage` if it is emitted before `YellowMessage`. In such a scenario, we can utilize the `Register Signal` keyword to register `GreenMessage` in the watchlist. This ensures continuous monitoring of the signal starting immediately after executing this keyword and prevents the occurrence of missing the emitted `GreenMessage` event.

```
*** Settings ***
Library    RobotFramework_DBus.DBusManager

*** Test Cases ***
Hello World
  connect    conn_name=test_dbus
  ...
  ...      namespace=org.example.HelloWorld
  ...      mode=local

  Register Signal    conn_name=test_dbus    signal=GreenMessage

  ${ret}=  Wait For Signal    conn_name=test_dbus
  ...
  ...      signal=YellowMessage
  ...      timeout=10
  Log To Console    ${ret}

  ${ret}=  Wait For Signal    conn_name=test_dbus
  ...
  ...      signal=GreenMessage
  ...      timeout=10
  Log To Console    ${ret}

  ${ret}=  Call Dbus Method    test_dbus    Hello    World
  Disconnect    test_dbus
```

Explanation:

In the above example, by using the `Register Signal` keyword, we have registered the `GreenMessage` signal in the watchlist. If the `GreenMessage` event is emitted during the `Wait for signal YellowMessage` timeframe, it will be

CHAPTER 2. DESCRIPTION2.5. EXAMPLE

recorded. Then, when we execute the `Wait for signal` keyword for **GreenMessage**, it will immediately return because the **GreenMessage** event has already occurred.

2.5.3 Example 3 - Non-blocking DBus Signal Handling with 'Set Signal Received Handler'

Scenario: In the two examples above, the testcase is blocked at the `Wait for Signal` keyword until the signal event is emitted. In the next example, we will explore the usage of the `Set Signal Received Handler` keyword. This is a non-blocking keyword that allows us to register a user-defined keyword, such as a callback function. The registered keyword will be called when the emitted signal event occurs. The testcase will continue with the next keyword immediately after the `Set Signal Received Handler` keyword, without waiting.

```
*** Settings ***
Library     RobotFramework_DBus.DBusManager
Library     test/dbus/client.py

*** Test Cases ***
Hello World
    connect      conn_name=test_dbus
    ...          namespace=org.example.HelloWorld
    ...          mode=local

    Set Signal Received Handler      conn_name=test_dbus
    ...          signal=RedMessage
    ...          handler=On Received Red Signal

    Log To Console      The test is continuing...

    Sleep      10s

    Disconnect      test_dbus

*** Keyword ***
On Received Red Signal
[Arguments]  ${arg1}=default 1
    log to console      Client received red signal. Payload: ${arg1}
    Unset Signal Received Handler      conn_name=test_dbus      signal=RedMessage      ↫
        ↪ handler=On Received Red Signal
```

Explanation:

In the above example, by using the `Set Signal Received Handler` keyword, we have registered the user-defined keyword `On Received Red Signal` as a callback function. When the **RedMessage** is emitted, this keyword will be executed. During the time when the **RedMessage** has not been emitted yet, the testcase will continue running the subsequent keywords as usual. It is important to note that the user-defined keyword should have an argument to handle the payload of the signal. If there is no longer a need to invoke the callback function, we should unregister the signal handler using the `Unset Signal Received Handler` keyword.

2.5.4 Example 4 - Invoking DBus Methods: Using 'Call Dbus Method' Keyword

Scenario:

In the next example, we will explore how to use the `Call Dbus Method` keyword to invoke a method in the DBus service and retrieve the return value from this method.

```
*** Settings ***
Library    RobotFramework_DBus.DBusManager
Library    test/dbus/client.py

*** Test Cases ***
Hello World
  connect      conn_name=test_dbus
  ...          namespace=org.example.HelloWorld
  ...          mode=local

  ${ret}=   Call Dbus Method    test_dbus    Hello    World
  Log To Console    ${ret}

  Disconnect    test_dbus
```

2.5.5 Example 5 - Testing a Remote DBus Service: Setup and Steps

Scenario: In the next example, we will explore how to test a DBus service on a different PC. To be able to test on a remote system, we need to perform two steps:

1. On the System Under Test (SUT), run the following command:

`dbus_client_agent 0.0.0.0 2507`

The DBus Client Agent supports the following command-line arguments:

- host (str, optional) The host where the agent is running. Default is 0.0.0.0.
- port (int, optional) The port where the agent is listening. Default is 2507.

2. On the test PC, make slight modifications to the keyword's connect parameters as follows:

```
*** Settings ***
Library    RobotFramework_DBus.DBusManager
Library    test/dbus/client.py

*** Test Cases ***
Hello World
  connect      conn_name=test_dbus
  ...          namespace=org.example.HelloWorld
  ...          mode=remote
  ...          host=172.17.0.2
  ...          port=2507

  Set Signal Received Handler    conn_name=test_dbus
  ...                          signal=RedMessage
  ...                          handler=On Received Red Signal

  Log To Console    The test is continuing...

  Sleep    10s

  Disconnect    test_dbus

*** Keyword ***
On Received Red Signal
  [Arguments]  ${arg1}=default 1
  log to console    Client received red signal. Payload: ${arg1}
  Unset Signal Received Handler    conn_name=test_dbus    signal=RedMessage    ↵
  ↵ handler=On Received Red Signal
```

Explanation:

In the above example, we set the value of the mode parameter for the connect keyword to remote, and add two parameters: host and port. The host parameter is set to the IP address of the System Under Test (SUT), which is 172.17.0.2, and the port parameter is set to 2507, which is the port that the agent is listening on.

2.6 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 assure that all EOL are managed by git correctly as defined.

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. Connect with me at cuong.nguyenhuynhtri@vn.bosch.com.

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 Pollerspoeck](#)

2.8.3 3rd Party Licenses

You must mention all 3rd party licenses (e.g. OSS) licenses used by your project here. Example:

Name	License	Type
Apache Felix	Apache 2.0 License .	Dependency

2.8.4 Used Encryption

Declaration of the usage of any encryption (see BIOS Repository Policy §4.a).

2.8.5 License

Copyright (c) 2009, 2018 Robert Bosch GmbH and its subsidiaries. This program and the accompanying materials are made available under the terms of the Bosch Internal Open Source License v4 which accompanies this distribution, and is available at <http://bios.intranet.bosch.com/bioslv4.txt>

CHAPTER 3. __INIT__.PY

Chapter 3

__init__.py

3.1 Class: DBusManager

Imported by:

```
from RobotFramework_DBus.__init__ import DBusManager
```

Class to manage all dbus communications.

CHAPTER 4. PRIORITY_QUEUE.PY

Chapter 4

priority_queue.py

4.1 Class: PriorityQueue

Imported by:

```
from RobotFramework_DBus.common.priority_queue import PriorityQueue
```

4.1.1 Method: put

CHAPTER 5. REGISTER_KEYWORD.PY

Chapter 5

register_keyword.py

5.1 Class: RegisterKeyword

Imported by:

```
from RobotFramework_DBus.common.register_keyword import RegisterKeyword
```

A class that provides a keyword as a callback function for a DBus signal received.

5.1.1 Method: get_kw_name

Get the handle keyword name.

Returns:

/ *Type*: str /

The handle keyword name.

5.1.2 Method: callback_func

Constructor for RegisterKeyword class.

Arguments:

- *observer*

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

Input arguments to be passed to the callback method.

Returns:

(no returns)

CHAPTER 6. SCHEDULED_JOB.PY

Chapter 6

scheduled_job.py

6.1 Class: ScheduledJob

Imported by:

```
from RobotFramework_DBus.common.scheduled_job import ScheduledJob
```

A threaded job that executes a function at a specified interval.

6.1.1 Method: stop

Stop the execution of the job.

Returns:

(*no returns*)

6.1.2 Method: run

Start the job execution loop.

Returns:

(*no returns*)

CHAPTER 7. THREAD_SAFE_DICT.PY

Chapter 7

thread_safe_dict.py

7.1 Class: ThreadSafeDict

Imported by:

```
from RobotFramework_DBus.common.thread_safe_dict import ThreadSafeDict
```

This class provides a dictionary with thread-safe operations by utilizing locks for synchronized access.

7.1.1 Method: clear

Remove all key-value pairs from the dictionary.

Returns:

(*no returns*)

7.1.2 Method: pop

Remove and return the value associated with a given key.

Arguments:

- **key**
/ *Condition: required / Type: Any /*
The key of dictionary to be pop.

Returns:

- **o**
/ *Type: Any /*
Value of the given key.

7.1.3 Method: popitem

Remove and return an arbitrary key-value pair from the dictionary.

Returns:

(*no returns*)

7.1.4 Method: update

Update the dictionary with key-value pairs from another dictionary.

Arguments:

CHAPTER 7. THREAD_SAFE.DICT.PY7.1. CLASS: THREADSAFEDICT

- `dict`
/ *Condition*: optional / *Type*: dict / *Default*: None /
Another dictionary-like object to update from.

Returns:

(no returns)

CHAPTER 8. UTILS.PY

Chapter 8

utils.py

8.1 Class: Singleton

Imported by:

```
from RobotFramework_DBus.common.utils import Singleton
```

Class to implement Singleton Design Pattern. This class is used to derive the DBusManager as only a single instance of this class is allowed.

8.2 Class: DictToClass

Imported by:

```
from RobotFramework_DBus.common.utils import DictToClass
```

Class for converting dictionary to class object.

8.2.1 Method: validate

8.3 Class: Utils

Imported by:

```
from RobotFramework_DBus.common.utils import Utils
```

Class to implement utilities for supporting development.

8.3.1 Method: make_unique_token

Generates a unique session token of specified length.

The make_unique_token function generates a unique session token of the specified length. The session token can be used to identify and associate a session with a specific client.

The session token is a string value that is guaranteed to be unique for each invocation of this function. It can be used as a secure identifier to track and manage client sessions within the DBusAgent.

Arguments:

- `length`
/ *Condition:* optional / *Type:* int / *Default:* 16 /
The length of the session token. Defaults to 16.

CHAPTER 8. UTILS.PY8.3. CLASS: UTILS**Returns:**

- `token`
/ *Type*: str /
A unique token.

8.3.2 Method: get_all_descendant_classes

Get all descendant classes of a class

Arguments:

- `cls`
/ *Condition*: required / *Type*: class /
Input class for finding children.

Returns:

- / *Type*: list /
Array of descendant classes.

8.3.3 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.

8.3.4 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

8.3.5 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.

8.3.6 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

CHAPTER 9. DBUS_CLIENT_AGENT.PY

Chapter 9

dbus_client_agent.py

9.1 Function: run_agent

Run the DBus Agent with the specified configuration.

Description:

The run_agent function starts the DBus Agent with the provided configuration. It parses the command-line arguments, such as the host and port options, and then starts the agent on the specified host and port.

The agent listens for incoming requests from clients and manages client sessions. It provides a mechanism to execute requests on corresponding DBus services through the assigned Executor instances.

Command-line Arguments:

--host (str, optional): The host where the agent is running. Default is '0.0.0.0'.

--port (int, optional): The port where the agent is listening. Default is 2507.

9.2 Class: DBusClientExecutor

Imported by:

```
from RobotFramework_DBus.dbus_agent.dbus_client_agent import DBusClientExecutor
```

The DBusClientExecutor class represents an executor responsible for handling client requests on specific DBus services. It receives requests from the DBusAgent and executes them on the corresponding DBus service.

9.2.1 Method: connect

Create a proxy object to DBus object.

Returns:

(*no returns*)

9.2.2 Method: disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(*no returns*)

9.2.3 Method: quit

Quit the DBus client.

Returns:*(no returns)***9.2.4 Method: get_monitoring_signal_payloads**

Get the payloads of a specific signal.

Arguments:

- **signal**
/ Condition: required / Type: str /
The name of the DBus signal to get payloads.

Returns:

- **payloads**
/ Type: str /
The signal's payloads.

9.2.5 Method: add_signal_to_captured_dict

Add a signal and its payloads to the captured dictionary when the signal be emitted.

Arguments:

- **signal**
/ Condition: required / Type: str /
The name of the DBus signal(s) which has been raised.
- **loop**
/ Condition: optional / Type: EventLoop / Default: None /
The Event loop which is running to wait for the raised signal.
- **payloads**
/ Condition: optional / Type: Any / Default: "" /
The payloads of the raised signal.

Returns:*(no returns)***9.2.6 Method: register_monitored_signal**

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- **signal**
/ Condition: optional / Type: str / Default: "" /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:*(no returns)*

9.2.7 Method: wait_for_signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `wait_signal`
`/ Condition: optional / Type: str / Default: '' /`
The name of the DBus signal to wait for.
- `timeout`
`/ Condition: optional / Type: int / Default: 0 /`
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
`/ Type: str /`
The signal payloads.

9.2.8 Method: call_dbus_method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
`/ Condition: optional / Type: str / Default: '' /`
The name of the DBus method to be called.
- `args`
`/ Condition: optional / Type: tuple / Default: None /`
Input arguments to be passed to the method.

Returns:

- `mtd_ret`
`/ Type: Any /`
Return from called method.

9.3 Class: DBusClientAgent

Imported by:

```
from RobotFramework.DBus.dbus_agent.dbus_client_agent import DBusClientAgent
```

The DBusClientAgent class acts as a mediator between clients and the corresponding DBus services they request. It manages client connections, session tokens, and assigns the appropriate DBusClientExecutor for each client session.

9.3.1 Method: get_session_token

Generates a unique session token for a client and returns it.

Returns:

`/ Type: str /`

The random and unique token.

9.3.2 Method: initialize_dbus_client

Initializes an DBusClientExecutor instance for a specific client.

An DBusClientExecutor is an object responsible for executing requests from clients on corresponding DBus services. By initializing a separate DBusClientExecutor for each client, the DBusClientAgent ensures that requests from different clients are handled independently.

The client session token is a unique identifier that can be used to associate the DBusClientExecutor with the specific client. This allows the DBusClientAgent to route incoming requests to the correct DBusClientExecutor based on the client session token.

Arguments:

- `session`

/ Condition: required / Type: str /

The client's session token.

- `namespace`

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

The namespace of the DBus service. This identifies the specific service or group of services. It is used to differentiate between different service instances. The namespace should be a string that uniquely identifies the service.

- `object_path`

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

The object path of the DBus service. This identifies the specific object within the service that the action will be performed on. The object path should be a string that follows the DBus object path naming convention. It typically consists of a hierarchical structure separated by slashes (/).

Returns:

(*no returns*)

9.3.3 Method: connect

Create a proxy object to DBus service.

Returns:

(*no returns*)

9.3.4 Method: disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(*no returns*)

9.3.5 Method: quit

Quit the DBus client.

Returns:

(*no returns*)

9.3.6 Method: get_monitoring_signal_payloads

Get the payloads of a specific signal.

Arguments:

CHAPTER 9. DBUS_CLIENT_AGENT.PY9.3. CLASS: DBUSCLIENTAGENT

- **session**
/ Condition: required / Type: str /
The client's session token.
- **signal**
/ Condition: required / Type: str /
The name of the DBus signal to get payloads.

Returns:

- **payloads**
/ Type: str /
The signal's payloads.

9.3.7 Method: register_monitored_signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- **session**
/ Condition: required / Type: str /
The client's session token.
- **signal**
/ Condition: required / Type: str /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

9.3.8 Method: wait_for_signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- **session**
/ Condition: required / Type: str /
The client's session token.
- **wait_signal**
/ Condition: optional / Type: str / Default: '' /
The name of the DBus signal to wait for.
- **timeout**
/ Condition: optional / Type: int / Default: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- **payloads**
/ Type: str /
The signal payloads.

9.3.9 Method: call_dbus_method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `session`
/ Condition: required / Type: str /
The client's session token.
- `method_name`
/ Condition: optional / Type: str / Default: '' /
The name of the DBus method to be called.
- `args`
/ Condition: optional / Type: tuple / Default: None /
Input arguments to be passed to the method.

Returns:

/ Type: Any /
Return from called method.

Chapter 10

dbus_client.py

10.1 Class: DBusClient

Imported by:

```
from RobotFramework_DBus.dbus_client import DBusClient
```

A client class for interacting with a specific DBus service.

10.1.1 Method: connect

Create a proxy object to DBus object.

Returns:

(*no returns*)

10.1.2 Method: disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(*no returns*)

10.1.3 Method: quit

Quit the DBus client.

Returns:

(*no returns*)

10.1.4 Method: set_signal_received_handler

Set a signal received handler for a specific signal.

Arguments:

- **signal**

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

The name of the DBus signal to handle.

- **handler**

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

The keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

Returns:

(no returns)

10.1.5 Method: unset_signal_received_handler

Unset a signal received handler for a specific signal.

Arguments:

- **signal**
/ Condition: required / *Type:* str /
The name of the DBus signal to handle.
- **handle_keyword**
/ Condition: optional / *Type:* str / *Type:* None /
The keyword which is handling for signal emitted event.

Returns:

(no returns)

10.1.6 Method: add_signal_to_captured_dict

Add a signal and its payloads to the captured dictionary when the signal be emitted.

Arguments:

- **signal**
/ Condition: required / *Type:* str /
The name of the DBus signal(s) which has been raised.
- **loop**
/ Condition: optional / *Type:* EventLoop / *Default:* None /
The Event loop which is running to wait for the raised signal.
- **payloads**
/ Condition: optional / *Type:* Any / *Default:* "" /
The payloads of the raised signal.

Returns:

(no returns)

10.1.7 Method: register_monitored_signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- **signal**
/ Condition: optional / *Type:* str / *Default:* "" /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

10.1.8 Method: wait_for_signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `wait_signal`
/ *Condition*: optional / *Type*: str / *Default*: '' /
The name of the DBus signal to wait for.
- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
/ *Type*: str /
The signal payloads.

10.1.9 Method: call_dbus_method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: '' /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- *Type*: Any /
Return from called method.

10.1.10 Method: call_dbus_method_with_keyword_args

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: '' /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- `ret_obj`
/ *Type*: Any /
Connection object.

CHAPTER 11. DBUS_CLIENT_REMOTE.PY

Chapter 11

dbus_client_remote.py

11.1 Class: DBusClientRemote

Imported by:

```
from RobotFramework_DBus.dbus_client_remote import DBusClientRemote
```

A client class for interacting with a specific DBus service on a remote machine.

11.1.1 Method: connect

Create a proxy object to DBus object.

Returns:

(*no returns*)

11.1.2 Method: disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(*no returns*)

11.1.3 Method: quit

Quit the DBus client.

Returns:

(*no returns*)

11.1.4 Method: do_signal_check

Checking if the signal was emitted.

Arguments:

- **signal**
/ *Condition*: required / *Type*: str /
The name of the DBus signal to check.
- **call_back_func**
/ *Condition*: required / *Type*: callable /
The function to be callback when receiving the signal.

Returns:

(*no returns*)

11.1.5 Method: set_signal_received_handler

Set a signal received handler for a specific signal.

Arguments:

- **signal**
/ Condition: required / *Type:* str /
The name of the DBus signal to handle.
- **handler**
/ Condition: required / *Type:* str /
The keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

Returns:

(no returns)

11.1.6 Method: unset_signal_received_handler

Unset a signal received handler for a specific signal.

Arguments:

- **signal**
/ Condition: required / *Type:* str /
The name of the DBus signal to handle.

Returns:

(no returns)

11.1.7 Method: register_monitored_signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- **signal**
/ Condition: optional / *Type:* str / *Default:* "" /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

11.1.8 Method: wait_for_signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- **wait_signal**
/ Condition: optional / *Type:* str / *Default:* "" /
The name of the DBus signal to wait for.
- **timeout**
/ Condition: optional / *Type:* int / *Default:* 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- payloads
 - / *Type*: str /
 - The signal payloads.

11.1.9 Method: call_dbus_method

Call a DBus method with the specified method name and input arguments.

Arguments:

- method_name
 - / *Condition*: optional / *Type*: str / *Default*: '' /
 - The name of the DBus method to be called.
- args
 - / *Condition*: optional / *Type*: tuple / *Default*: None /
 - Input arguments to be passed to the method.

Returns:

- ret_obj
 - / *Type*: Any /
 - Connection object.

CHAPTER 12. DBUS_MANAGER.PY

Chapter 12

dbus_manager.py

12.1 Class: DBusManager

Imported by:

```
from RobotFramework_DBus.dbus_manager import DBusManager
```

Class to manage all DBus connections.

12.1.1 Method: quit

Quit connection manager.

Returns:

(*no returns*)

12.1.2 Method: add_connection

Add a connection to managed dictionary.

Arguments:

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

Returns:

(*no returns*)

12.1.3 Method: remove_connection

Remove a connection by name.

Arguments:

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

Returns:

(*no returns*)

12.1.4 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.

12.1.5 Keyword: disconnect

Keyword for disconnecting a connection by name.

Arguments:

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

Returns:

(no returns)

12.1.6 Keyword: connect

Keyword used to establish a DBus connection.

Arguments:

- conn_name
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used.
- namespace
/ *Condition*: optional / *Type*: str / *Default*: '' /
The namespace of the DBus service. This identifies the specific service or group of services. It is used to differentiate between different service instances. The namespace should be a string that uniquely identifies the service.
- object_path
/ *Condition*: optional / *Type*: str / *Default*: None /
The object path of the DBus service. This identifies the specific object within the service that the action will be performed on. The object path should be a string that follows the DBus object path naming convention. It typically consists of a hierarchical structure separated by slashes (/).
- mode
/ *Condition*: optional / *Type*: str / *Default*: 'local' /
The mode of testing the DBus service. Possible values are 'local' or 'remote'. 'local' indicates testing on the current system, while 'remote' indicates testing on a remote system.

CHAPTER 12. DBUS_MANAGER.PY12.1. CLASS: DBUSMANAGER

• host

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

The IP address or hostname of the remote system where the DBus agent is running.

This parameter is applicable only if mode is set to 'remote'.

• port

/ Condition: optional / *Type:* int / *Default:* 2507 /

The port number on which the DBus agent is listening on the remote system.

This parameter is applicable only if mode is set to 'remote'.

Returns:

(no returns)

12.1.7 Keyword: set_signal_received_handler

Keyword used to set a signal received handler for a specific DBus connection and signal.

Arguments:

• conn_name

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

The name of the DBus connection.

• signal

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

The name of the DBus signal to handle.

• handler

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

The keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

Returns:

(no returns)

12.1.8 Keyword: unset_signal_received_handler

Keyword used to set a signal received handler for a specific DBus connection and signal.

Arguments:

• conn_name

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

The name of the DBus connection.

• signal

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

The name of the DBus signal to handle.

• handler

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

The robotframework keyword which is handling the signal emitted event.

Returns:

(no returns)

12.1.9 Keyword: register_signal

Keyword used to register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name of the DBus connection.
- `signal`
/ *Condition*: optional / *Type*: str / *Default*: '' /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

12.1.10 Keyword: call_dbus_method

Keyword used to call a DBus method with the specified method name and input arguments.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name of the DBus connection.
- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: '' /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- `ret_obj`
/ *Type*: Any /
Return from called method.

12.1.11 Keyword: wait_for_signal

Keyword used to wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name of the DBus connection.
- `signal`
/ *Condition*: optional / *Type*: str / *Default*: '' /
The name of the DBus signal to wait for.
- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

CHAPTER 12. DBUS_MANAGER.PY12.1. CLASS: DBUSMANAGER**Returns:**

- payloads
/ *Type*: str /
The signal payloads.

CHAPTER 13. APPENDIX

Chapter 13

Appendix

About this package:

Table 13.1: Package setup

Setup parameter	Value
Name	RobotFramework_DBus
Version	0.1.3
Date	19.10.2023
Description	Robot Framework QConnect library extension for dbus testing
Package URL	robotframework-dbus
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

Chapter 14

History

0.1.0	05/2023
<i>Initial version</i>	

8.9 RobotLog2RQM

RobotLog2RQM

v. 1.2.3

Tran Duy Ngoan

14.03.2024

Contents

1	Introduction	1
2	Description	2
2.1	Get Robot Framework XML result	2
2.2	Tool features	2
2.2.1	Usage	3
2.2.2	Basic import feature	3
2.2.3	Verify the given arguments	4
2.2.4	Import multiple *.xml result files	4
2.2.5	Create missing Test Case on RQM	4
2.2.6	Update existing Test Case on RQM	5
2.3	Robot Framework Test Case Information on RQM:	5
3	CRQM.py	7
3.1	Function: get_xml_tree	7
3.2	Class: CRQMCClient	7
3.2.1	Method: login	8
3.2.2	Method: verifyProjectName	8
3.2.3	Method: disconnect	8
3.2.4	Method: config	8
3.2.5	Method: userURL	9
3.2.6	Method: integrationURL	9
3.2.7	Method: webIDfromResponse	10
3.2.8	Method: webIDfromGeneratedID	10
3.2.9	Method: getResourceByID	10
3.2.10	Method: getAllByResource	11
3.2.11	Method: getAllBuildRecords	11
3.2.12	Method: getAllConfigurations	11
3.2.13	Method: getAllTeamAreas	11
3.2.14	Method: addTeamAreaNode	12
3.2.15	Method: createTestcaseTemplate	12
3.2.16	Method: createTCERTemplate	13
3.2.17	Method: createExecutionResultTemplate	14
3.2.18	Method: createBuildRecordTemplate	15
3.2.19	Method: createConfigurationTemplate	15
3.2.20	Method: createTSERTemplate	16
3.2.21	Method: createTestsuiteResultTemplate	16

CONTENTSCONTENTS

3.2.22 Method: createResource	17
3.2.23 Method: createBuildRecord	18
3.2.24 Method: createConfiguration	18
3.2.25 Method: updateResourceByID	19
3.2.26 Method: linkListTestcase2Testplan	19
3.2.27 Method: linkListTestcase2Testsuite	20
4 robotlog2rqm.py	21
4.1 Function: get_from_tags	21
4.2 Function: convert_to_datetime	21
4.3 Function: process_suite_metadata	22
4.4 Function: process_metadata	22
4.5 Function: process_suite	22
4.6 Function: process_test	23
4.7 Function: RobotLog2RQM	23
4.8 Class: Logger	23
4.8.1 Method: config	24
4.8.2 Method: log	24
4.8.3 Method: log_warning	24
4.8.4 Method: log_error	25
5 Appendix	26
6 History	27

CHAPTER 1. INTRODUCTION

Chapter 1

Introduction

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

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

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

So that **RobotLog2RQM** tool can:

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

Chapter 2

Description

2.1 Get Robot Framework XML result

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

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

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

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

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

*** Test Cases ***
Testcase 01
[Documentation]  This test is traceable with provided tcid
[Tags]  TCID-1001  FID-112  FID-111  ↵
        ↵ robotfile-https://github.com/test-fullautomation
Log      This is Testcase 01

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

Listing 2.1: Sample Robot Framework test case

Hint

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

2.2 Tool features

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

Its usage and features are described as following sections.

CHAPTER 2. DESCRIPTION2.2. TOOL FEATURES**2.2.1 Usage**

Use below command to get tools's usage:

```
RobotLog2DB -h
```

The tool's usage should be showed as below:

```
usage: RobotLog2RQM (RobotXMLResult to RQM importer) [-h] [-v] [--recursive]
           [--createmissing] [--updatetestcase] [--dryrun]
           resultxmlfile host project user password testplan

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

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

optional arguments:
-h, --help          show this help message and exit
-v, --version       Version of the RobotLog2RQM importer.
--recursive        if set, then the path is searched recursively for
                   log files to be imported.
--createmissing    if set, then all testcases without tcid are created
                   when importing.
--updatetestcase   if set, then testcase information on RQM will be updated
                   bases on robot testfile.
--dryrun           if set, then verify all input arguments
                   (includes RQM authentication) and show what would be done.
```

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

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

2.2.2 Basic import feature

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

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

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

- Login the RQM server with the provided credential, tehn verify the existences of given `project` , `testplan` on RQM
- Create RQM **Build Record** and **Test Environment** (if already provided in Robot Framework test case and not existing on RQM)
- Create new RQM **Test Case Execution Record - TCER** (if it is not existing) bases on test case ID (defined `TCID-xxx` in `[Tags]` of Robot Framework Test Cases) and `testplan` ID
- Create new RQM **Test Case Execution Result** which contents the detail and result state of Robot Framework test case
- Link all test case(s) to provided `testplan`

2.2.3 Verify the given arguments

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

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

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

2.2.4 Import multiple `*.xml` result files

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

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

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

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

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

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

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

2.2.5 Create missing Test Case on RQM

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

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

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

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

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

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

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

2.2.6 Update existing Test Case on RQM

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

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

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

2.3 Robot Framework Test Case Information on RQM:

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

<i>CHAPTER 2. DESCRIPTION</i>	<i>2.3. ROBOT FRAMEWORK TEST CASE INFORMATION ON RQM:</i>
-------------------------------	---

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

Table 2.1: RQM data & Robot Framework

CHAPTER 3. CRQM.PY

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.
- `bdt_d_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.

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT

- `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 resource and ID. The provided ID can be internalID (contains only digits) or externalID.

Arguments:

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

Returns:

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

3.2.7 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:

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

Returns:

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

3.2.9 Method: getResourceByID

Return data of provided resource and ID by GET method

Arguments:

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

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT

- **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:

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

Returns:

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

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

3.2.11 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 dConfiguration 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:

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT

```
{
    '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.

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT

- **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.

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT**Returns:**

- `sTCERxml`
/ Condition: required / 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).

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT

- 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.

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT

- 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': ''
}
```

CHAPTER 3. CRQM.PY3.2. CLASS: CRQMCLIENT**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

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:

- RQMCClient
/ 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 tcid are created when importing.
 - updatetestcase : if True, then testcases information on RQM will be updated bases on robot testfile.
 - dryrun : if True, then verify all input arguments (includes RQM authentication) and show what would be done.

Returns:

(*no returns*)

4.8 Class: Logger

Imported by:

CHAPTER 4. ROBOTLOG2RQM.PY4.8. CLASS: LOGGER

```
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.
- `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

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	RobotLog2RQM
Version	1.2.3
Date	14.03.2024
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

Chapter 6

History

0.1.0	07/2022
<i>Initial version</i>	
1.1.1	01.08.2022
<i>Rework repository's document bases on GenPackageDoc</i>	
1.1.2	25.08.2022
<ul style="list-style-type: none">- Correct indent of sourcode's docstring.- Update new style for history.	
1.1.3	13.10.2022
<ul style="list-style-type: none">- Fix findings and enhance README and document files- Change argument name 'outputfile' to 'resultxmlfile'	
1.1.4	10.11.2022
<i>Rename package to RobotLog2RQM</i>	
1.2.0	09.01.2023
<ul style="list-style-type: none">- Rework optional arguments and improve logging messages- Update README and document for publishing pypi	
1.2.1	14.06.2023
<i>Update README: fix links issue and update installation section</i>	
1.2.2	06.03.2024
<i>Fix findings in documentation</i>	
1.2.3	14.03.2024
<i>Add support for basic authentication as an alternative to SSO system</i>	

RobotLog2RQM.pdf

*Created at 10.04.2024 - 12:33:55
by GenPackageDoc v. 0.41.1*

8.10 RobotLog2DB

RobotLog2DB

v. 1.4.1

Tran Duy Ngoan

15.03.2024

Contents

1	Introduction	1
2	Description	2
2.1	Get Robot Framework XML result	2
2.1.1	Robot Framework test case Settings	2
2.1.2	Sample Robot Framework Test Case	3
2.1.3	Execute Robot Framework test case(s) to get result file	4
2.2	Tool features	5
2.2.1	Usage	5
2.2.2	Verify provided arguments	6
2.2.3	Searching *.xml result file(s)	6
2.2.4	Handle essential information for TestResultWebApp	6
2.2.5	Append to existing execution result	8
2.3	Display on WebApp	10
3	CDataBase.py	11
3.1	Class: CDataBase	11
3.1.1	Method: connect	11
3.1.2	Method: disconnect	12
3.1.3	Method: cleanAllTables	12
3.1.4	Method: sCreateNewTestResult	12
3.1.5	Method: nCreateNewFile	13
3.1.6	Method: vCreateNewHeader	14
3.1.7	Method: nCreateNewSingleTestCase	16
3.1.8	Method: nCreateNewTestCase	17
3.1.9	Method: vCreateTags	18
3.1.10	Method: vSetCategory	18
3.1.11	Method: vUpdateStartTime	19
3.1.12	Method: arGetCategories	19
3.1.13	Method: vCreateAbortReason	19
3.1.14	Method: vCreateReanimation	20
3.1.15	Method: vCreateCCRdata	20
3.1.16	Method: vFinishTestResult	20
3.1.17	Method: vUpdateEvtbls	20
3.1.18	Method: vUpdateEvtbl	21
3.1.19	Method: vEnableForeignKeyCheck	21
3.1.20	Method: sGetLatestFileID	21

<u>CONTENTS</u>	<u>CONTENTS</u>
3.1.21 Method: vUpdateFileEndTime	21
3.1.22 Method: vUpdateResultEndTime	22
3.1.23 Method: bExistingResultID	22
3.1.24 Method: arGetProjectVersionSWByID	22
4 robotlog2db.py	23
4.1 Function: collect_xml_result_files	23
4.2 Function: validate_xml_result	23
4.3 Function: is_valid_uuid	24
4.4 Function: is_valid_config	24
4.5 Function: get_from_tags	25
4.6 Function: get_branch_from_swversion	25
4.7 Function: format_time	25
4.8 Function: retrieve_result_starttime	26
4.9 Function: retrieve_result_endtime	26
4.10 Function: process_suite_metadata	26
4.11 Function: process_metadata	27
4.12 Function: process_suite	27
4.13 Function: process_test	28
4.14 Function: process_config_file	28
4.15 Function: normalize_path	29
4.16 Function: RobotLog2DB	29
4.17 Class: Logger	30
4.17.1 Method: config	30
4.17.2 Method: log	30
4.17.3 Method: log_warning	31
4.17.4 Method: log_error	31
5 Appendix	32
6 History	33

Chapter 1

Introduction

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

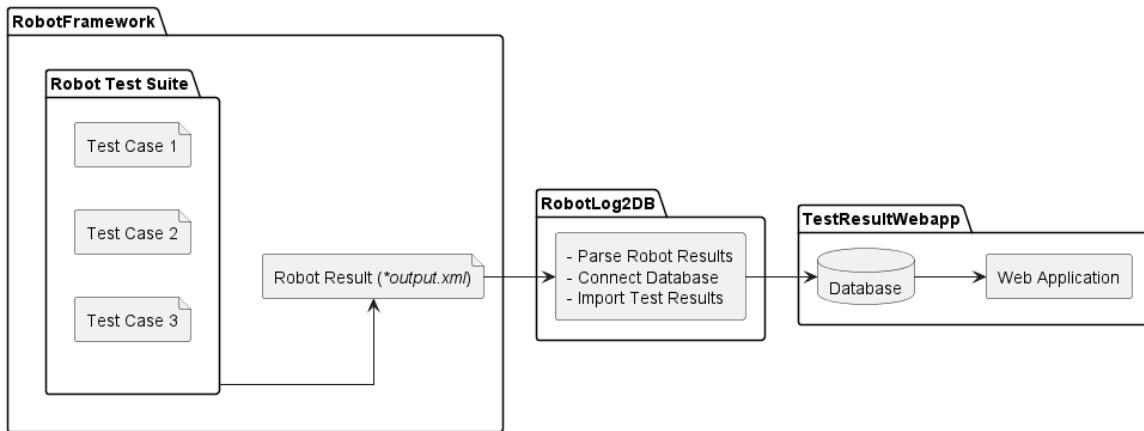


Figure 1.1: Tool data flow

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

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

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

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

CHAPTER 2. DESCRIPTION

Chapter 2

Description

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

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

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

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

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

2.1 Get Robot Framework XML result

2.1.1 Robot Framework test case Settings

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

For the whole test execution:

- Project/Variant

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

- Versions

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

For the Suite/File information:

- Description/Documentation

```
Documentation  ${Suite_description}
```

- Author

```
Metadata    author      ${Author_name}
```

- Component

CHAPTER 2. DESCRIPTION2.1. GET ROBOT FRAMEWORK XML RESULT

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

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

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

- Test Machine

```
Metadata machine ${COMPUTERNAME}
```

- Tester

```
Metadata tester ${USER}
```

For test case information:

- Issue ID

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

- Testcase ID

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

- Requirement ID

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

2.1.2 Sample Robot Framework Test Case

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

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

# File/Suite level
Documentation This is description for robot test file
Metadata author Tran Duy Ngoan (RBVH/ECM1)
Metadata component Import_Tools
Metadata testtool Robot Framework 4.1.3 (Python 3.9.16 on win32)
Metadata machine ${COMPUTERNAME}
Metadata tester ${USER}

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

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

Listing 2.1: Sample Robot Framework testcase

Hint

Instead of setting `Metadata` in Robot Framework test case, you can also specify them as the `--metadata` command line argument when executing Robot Framework test cases. Furthermore, in case you are using RobotFramework AIO, above highlighted `Metadata` definitions are not required because they have been handled by `RobotFramework.TestsuitesManagement` library within `Suite Setup`.

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

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

```
robot your_testcases.robot
```

Or with python module

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

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

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

2.2 Tool features

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

Its usage and enhance features are described as below.

2.2.1 Usage

Use below command to get tools's usage:

```
RobotLog2DB -h
```

The tool's usage should be showed as below:

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

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

positional arguments:
resultxmlfile      absolute or relative path to the result file or directory
                   of result files to be imported.
server            server which hosts the database (IP or URL).
user              user for database login.
password          password for database login.
database          database schema for database login.

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

As above instruction, **RobotLog2DB** tool requires 5 positional arguments which contains all required information for importing.

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

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

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

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

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

2.2.2 Verify provided arguments

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

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

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

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

2.2.3 Searching `*.xml` result file(s)

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

When the folder is used, **RobotLog2DB** will only search for `*.xml` file(s) under given directory and exclude any file within subdirectories as default.

In case you have result file(s) under the subdirectory of given folder and want these result files will also be imported, the optional argument `--recursive` should be used when executing **RobotLog2DB** command.

When `--recursive` argument is set, **RobotLog2DB** will walk through the given directory and its subdirectories to discover and collect all available `*.xml` for importing.

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

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

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

2.2.4 Handle essential information for TestResultWebApp

Default values

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

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

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

Specify essential information with optional arguments

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



Warning

These below settings may overwrite the existing information of `project`, `version_sw` and `component` which have been defined in Robot Framework test case in [above section](#).

So, **only** use these arguments in case you want to define the missing information or overwrite the existing values with your expected values for the importing result.

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



Notice

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

- `"testtool"` to specify the **Test toolchain** as `string` value.
- `"tester"` to specify the **Test user** as `string` value.
- `"components"` to specify the **Component** information which will be displayed on [TestResultWebApp](#). Value can be:

* `string` : to specify the same **Component** for all test cases within this execution.

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

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

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

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

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

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

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

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

2.2.5 Append to existing execution result

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

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

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

The command will be used as below:

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

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

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

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

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

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

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

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

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

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

Notice

When using append mode and `project / variant`, `version_sw` are provided within `--variant VARIANT`, `--versions VERSIONS` or `--config CONFIG` arguments, they will be validated with the existing values in database.

An error will be raised in case the given value is not matched with the existing ones. E.g:

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



CHAPTER 2. DESCRIPTION2.3. DISPLAY ON WEBAPP

2.3 Display on WebApp

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

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

Execution result metadata:

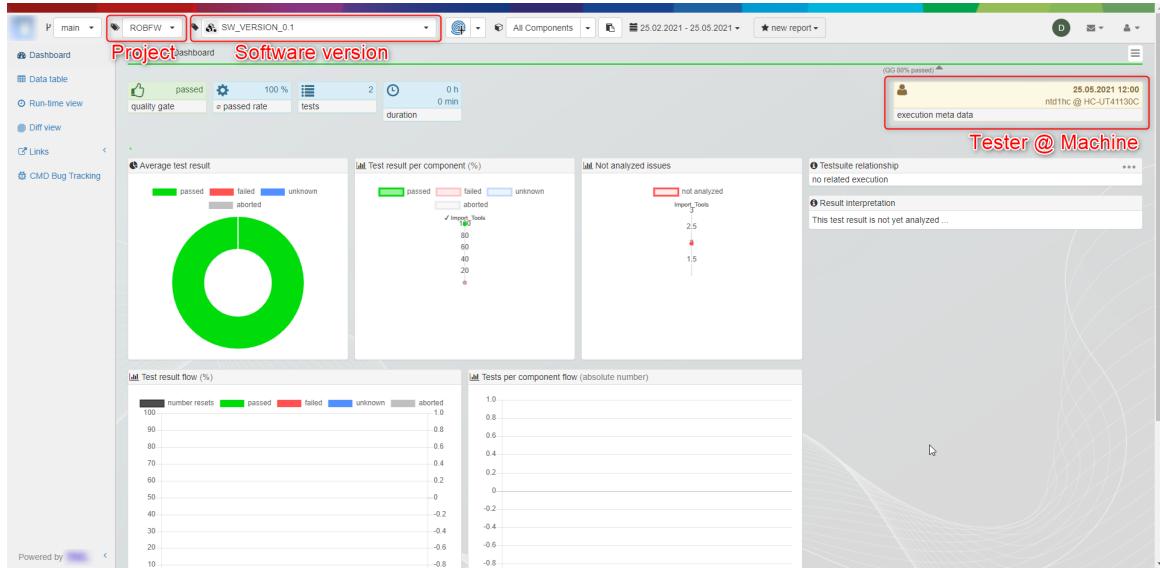


Figure 2.1: Dashboard view

Suite/File metadata and test case information:



Figure 2.2: Datatable view

CHAPTER 3. CDATABASE.PY

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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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. Default 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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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 repetition 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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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_EXECUTE MANY` 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 repetition 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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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: vUpdateStartTime

Create start-end time entry.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
UUID of test result.
- `_tbl_result_time_start`
/ Condition: required / Type: str /
Result start time as format %Y-%m-%d %H:%M:%S.
- `_tbl_result_time_end`
/ Condition: required / Type: str /
Result end time as format %Y-%m-%d %H:%M:%S.

Returns:

(no returns)

3.1.12 Method: arGetCategories

Get existing categories.

Arguments:

(no arguments)

Returns:

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

3.1.13 Method: vCreateAbortReason

Create abort reason entry.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
UUID of test result.
- `_tbl_abort_reason`
/ Condition: required / Type: str /
Abort reason.
- `_tbl_abort_message`
/ Condition: required / Type: str /
Detail message of abort.

Returns:

(no returns)

3.1.14 Method: vCreateReanimation

Create reanimation entry.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
UUID of test result.
- `_tbl_num_of_reanimation`
/ Condition: required / Type: int /
Counter of target reanimation during execution.

Returns:

(*no returns*)

3.1.15 Method: vCreateCCRdata

Create CCR data per test case.

Arguments:

- `_tbl_test_case_id`
/ Condition: required / Type: int /
test case ID.
- `lCCRdata`
/ Condition: required / Type: list /
list of CCR data.

Returns:

(*no returns*)

3.1.16 Method: vFinishTestResult

Finish upload:

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

Arguments:

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

Returns:

(*no returns*)

3.1.17 Method: vUpdateEvtbls

Call update_evtbls stored procedure.

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.1.18 Method: vUpdateEvtbl

Call update_evtbl stored procedure to update provided test_result_id.

Arguments:

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

Returns:

(no returns)

3.1.19 Method: vEnableForeignKeyCheck

Switch foreign_key_checks flag.

Arguments:

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

Returns:

(no returns)

3.1.20 Method: sGetLatestFileID

Get latest file ID from `tbl_file` table.

Arguments:

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

Returns:

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

3.1.21 Method: vUpdateFileEndTime

Update test file end time.

Arguments:

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

Returns:

(no returns)

3.1.22 Method: vUpdateResultEndTime

Update test result end time.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
Result UUID to be updated.
- `_tbl_result_time_end`
/ Condition: required / Type: str /
Result end time as format `%Y-%m-%d %H:%M:%S`.

Returns:

(no returns)

3.1.23 Method: bExistingResultID

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

Arguments:

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

Returns:

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

3.1.24 Method: arGetProjectVersionSWByID

Get the project and version_sw information of given `test_result_id`

Arguments:

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

Returns:

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

CHAPTER 4. ROBOTLOG2DB.PY

Chapter 4

robotlog2db.py

4.1 Function: collect_xml_result_files

Collect all valid Robot xml result file in given path.

Arguments:

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

Returns:

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

4.2 Function: validate_xml_result

Verify the given xml result file is valid or not.

Arguments:

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

Returns:

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

4.3 Function: is_valid_uuid

Verify the given UUID is valid or not.

Arguments:

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

Returns:

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

4.4 Function: is_valid_config

Validate the json configuration base on given schema.

Default schema supports below information:

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

Arguments:

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

Returns:

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

4.5 Function: get_from_tags

Extract testcase information from tags.

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

Arguments:

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

Returns:

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

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/feature branch. The leading number is the current year. E.g. 17.0F03
- All software version with .1S, .2S, ... is a stabl 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: 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:

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

4.8 Function: retrieve_result_starttime

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

Arguments:

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

Returns:

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

4.9 Function: retrieve_result_endtime

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

Arguments:

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

Returns:

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

4.10 Function: process_suite_metadata

Try to find metadata information from all suite levels.

Metadata at top suite level has a highest priority.

Arguments:

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

Returns:

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

4.11 Function: process_metadata

Extract metadata from suite result bases on DEFAULT_METADATA.

Arguments:

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

Returns:

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

4.12 Function: process_suite

Process to the lowest suite level (test file):

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

Arguments:

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

Returns:

(no returns)

4.13 Function: process_test

Process test case data and create new test case record.

Arguments:

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

Returns:

(no returns)

4.14 Function: process_config_file

Parse information from configuration file:

- component:

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

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

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

Arguments:

CHAPTER 4. ROBOTLOG2DB.PY4.15. FUNCTION: NORMALIZE_PATH

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

Returns:

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

4.15 Function: normalize_path

Normalize path file.

Arguments:

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

4.16 Function: RobotLog2DB

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

Flow to import Robot results to database:

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

Arguments:

- `args`
/ Condition: required / Type: ArgumentParser object /
 Argument parser object which contains:
 - `resultxmlfile` : path to the xml result file or directory of result files to be imported.
 - `server` : server which hosts the database (IP or URL).
 - `user` : user for database login.
 - `password` : password for database login.
 - `database` : database name.
 - `recursive` : if True, then the path is searched recursively for log files to be imported.
 - `dryrun` : if True, then verify all input arguments (includes DB connection) and show what would be done.
 - `append` : if True, then allow to append new result(s) to existing execution result UUID which is provided by `--UUID` argument.
 - `UUID` : UUID used to identify the import and version ID on TestResultWebApp.

CHAPTER 4. ROBOTLOG2DB.PY4.17. CLASS: LOGGER

- 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)*

4.17 Class: Logger

Imported by:

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

Logger class for logging message.

4.17.1 Method: config

Configure Logger class.

Arguments:

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

Returns:*(no returns)*

4.17.2 Method: log

Write log message to console/file output.

Arguments:

- msg
/ Condition: optional / Type: str / Default: "" /
Message which is written to output.
- color
/ Condition: optional / Type: str / Default: None /
Color style for the message.
- indent
/ Condition: optional / Type: int / Default: 0 /
Offset indent.

Returns:*(no returns)*

4.17.3 Method: log_warning

Write warning message to console/file output.

Arguments:

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

Returns:

(no returns)

4.17.4 Method: log_error

Write error message to console/file output.

Arguments:

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

Returns:

(no returns)

CHAPTER 5. APPENDIX

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	RobotLog2DB
Version	1.4.1
Date	15.03.2024
Description	Imports robot result(s) to TestResultWebApp database
Package URL	robotframework-robotlog2db
Author	Tran Duy Ngoan
Email	Ngoan.TranDuy@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

CHAPTER 6. HISTORY

Chapter 6

History

0.1.0	07/2022
<i>Initial version</i>	
1.2.1	22.08.2022
<i>Rework repository's document bases on GenPackageDoc</i>	
1.2.2	13.10.2022
<ul style="list-style-type: none"> - Fix findings and enhance README and document files - Change argument name 'outputfile' to 'resultxmlfile' 	
1.2.3	10.11.2022
<i>Rename package to RobotLog2DB</i>	
1.2.4	18.11.2022
<i>Add -append argument which allow to append into existing UUID</i>	
1.3.0	06.12.2022
<ul style="list-style-type: none"> - Rework optional arguments in command line - Improve coding: logging messages, shorten variable names, ... 	
1.3.1	09.01.2023
<i>Improve messages when verify configuration json file</i>	
1.3.2	28.02.2023
<ul style="list-style-type: none"> - Rename key to 'components' in configuration json file - Change implementation of append mode to raise errors without proper given UUID - Enhance console log with component and append mode information - Add more supported keys in configuration json file 	
1.3.3	01.03.2023
<i>Add a validation for xmlresultfile with robot schema</i>	
1.3.4	13.03.2023
<ul style="list-style-type: none"> - Add a validation for existing project/variant and version_sw in db when using append mode - Remove db maxlen handlers: truncations and validation 	
1.3.5	21.03.2023
<i>Support 4 byte characters for importing to db</i>	
1.3.6	05.04.2023
<ul style="list-style-type: none"> - Enhance console log with test case counter and component statistics - Add try/except for database access 	
1.3.7	14.06.2023
<i>Update README with new instruction for installation and image's links</i>	

CHAPTER 6. HISTORY

1.3.8	21.06.2023
<i>Fix issue with suite starttime at root suite is N/A</i>	
1.3.9	23.06.2023
<ul style="list-style-type: none">- Retrieve start/end time for suite- Update result schema to support Robotframework Version 6.1	
1.4.0	19.09.2023
<i>Adaption for Robotframework 6.1 with change of <code>TestSuite.source</code> datatype</i>	
1.4.1	15.03.2024
<i>update <code>robot.xsd</code> schema to support the new log level <code>USER</code> of Robotframework</i>	

RobotLog2DB.pdf

*Created at 10.04.2024 - 12:33:57
by GenPackageDoc v. 0.41.1*

8.11 RobotFramework_DoIP

RobotFramework_DoIP

v. 0.1.2

Hua Van Thong

08.04.2024

Contents

1	Introduction	1
1.1	Overview	1
1.2	Abbreviations	1
1.3	Terms and definitions	1
1.3.1	Diagnostic Power Mode	1
1.3.2	Transport protocol	2
1.3.3	Diagnostics tester	2
1.3.4	Diagnostics gateway	2
1.3.5	Logical addresses	2
2	Description	3
2.1	System Overview	3
2.2	DoIP application scenarios	4
2.2.1	Example of Diagnostic Process	4
2.2.2	Example of Vehicle Identification	6
3	The Ecu Simulator	7
3.1	Initialize	7
3.2	Start	8
3.3	Example	9
4	DoipKeywords.py	11
4.1	Class: DoipKeywords	11
4.1.1	Method: connect_to_ecu	11
4.1.2	Method: send_diagnostic_message	12
4.1.3	Method: receive_diagnostic_message	13
4.1.4	Method: reconnect_to_ecu	13
4.1.5	Method: disconnect	14
4.1.6	Method: await_vehicle_announcement	14
4.1.7	Method: get_entity	15
4.1.8	Method: request_entity_status	15
4.1.9	Method: request_vehicle_identification	16
4.1.10	Method: request_alive_check	16
4.1.11	Method: request_activation	17
4.1.12	Method: request_diagnostic_power_mode	17
5	RobotFramework_DoIP.py	18
5.1	Function: get_version	18

CONTENTSCONTENTS

5.2 Function: get_version_date	18
6 __init__.py	19
6.1 Class: RobotFramework_DoIP	19
7 Appendix	20
8 History	21

CHAPTER 1. INTRODUCTION

Chapter 1

Introduction

1.1 Overview

RobotFramework.DoIP is a Robot Framework library specifically designed for interacting with Electronic Control Units (ECUs) using the Diagnostics over Internet Protocol (DoIP).

At its core, DoIP serves as a communication bridge between external diagnostic tools and a vehicle's ECUs. This library, RobotFrameworkDoIP, provides a set of keywords that enable users to perform diagnostic operations and engage with ECUs, facilitating automated testing processes and interaction with vehicles through the DoIP protocol.

The **RobotFramework.DoIP** sources can be found in repository **robotframework-doip**: [DoIP](#)

1.2 Abbreviations

Table 1.1: Abbreviation

Abbreviation / Acronym	Description
ARP	Address Resolution Protocol
DHCP	Diagnostic Host Configuration Protocol
EID	Entity identifier
GID	Group identifier
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
TCP	Transmission Control Protocol
TCP/IP	A family of communication protocols used in computer networks
VIN	Vehicle Identification Number
UDP	User Datagram Protocol

1.3 Terms and definitions

1.3.1 Diagnostic Power Mode

The vehicle internal power supply status affects the diagnostic capabilities of all ECUs on the in-vehicle network and identifies the status of all ECUs in all gateway subnetworks that allow diagnostic communication. Its purpose is to provide external test equipment with whether it can be performed on the connected vehicle.

1.3.2 Transport protocol

Transport protocols reside on top of the IP protocol. Transport protocols run over the best-effort IP layer to provide a mechanism for applications to communicate with each other without directly interacting with the IP layer.

The important thing here is that DoIP does not represent a diagnostic protocol according to ISO 13400 but rather an expanded transport protocol. This means that the transmission of diagnostic packets is defined in DoIP, but the contained diagnostic services continue to be specified and described by diagnostic protocols such as KWP2000 and UDS.

1.3.3 Diagnostics tester

A diagnostic tester / diagnostic client enables the sending of diagnostic requests. Testers can take the form of external devices, such as in repair shops, or on-board testers in the vehicle. The receiving ECU must process the diagnostic requests and return an associated diagnostic response to the tester.

1.3.4 Diagnostics gateway

The gateway assumes the role of the intermediary. Requests of the tester are forwarded to internal networks so that a desired ECU can receive and process them. As soon as a response from the requested ECU is available, the gateway routes this back to the tester.

1.3.5 Logical addresses

A diagnostic gateway always requires two pieces of information in order to forward diagnostic requests and responses.

- First, it requires a logical address that uniquely identifies the ECU to be diagnosed in the vehicle, equivalent to the `ecu logical address` parameter.
- Second, the gateway must know which messages on the respective bus system or network will be used to send diagnostic requests and to receive diagnostic responses, equivalent to `client logical address` parameter.

Both pieces of information must be available for an ECU for it to be accessible via the gateway.

Chapter 2

Description

2.1 System Overview

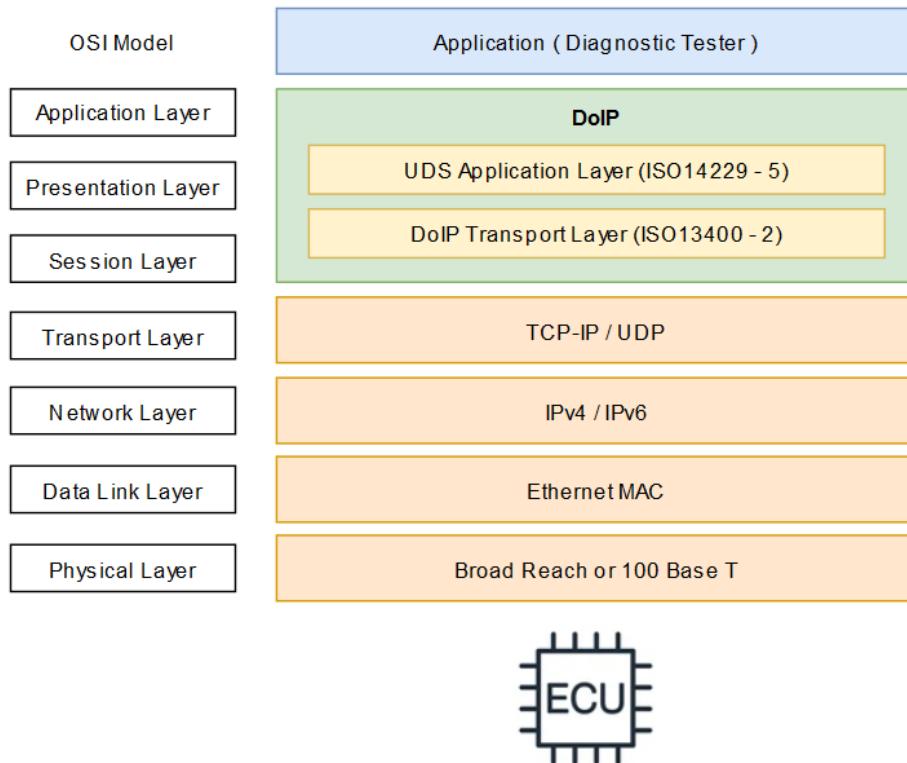


Figure 2.1: System overview

- The DoIP (Diagnostics over Internet Protocol) protocol is a standard for vehicle diagnostics that allows communication between diagnostic tester devices and electronic control units (ECU) over Ethernet networks.
- DoIP is a standardized diagnostic transport protocol according to ISO 13400.
 - DoIP Transport Layer (ISO 13400-2) is equipped with features to establish and maintain connection between external tester device and DoIP gateway inside the vehicle.
 - UDS application layer (ISO 14229-5) is the application profile that implements UDS on IP.
- The overall goal of the protocol is to encapsulate diagnostics messages of protocol standards like Unified Diagnostic Services (UDS) and route them to and from the ECU.

CHAPTER 2. DESCRIPTION2.2. DOIP APPLICATION SCENARIOS

- The DoIP gateway or server can be a part of the ECU. A vehicle can have multiple DoIP entities and multiple testing devices and ECUs can route their traffic via a single DoIP entity.
- DoIP uses both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) for specific phases of the underlying layer. The initial announcement and identification messages are over UDP, after which the communication switches over to TCP.

2.2 DoIP application scenarios

This section will provide some features along with examples through system above:

- Vehicle identification and announcement: Is necessary to detect who is participating in the DoIP communication.
- Request diagnostic message: Request for diagnostic information, which is crucial for diagnosing vehicle issues and ensuring effective communication within the DoIP network.
- Routing Activation: Allows that single Diagnostic Message pathes are activated or not to treat different protocols different (like UDS and OBD) and to also treat single testers different.
- Entity DoIP (Gateway): Provides general information of the single DoIP entity. Usually used by the testers to get the current DoIP protocol relevant information from the single DoIP Entities.
- Alive mechanism: Is used to maintain different tester connections.

2.2.1 Example of Diagnostic Process

Vehicle Diagnostics

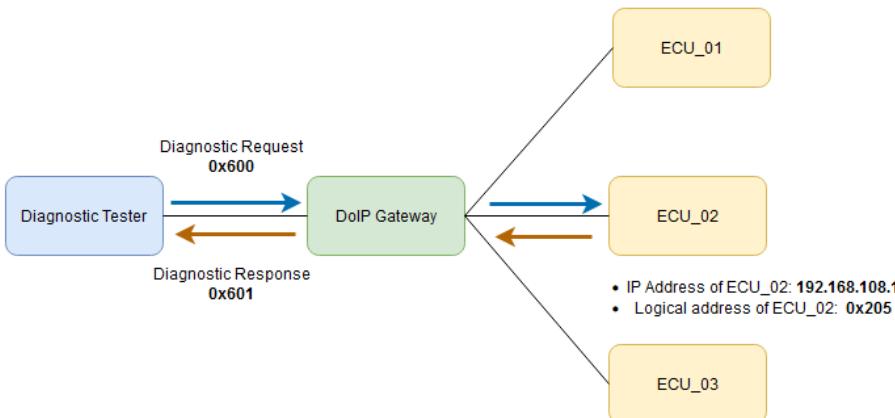


Figure 2.2: Vehicle diagnostic process demonstration

- **Use Case:**

A vehicle diagnostic tool needs to establish a DoIP connection to communicate with an Electronic Control Unit (ECU) within a vehicle for diagnostic purposes.

- **Scenario:**

The diagnostic tool initiates a connection to the ECU's IP address (192.168.108.1) and logical address 517 (0x205) using the **RobotFramework.DoIP** library.

It sends a diagnostic message (600) to the ECU, receives a response, logs the response to the console, and then disconnects from the ECU.

```
*** Settings ***
Library   RobotFramework_DoIP
```

CHAPTER 2. DESCRIPTION2.2. DOIP APPLICATION SCENARIOS

```
*** Test Cases ***
Test Establish an DoIP connection
    # Establish an connection to ecu ip address and ecu logical address
    Connect To ECU      192.168.108.1      517
    Send Diagnostic Message      600
    ${res}= Receive Diagnostic Message
    Log To Console      ${resp}
    Disconnect
```

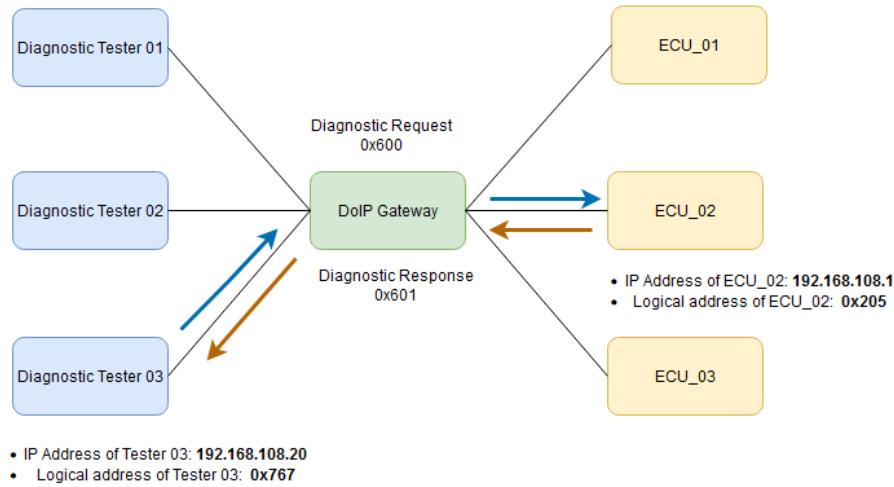
Remote Testing

Figure 2.3: Remote diagnostic process demonstration

• Use Case:

A remote testing environment requires a connection between a target tester and an ECU located in a different location.

• Scenario:

The target tester, located at IP address 192.168.108.20, establishes a DoIP connection to the ECU at IP address 192.168.108.1 and logical address 517 (0x205). Additionally, the target tester specifies its own logical address 1895 (0x767) using the **RobotFramework_DoIP** library.

It sends a diagnostic message (600) to the ECU, receives a response, logs the response to the console, and then disconnects from the ECU.

```
*** Settings ***
Library     RobotFramework_DoIP

*** Test Cases ***
Test Establish an DoIP connection between specific tester and target ECU
    # Establish an connection to ecu ip address and ecu logical address
    Connect To ECU      192.168.108.1      517      client_ip_address=192.168.108.20      ↵
    ↵ client_logical_address=1895
    Send Diagnostic Message      600
    ${res}= Receive Diagnostic Message
    Log To Console      ${resp}
    Disconnect
```

2.2.2 Example of Vehicle Identification

- **Use Case:**

A diagnostic tool needs to request vehicle identification information from an Electronic Control Unit (ECU) within a vehicle.

- **Scenario:**

The diagnostic tool initiates a DoIP connection to the ECU's IP address (192.168.108.1) and logical address (205) using the **RobotFramework_DoIP** library. It then sends a request for vehicle identification information to the ECU.

```
*** Settings ***
Library    RobotFramework_DoIP

*** Test Cases ***
Test Request Vehicle Identification
    Connect To ECU      192.168.108.1      205
    Request Vehicle Identification
```

Chapter 3

The Ecu Simulator

This chapter provides a detailed explanation of the utilization of the ECU simulator through DoIP base on doipclient library. It serves for development or testing scenarios where a physical device is not available.

The ECU simulator is designed to receive messages and respond accordingly to the following types of messages:

- Alive Check Request
- Diagnostic Power Mode Request
- Doip Entity Status Request
- Routing Activation Request
- Vehicle Identification Request

3.1 Initialize

This function sets up an instance of an ECU, initializes its attributes with default values, and includes placeholders for various properties that can be customized based on specific requirements.

```
def __init__(self, ecu_type, ip_address, tcp_port, udp_port):
    # Initialize ECU attributes with default values
    self.ecu_type = ecu_type
    self.ip_address = ip_address
    self.tcp_port = tcp_port
    self.udp_port = udp_port
    self.tcp_socket = None
    self.udp_socket = None
    # Set default values for various ECU properties
    # These values might be placeholders and can be updated based on your actual ↵
    ↵ requirements
    self._ecu_logical_address = 3584
    self._client_logical_address = 3584
    self._logical_address = 55
    self._response_code = doip_message.RoutingActivationResponse.ResponseCode.Success
    self._diagnostic_power_mode = ↵
    ↵ doip_message.DiagnosticPowerModeResponse.DiagnosticPowerMode.Ready
    self._node_type = 1
    self._max_concurrent_sockets = 16
    self._currently_open_sockets = 1
    self._max_data_size = None
    self._vin = '19676527011956855057'
    self._eid = b'11111'
    self._gid = b'222222'
    self._further_action_required = ↵
    ↵ doip_message.VehicleIdentificationResponse.FurtherActionCodes.NoFurtherActionRequired
    self._vin_sync_status = ↵
    ↵ doip_message.VehicleIdentificationResponse.SynchronizationStatusCodes.Synchronized
```

3.2 Start

This method is responsible for initializing and setting up TCP and UDP sockets, binding them to specific IP addresses and ports, and then starting separate threads to handle the communication on these sockets concurrently.

```
def start(self):
    # Create TCP socket
    self.tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.tcp_socket.bind((self.ip_address, self.tcp_port))
    self.tcp_socket.listen(5)

    # Create UDP socket
    self.udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.udp_socket.bind((self.ip_address, self.udp_port))

    # Start listening on separate threads
    tcp_thread = threading.Thread(target=self.listen_tcp)
    udp_thread = threading.Thread(target=self.listen_udp)

    tcp_thread.start()
    udp_thread.start()
```

Explanation:

1. TCP Socket Setup

- A TCP socket is created using the `socket` module with the `socket.AF_INET` family (IPv4) and `socket.SOCK_STREAM` type (TCP).
- The TCP socket is bound to the specified IP address `self.ip_address` and TCP port `self.tcp_port`.
- The TCP socket is set to listen for incoming connections with a backlog of 5 connections.

2. UDP Socket Setup

- A UDP socket is created using the same `socket` module with the `socket.AF_INET` family (IPv4) and `socket.SOCK_DGRAM` type (UDP).
- The UDP socket is bound to the specified IP address `self.ip_address` and UDP port `self.udp_port`.

3. Thread Creation

- Two separate threads `tcp_thread` and `udp_thread` are created using the `threading` module.
- The target parameter of each thread is set to point to specific methods `self.listen_tcp` and `self.listen_udp`, suggesting that these methods likely contain the logic for handling TCP and UDP communication.

4. Thread Start

- Both threads are started concurrently using the `start` method, allowing the ECU to handle TCP and UDP communication simultaneously.

3.3 Example

We have provided an example demonstrating the usage of the ECU simulator in the file located at `test_ecu_simulator.py`

```
if __name__ == "__main__":
    # Create and start instances of different ECUs using the factory pattern and ↵
    ↵ abstract class
    factory = ECUFactory()

    positive_ecu = factory.create_ecu(ECUType.POSITIVE_ECU, POSITIVE_ECU_IP, ↵
    ↵ POSITIVE_TCP_PORT, POSITIVE_UDP_PORT)
    negative_ecu = factory.create_ecu(ECUType.NEGATIVE_ECU, NEGATIVE_ECU_IP, ↵
    ↵ NEGATIVE_TCP_PORT, NEGATIVE_UDP_PORT)
    # Start positive and negative ECUs
    positive_ecu.start()
    negative_ecu.start()
```

In the given example, an instance of the ECU is created in `ecu_simulator.py` by specifying the ECU's IP address, TCP port, and UDP port. Subsequently, the start method is invoked to initiate its operation.

Output:

```
TCP Server 172.17.0.5 listening on port 13400
UDP Server 172.17.0.5 listening on port 13400
TCP Server 172.17.0.5 listening on port 12346
UDP Server 172.17.0.5 listening on port 12347
```

Now you can execute the test by running the file located at `test_ecu_simulator.py`

```
def test_positive_ecu_simulator():
    try:
        ip = '172.17.0.5'
        ecu_logical_address = 57344

        # Create a DoIPClient instance for positive ECU simulator
        doip = DoIPClient(ip, ecu_logical_address, activation_type=None)

        # Test various interactions
        print(doip.request_diagnostic_power_mode())
        print(doip.request_entity_status())
        print(doip.request_alive_check())
        print(doip.request_activation(1))
        print(doip.get_entity())
        print(doip.request_vehicle_identification(vin="1" * 17))
        print(doip.request_vehicle_identification(eid=b"1" * 6))

    except Exception as e:
        print(f"Error during positive ECU simulation: {e}")
```

Output:

```
# Diagnostic power mode response
DiagnosticPowerModeResponse (0x4004): { diagnostic_power_mode : ↵
    ↪ DiagnosticPowerMode.Ready }

# Entity status response
EntityStatusResponse (0x4002): { node_type : 1, max_concurrent_sockets : 16, ↵
    ↪ currently_open_sockets : 1, max_data_size : None }

# Alive check response
AliveCheckResponse (0x8): { source_address : 3584 }

# Routing activation response
RoutingActivationResponse (0x6): { client_logical_address : 3584, logical_address : ↵
    ↪ 55, response_code : ResponseCode.Success, reserved : 0, vm_specific : None }

# Get entity response
(('172.17.0.5', 13400), VehicleIdentificationResponse(b'19676527011956855', 3584, ↵
    ↪ b'11111\x00', b'222222', 0, 0))

# Vehicle identification response
VehicleIdentificationResponse (0x4): { vin: "19676527011956855", logical_address : ↵
    ↪ 3584, eid : b'11111\x00', gid : b'222222', further_action_required : ↵
    ↪ FurtherActionCodes.NoFurtherActionRequired, vin_sync_status : ↵
    ↪ SynchronizationStatusCodes.Synchronized }
VehicleIdentificationResponse (0x4): { vin: "19676527011956855", logical_address : ↵
    ↪ 3584, eid : b'11111\x00', gid : b'222222', further_action_required : ↵
    ↪ FurtherActionCodes.NoFurtherActionRequired, vin_sync_status : ↵
    ↪ SynchronizationStatusCodes.Synchronized }
```

CHAPTER 4. DOIPKEYWORDS.PY

Chapter 4

DoipKeywords.py

4.1 Class: DoipKeywords

Imported by:

```
from RobotFramework_DoIP.DoipKeywords import DoipKeywords
```

4.1.1 Method: connect_to_ecu

Description:

Establishing a DoIP connection to an (ECU) within the context of automotive communication.

Parameters:

- **param ecu_ip_address (required):** The IP address of the ECU to establish a connection. This should be an address like "192.168.1.1" or an IPv6 address like "2001:db8::".
- type ecu_ip_address: str
- **param ecu_logical_address (required):** The logical address of the ECU.
- type ecu_logical_address: any
- **param tcp_port (optional):** The TCP port used for unsecured data communication (default is **TCP_DATA_UNSECURED**).
- type tcp_port: int
- **param udp_port (optional):** The UDP port used for ECU discovery (default is **UDP_DISCOVERY**).
- type udp_port: int
- **param activation_type (optional):** The type of activation, which can be the default value (**ActivationTypeDefault**) or a specific value based on application-specific settings.
- type activation_type: RoutingActivationRequest.ActivationType,
- **param protocol_version (optional):** The version of the protocol used for the connection (default is 0x02).
- type protocol_version: int
- **param client_logical_address (optional):** The logical address that this DoIP client will use to identify itself. This should be 0x0E00 to 0x0FFF. Can typically be left as default.
- type client_logical_address: int
- **param client_ip_address (optional):** If specified, attempts to bind to this IP as the source for both TCP and UDP traffic. Useful if you have multiple network adapters. Can be an IPv4 or IPv6 address just like ecu_ip_address, though the type should match.
- type client_ip_address: str
- **param use_secure (optional):** Enables TLS. If set to True, a default SSL context is used. For more information on SSL contexts, see the [Robot Framework documentation](#). An SSL context can be passed directly. Untested. Should be combined with changing tcp_port to 3496.

CHAPTER 4. DOIPKEYWORDS.PY4.1. CLASS: DOIPKEYWORDS

- type `use_secure`: Union[bool,ssl.SSLContext]
- param `auto_reconnect_tcp` (optional): Attempt to automatically reconnect TCP sockets that were closed by peer
- type `auto_reconnect_tcp`: bool

Return:

None

Exception:

raises ConnectionError: Failed to establish a DoIP connection

Usage:

```
# Explicitly specifies all establishing a connection
• Connect To ECU | 172.17.0.111 | 1863 |
• Connect To ECU | 172.17.0.111 | 1863 | client_ip_address=172.17.0.5 | client_logical_address=1895 |
• Connect To ECU | 172.17.0.111 | 1863 | client_ip_address=172.17.0.5 | client_logical_address=1895 | activation_type=0 |
```

4.1.2 Method: send_diagnostic_message**Description:**

Send a raw diagnostic payload (ie: UDS) to the ECU.

Parameters:

- param `diagnostic_payload`: UDS payload to transmit to the ECU
- type `diagnostic_payload`: string
- param `timeout`: send diagnostic time out (default: A_PROCESSING_TIME)
- type `timeout`: int (s)

Return:

None

Exception:

raises ConnectionRefusedError: DoIP connection attempt failed raises IOError: DoIP negative acknowledgement received

Usage:

```
# Explicitly specifies all diagnostic message properties
• Send Diagnostic Message | 1040 |
• Send Diagnostic Message | 1040 | timeout=10 |
```

CHAPTER 4. DOIPKEYWORDS.PY4.1. CLASS: DOIPKEYWORDS**4.1.3 Method: receive_diagnostic_message****Description:**

Receive a raw diagnostic payload (ie: UDS) from the ECU.

Parameters:

- param `timeout`: time waiting diagnostic message (default: None)
- type `timeout`: int (s)

Return:

None

Exception:

raises ConnectionRefusedError: DoIP connection attempt failed
raises IOError: DoIP negative acknowledgement received

Usage:

```
# Explicitly specifies all diagnostic message properties
• Receive Diagnostic Message |
• Receive Diagnostic Message | timeout=10 |
```

4.1.4 Method: reconnect_to_ecu**Description:**

Attempts to re-establish the connection. Useful after an ECU reset

Parameters:

- param `close_delay`: Time to wait between closing and re-opening socket (default: **A_PROCESSING_TIME**)
- type `close_delay`: int (s)

Return: None**Exception:**

raises ConnectionRefusedError: DoIP connection attempt failed

Usage:

```
# Explicitly specifies all diagnostic message properties
• Reconnect To Ecu |
• Reconnect To Ecu | close_delay=10 |
```

CHAPTER 4. DOIPKEYWORDS.PY4.1. CLASS: DOIPKEYWORDS**4.1.5 Method: disconnect****Description:**

Close the DoIP client

Parameters:

None

Return:

None

Exception:

raises ConnectionRefusedError: DoIP connection attempt failed raises ConnectionAbortedError: close DoIP connection aborted

Usage:

```
# Explicitly specifies all diagnostic message properties
• Disconnect
```

4.1.6 Method: await_vehicle_announcement**Description:**

When an ECU first turns on, it's supposed to broadcast a Vehicle Announcement Message over UDP 3 times to assist DoIP clients in determining ECU IP's and Logical Addresses. Will use an IPv4 socket by default, though this can be overridden with the ipv6 parameter.

Parameters:

- param udp_port: The UDP port to listen on. Per the spec this should be 13400, but some VM's use a custom
- one.
- type udp_port: int, optional
- param timeout: Maximum amount of time to wait for message
- type timeout: float, optional
- param ipv6: Bool forcing IPV6 socket instead of IPV4 socket
- type ipv6: bool, optional
- param source_interface: Interface name (like "eth0") to bind to for use with IPv6. Defaults to None. will use the default interface (which may not be the one connected to the ECU). Does nothing for IPv4, which will bind to all interfaces uses INADDR_ANY.
- type source_interface: str, optional

Return:

- return: IP Address of ECU and VehicleAnnouncementMessage object
- rtype: tuple

Exception:

raises TimeoutError: If vehicle announcement not received in time

Usage:

```
# Explicitly specifies all diagnostic message properties
• Await Vehicle Annoucement
• Await Vehicle Annoucement | timeout=10
```

CHAPTER 4. DOIPKEYWORDS.PY4.1. CLASS: DOIPKEYWORDS**4.1.7 Method: get_entity****Description:**

Sends a VehicleIdentificationRequest and awaits a VehicleIdentificationResponse from the ECU, either with a specified VIN, EIN, or nothing. Equivalent to the request_vehicle_identification() method but can be called without instantiation

Parameters:

- param udp_port: The UDP port to listen on. Per the spec this should be 13400, but some VM's use a custom
- one.
- type udp_port: int, optional
- param timeout: Maximum amount of time to wait for message
- type timeout: float, optional
- param ipv6: Bool forcing IPV6 socket instead of IPV4 socket
- type ipv6: bool, optional
- param source_interface: Interface name (like "eth0") to bind to for use with IPv6. Defaults to None will use the default interface (which may not be the one connected to the ECU). Does nothing for IPv4, which will bind to all interfaces uses INADDR_ANY.
- type source_interface: str, optional

Return:

- return: IP Address of ECU and VehicleAnnouncementMessage object
- rtype: tuple

Exception:

raises TimeoutError: If vehicle announcement not received in time

Usage:

- Get Entity |
- Get Entity | ecu_ip_address=172.17.0.111 |
- Get Entity | ecu_ip_address=172.17.0.111 | protocol_version=0x02

4.1.8 Method: request_entity_status**Description:**

Request that the ECU send a DoIP Entity Status Response

Parameters:

None

Return:

None

Exception:

None

Usage:

- Request Entity Status

4.1.9 Method: request_vehicle_identification

Description:

Sends a VehicleIdentificationRequest and awaits a VehicleIdentificationResponse from the ECU, either with a specified VIN, EIN, or nothing

Parameters:

```
param eid EID of the Vehicle
type eid bytes, optional
param vin VIN of the Vehicle
type vin str, optional
```

Return:

None

Exception:

None

Usage:

- Request Vehicle Identification
- Request Vehicle Identification | eid=0x123456789abc
- Request Vehicle Identification | vin=0x123456789abc

4.1.10 Method: request_alive_check

Description:

Request that the ECU send an alive check response

Parameters:

None

Return:

None

Exception:

None

Usage:

- Request Vehicle Identification
- Request Vehicle Identification | eid=0x123456789abc
- Request Vehicle Identification | vin=0x123456789abc

4.1.11 Method: request_activation**Description:**

Requests a given activation type from the ECU for this connection using payload type 0x0005

Parameters:

- param **activation_type** (required): The type of activation to request - see Table 47 ("Routing activation request activation types") of ISO-13400, but should generally be 0 (default) or 1 (regulatory diagnostics)
- type activation_type: RoutingActivationRequest.ActivationType
- param vm_specific (optional): 4 byte long int
- type vm_specific: int, optional
- param **disable_retry**: Disables retry regardless of auto_reconnect_tcp flag. This is used by activation requests during connect/reconnect.
- type disable_retry: bool, optional

Return:

None

Exception:

None

Usage:

- Request Routing Activation | \${0x02}
- Request Routing Activation | vm_specific=
- Request Routing Activation | vin=0x123456789abc

4.1.12 Method: request_diagnostic_power_mode**Description:**

Request that the ECU send a Diagnostic Power Mode response

Parameters:

None

Return:

None

Exception:

None

Usage:

- Request Diagnostic Power Mode

CHAPTER 5. ROBOTFRAMEWORK_DOIP.PY

Chapter 5

RobotFramework_DoIP.py

5.1 Function: `get_version`

5.2 Function: `get_version_date`

CHAPTER 6. __INIT__.PY

Chapter 6

`__init__.py`

6.1 Class: RobotFramework_DoIP

Imported by:

```
from RobotFramework_DoIP.__init__ import RobotFramework_DoIP
```

RobotFrameworkDoIP is a Robot Framework library aimed to provide DoIP protocol for diagnostic message.

CHAPTER 7. APPENDIX

Chapter 7

Appendix

About this package:

Table 7.1: Package setup

Setup parameter	Value
Name	RobotFramework_DoIP
Version	0.1.2
Date	08.04.2024
Description	RobotFramework for DoIP Client
Package URL	robotframework-doip
Author	Hua Van Thong
Email	thong.huavan@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 8. HISTORY

Chapter 8

History

0.1.0	09/2023
<i>Initial version</i>	
0.1.1	12/2023
<i>Add ecu simulator to use for self test</i>	
0.1.2	4/2024
<i>Update the documentation for DoIP</i>	

RobotFramework_DoIP.pdf
Created at 10.04.2024 - 12:34:00
by GenPackageDoc v. 0.41.1

8.12 PyTestLog2DB

PyTestLog2DB

v. 0.2.8

Tran Duy Ngoan

14.12.2023

Contents

1	Introduction	1
2	Description	2
2.1	Execute pytest test case(s) to get result file	2
2.2	Tool features	2
2.2.1	Usage	2
2.2.2	Verify provided arguments	3
2.2.3	Searching *.xml result file(s)	3
2.2.4	Handle essential information for TestResultWebApp	4
2.2.5	Append to existing execution result	5
2.3	Display on TestResultWebApp	7
3	CDataBase.py	8
3.1	Class: CDataBase	8
3.1.1	Method: connect	8
3.1.2	Method: disconnect	9
3.1.3	Method: cleanAllTables	9
3.1.4	Method: sCreateNewTestResult	9
3.1.5	Method: nCreateNewFile	10
3.1.6	Method: vCreateNewHeader	11
3.1.7	Method: nCreateNewSingleTestCase	13
3.1.8	Method: nCreateNewTestCase	14
3.1.9	Method: vCreateTags	15
3.1.10	Method: vSetCategory	15
3.1.11	Method: vUpdateStartTime	16
3.1.12	Method: arGetCategories	16
3.1.13	Method: vCreateAbortReason	16
3.1.14	Method: vCreateReanimation	17
3.1.15	Method: vCreateCCRdata	17
3.1.16	Method: vFinishTestResult	17
3.1.17	Method: vUpdateEvtbls	17
3.1.18	Method: vUpdateEvtbl	18
3.1.19	Method: vEnableForeignKeyCheck	18
3.1.20	Method: sGetLatestFileID	18
3.1.21	Method: vUpdateFileEndTime	18
3.1.22	Method: vUpdateResultEndTime	19
3.1.23	Method: bExistingResultID	19

<u>CONTENTS</u>	<u>CONTENTS</u>
3.1.24 Method: arGetProjectVersionSWByID	19
4 pytestlog2db.py	20
4.1 Function: collect_xml_result_files	20
4.2 Function: validate_xml_result	20
4.3 Function: is_valid_uuid	21
4.4 Function: is_valid_config	21
4.5 Function: parse_pytest_xml	22
4.6 Function: get_branch_from_swversion	22
4.7 Function: get_test_result	22
4.8 Function: process_component_info	23
4.9 Function: process_config_file	23
4.10 Function: process_test	24
4.11 Function: process_suite	24
4.12 Function: PyTestLog2DB	25
4.13 Class: Logger	25
4.13.1 Method: config	26
4.13.2 Method: log	26
4.13.3 Method: log_warning	26
4.13.4 Method: log_error	27
5 Appendix	28
6 History	29

Chapter 1

Introduction

PyTestLog2DB is a command-line tool that enables you to import [pytest](#) XML result files into TestResultWebApp's database for presenting an overview about the whole test execution and detail of each test result.

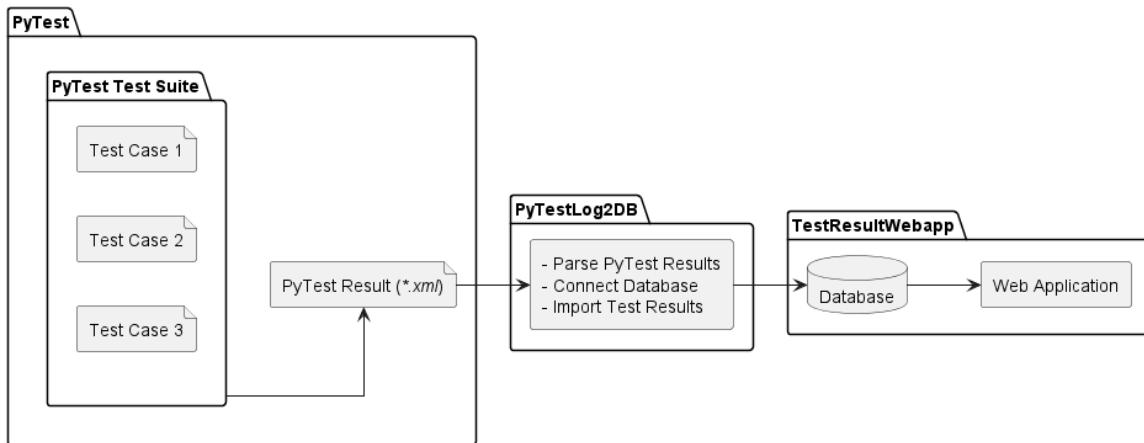


Figure 1.1: Tool data flow

PyTestLog2DB tool requires several arguments, including the location of the pytest XML result file(s) to parse all information of test execution result and TestResultWebApp's database credential for importing that result.

[TestResultWebApp](#) requires some mandatory information to manage and display the test result properly, but the generated `*xml` file contains only the basic test result information. So that, **PyTestLog2DB** tool will set those required information with [default values](#).

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

Finally, **PyTestLog2DB** also allows you to append existing results in the database, which is helpful when you need to update previous test results or add the missing XML result file(s) from previous tool execution.

CHAPTER 2. DESCRIPTION

Chapter 2

Description

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

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

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

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

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

2.2 Tool features

2.2.1 Usage

Use below command to get tools's usage:

```
PyTestLog2DB -h
```

The tool's usage should be showed as below:

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

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

positional arguments:
resultxmlfile      absolute or relative path to the pytest JUnit XML report
                   file or directory of report files to be imported.
server            server which hosts the database (IP or URL).
user              user for database login.
password          password for database login.
database          database schema for database login.

optional arguments:
-h, --help         show this help message and exit
-v, --version      version of the PyTestLog2DB importer.
--recursive       if set, then the path is searched recursively for output
                  files to be imported.
--dryrun          if set, then verify all input arguments (includes DB connection)
                  and show what would be done.
--append          is used in combination with --UUID UUID. If set, allow to append
                  new result(s) to existing execution result UUID in --UUID argument.
--UUID UUID        UUID used to identify the import and version ID on webapp.
```

CHAPTER 2. DESCRIPTION2.2. TOOL FEATURES

```

    ↵ import.           If not provided PyTestLog2DB will generate an UUID for the whole ↵
--variant VARIANT   variant name to be set for this import.
--versions VERSIONS metadata: Versions (Software;Hardware;Test) to be set for this import
                     (semicolon separated)
--config CONFIG     configuration json file for component mapping information.

```

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

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

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

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

2.2.2 Verify provided arguments

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

When executing in dryrun mode, **PyTestLog2DB** will:

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

This feature will help you to ensure that there is no error when executing **PyTestLog2DB** tool (normal mode) to create new record(s) and update TestResultWebApp's database.

2.2.3 Searching **.xml* result file(s)

The first argument `resultxmlfile` of **PyTestLog2DB** can be a single file or the folder that contains multiple result files.

When the folder is used, **PyTestLog2DB** will only search for **.xml* file(s) under given directory and exclude any file within subdirectories as default.

In case you have result file(s) under the subdirectory of given folder and want these result files will also be imported, the optional argument `--recursive` should be used when executing **PyTestLog2DB** command.

When `--recursive` argument is set, **PyTestLog2DB** will walk through the given directory and its subdirectories to discover and collect all available **.xml* for importing.

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

```

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

```

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

2.2.4 Handle essential information for TestResultWebApp

Default values

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

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

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

- **Project:** `PyTest`
- **Software version:** the execution time `%Y%m%d_%H%M%S` as default value. E.g. `20221128_143547`
- **Hardware version:** empty string
- **Test version:** empty string
- **Component:** `unknown`
- **Test tool:** the combination of current Python and pytest version. E.g. `PyTest 6.2.5 (Python 3.9.0)`
- **Tester:** The current user

Specify essential information with optional arguments

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

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



Notice

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

- `"testtool"` to specify the **Test toolchain** as `string` value.
- `"tester"` to specify the **Test user** as `string` value.
- `"components"` to specify the **Component** information which will be displayed on [TestResultWebApp](#). Value can be:
 - * `string` : to specify the same **Component** for all test casea within this execution.
 - {
 `"components" : "atest",`
 `...`
}
 - * `object` : to define the mapping json object between **Component** info as key and a test case `classname` (list of `classname`) (*.robot file) as value.

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

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

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

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

- Test case(s) which its `classname` contains `test-data.test-tsclass.TestSuite1` is belong `Testsuite1` component
- Test case(s) which its `classname` contains `test-data.test-tsclass.TestSuite2` is belong `TestSuite2` component
- `Others` component consists of all test case(s) which its `classname` contains `test-data.test-ts1` and `test-data.test-ts2`.

Hint

The `classname` of test case in the generated pytest result `*.xml` file is the combination of directory, test file and class of defined pytest test case.

Therefore, you can use directory information for component mapping to group all test cases in a folder as the same component.

In addition, you can use the optional argument `--junit-prefix=str` when executing pytest to set the prefix to the `classname` of the test case in the result `*.xml` file. After that, you can use that prefix information for component mapping when importing the result `*.xml` file with the `PyTestLog2DB` tool.

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

2.2.5 Append to existing execution result

`PyTestLog2DB` also allows you to append new test result(s) (missing from previous import, on other test setup, ...) into the existing execution result (identified by the `UUID`) in `TestResultWebApp`'s database. The combination of optional arguments `--UUID <UUID>` and `--append` should be used in this case.

The `--append` makes `PyTestLog2DB` run in append mode and the `--UUID <UUID>` is used to specify the existing UUID of execution result to be appended.

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

The command will be used as below:

```
python -m PyTestLog2DB output.xml localhost testuser testpw testdb --UUID ↵
    ↵ c7991c07-4de2-4d65-8568-00c5556c82aa --append
```

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

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

```
FATAL ERROR: Execution result with UUID 'c7991c07-4de2-4d65-8568-00c5556c82aa' is ↵
    ↵ already existing.
    Please use other UUID (or remove '--UUID' argument from your command) ↵
    ↵ for new execution result.
```

CHAPTER 2. DESCRIPTION2.2. TOOL FEATURES

```
Or add '--append' argument in your command to append new result(s) to ↵
↳ this existing UUID.
```

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

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

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

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

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

```
FATAL ERROR: '--append' argument should be used in combination with '--UUID UUID` ↵
↳ argument
```

Notice

When using append mode and `project` / `variant`, `version_sw` are provided within `--variant VARIANT`, `--versions VERSIONS` or `--config CONFIG` arguments, they will be validated with the existing values in database.

An error will be raised in case the given value is not matched with the existing ones. E.g:

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



CHAPTER 2. DESCRIPTION2.3. DISPLAY ON TESTRESULTWEBAPP

2.3 Display on TestResultWebApp

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

Dashboard view:

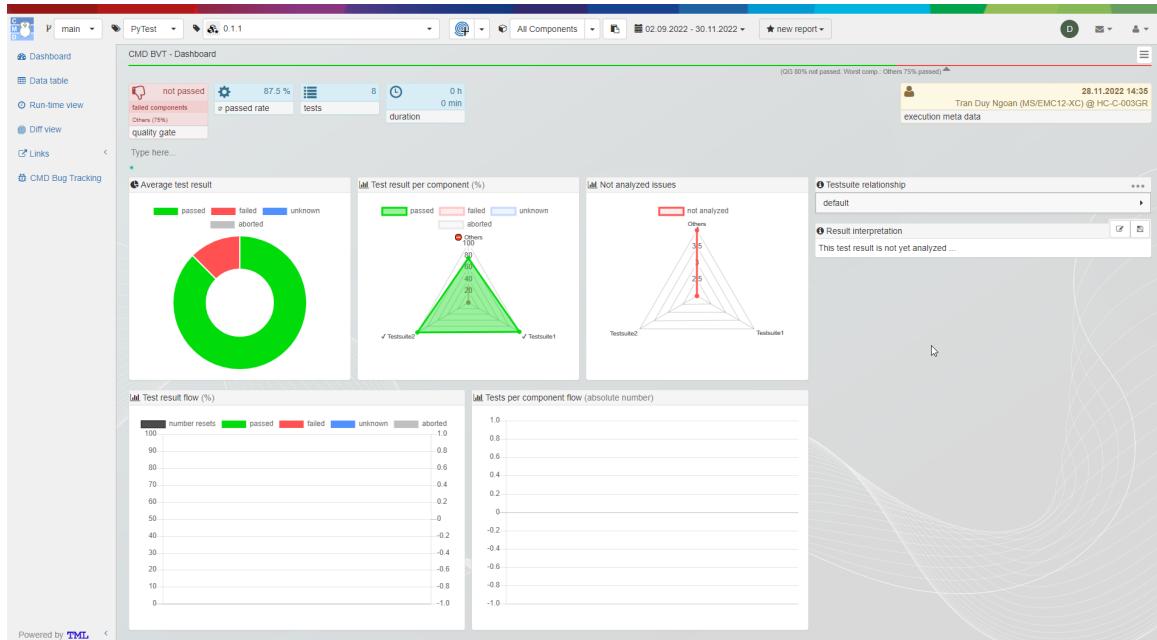


Figure 2.1: Dashboard view

Datatable view:

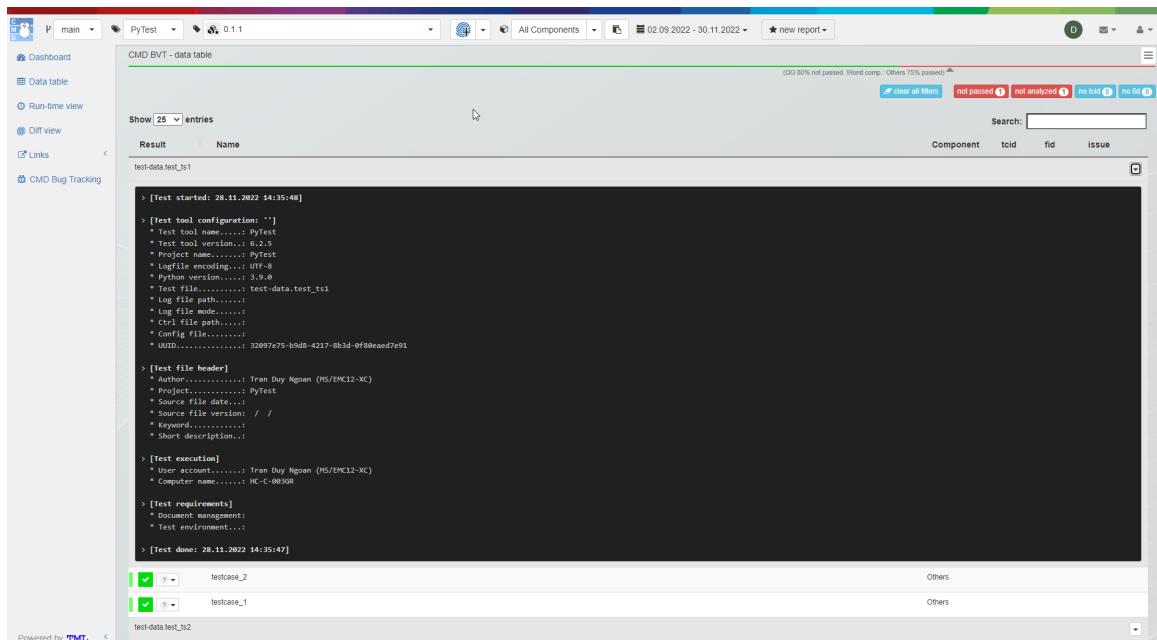


Figure 2.2: Datatable view

CHAPTER 3. CDATABASE.PY

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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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. Default 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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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 repetition 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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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_EXECUTE MANY` 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 repetition 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.

CHAPTER 3. CDATABASE.PY3.1. CLASS: CDATABASE

- `_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: vUpdateStartTime

Create start-end time entry.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
UUID of test result.
- `_tbl_result_time_start`
/ Condition: required / Type: str /
Result start time as format `%Y-%m-%d %H:%M:%S`.
- `_tbl_result_time_end`
/ Condition: required / Type: str /
Result end time as format `%Y-%m-%d %H:%M:%S`.

Returns:

(no returns)

3.1.12 Method: arGetCategories

Get existing categories.

Arguments:

(no arguments)

Returns:

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

3.1.13 Method: vCreateAbortReason

Create abort reason entry.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
UUID of test result.
- `_tbl_abort_reason`
/ Condition: required / Type: str /
Abort reason.
- `_tbl_abort_message`
/ Condition: required / Type: str /
Detail message of abort.

Returns:

(no returns)

3.1.14 Method: vCreateReanimation

Create reanimation entry.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
UUID of test result.
- `_tbl_num_of_reanimation`
/ Condition: required / Type: int /
Counter of target reanimation during execution.

Returns:

(*no returns*)

3.1.15 Method: vCreateCCRdata

Create CCR data per test case.

Arguments:

- `_tbl_test_case_id`
/ Condition: required / Type: int /
test case ID.
- `lCCRdata`
/ Condition: required / Type: list /
list of CCR data.

Returns:

(*no returns*)

3.1.16 Method: vFinishTestResult

Finish upload:

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

Arguments:

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

Returns:

(*no returns*)

3.1.17 Method: vUpdateEvtbls

Call update_evtbls stored procedure.

Arguments:

(*no arguments*)

Returns:

(*no returns*)

3.1.18 Method: vUpdateEvtbl

Call update_evtbl stored procedure to update provided test_result_id.

Arguments:

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

Returns:

(no returns)

3.1.19 Method: vEnableForeignKeyCheck

Switch foreign_key_checks flag.

Arguments:

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

Returns:

(no returns)

3.1.20 Method: sGetLatestFileID

Get latest file ID from `tbl_file` table.

Arguments:

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

Returns:

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

3.1.21 Method: vUpdateFileEndTime

Update test file end time.

Arguments:

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

Returns:

(no returns)

3.1.22 Method: vUpdateResultEndTime

Update test result end time.

Arguments:

- `_tbl_test_result_id`
/ Condition: required / Type: str /
Result UUID to be updated.
- `_tbl_result_time_end`
/ Condition: required / Type: str /
Result end time as format `%Y-%m-%d %H:%M:%S`.

Returns:

(no returns)

3.1.23 Method: bExistingResultID

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

Arguments:

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

Returns:

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

3.1.24 Method: arGetProjectVersionSWByID

Get the project and version_sw information of given `test_result_id`

Arguments:

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

Returns:

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

CHAPTER 4. PYTESTLOG2DB.PY

Chapter 4

pytestlog2db.py

4.1 Function: collect_xml_result_files

Collect all valid Robot xml result file in given path.

Arguments:

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

Returns:

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

4.2 Function: validate_xml_result

Verify the given xml result file is valid or not.

Arguments:

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

Returns:

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

4.3 Function: is_valid_uuid

Verify the given UUID is valid or not.

Arguments:

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

Returns:

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

4.4 Function: is_valid_config

Validate the json configuration base on given schema.

Default schema supports below information:

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

Arguments:

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

Returns:

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

4.5 Function: parse_pytest_xml

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

Arguments:

- `xmlfiles`
/ *Condition: required / Type: str /*
Path to pytest *.xml result file(s).

Returns:

- `oMergedTree`
/ *Type: etree.Element object /*
The result object which is parsed from provided pytest *.xml result file(s).

4.6 Function: get_branch_from_swversion

Get branch name from software version information.

Convention of branch information in suffix of software version:

- All software version with .0F is the main/feature branch. The leading number is the current year. E.g. 17.0F03
- All software version with .1S, .2S, ... is a stabl 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: type /`
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: tpyle /

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

4.9 Function: process_config_file

Parse information from configuration file:

- component:

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

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

Arguments:

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

Returns:

- dConfig
/ Type: dict /
 Configuration object.

4.10 Function: process_test

Process test case data and create new test case record.

Arguments:

- `db`
/ Condition: required / Type: CDataBase object /
CDataBase object.
- `test`
/ Condition: required / Type: etree.Element object /
Robot test object.
- `file_id`
/ Condition: required / Type: int /
File ID for mapping.
- `test_result_id`
/ Condition: required / Type: str /
Test result ID for mapping.
- `component_name`
/ Condition: required / Type: str /
Component name which this test case is belong to.
- `test_number`
/ Condition: required / Type: int /
Order of test case in file.
- `start_time`
/ Condition: required / Type: datetime object /
Start time of testcase.

Returns:

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

4.11 Function: process_suite

Process to the lowest suite level (test file):

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

Arguments:

- `db`
/ Condition: required / Type: CDataBase object /
CDataBase object.
- `suite`
/ Condition: required / Type: etree.Element object /
Robot suite object.

CHAPTER 4. PYTESTLOG2DB.PY4.12. FUNCTION: PYTESTLOG2DB

- `_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 verify all input arguments (includes DB connection) and show what would be done.
 - append : if True, then allow to append new result(s) to existing execution result UUID which is provided by --UUID argument.
 - UUID : UUID used to identify the import and version ID on TestResultWebApp.
 - variant : variant name to be set for this import.
 - versions : metadata: Versions (Software;Hardware;Test) to be set for this import.
 - config : configuration json file for component mapping information.

Returns:*(no returns)***4.13 Class: Logger**

Imported by:

[CHAPTER 4. PYTESTLOG2DB.PY](#)[4.13. CLASS: LOGGER](#)

```
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:

CHAPTER 4. PYTESTLOG2DB.PY4.13. CLASS: LOGGER

- `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

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	PyTestLog2DB
Version	0.2.8
Date	14.12.2023
Description	Imports pytest result(s) to TestResultWebApp database
Package URL	python-pytestlog2db
Author	Tran Duy Ngoan
Email	Ngoan.TranDuy@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

CHAPTER 6. HISTORY

Chapter 6

History

0.1.0	11/2022
<i>Initial version</i>	
0.1.1	22.11.2022
<i>Initial implementation of PyTestLog2DB tool</i>	
0.1.2	01.12.2022
<i>Add tool's document</i>	
0.1.4	24.02.2023
<ul style="list-style-type: none"> - Rename renamed key "components" in configuration json - Change behaviour of append mode without UUID to raise a fatal error - Improve console log for append mode and matched component for testcase - Add command line arguments -versions and -variant 	
0.1.5	27.02.2023
<i>Change behaviour of append mode with not existing UUID to raise a fatal error</i>	
0.1.6	01.03.2023
<i>Add a validation for xmlresultfile with junitxml schema</i>	
0.1.7	10.03.2023
<ul style="list-style-type: none"> - Add a validation for existing project/variant and version_sw in db when using append mode - Remove db maxlenlength handlers: truncations and validation 	
0.1.8	21.03.2023
<i>Support 4 byte characters for importing to db</i>	
0.1.9	20.04.2023
<ul style="list-style-type: none"> - Enhance console log with test case counter and component statistics - Add try/except for database access 	
0.2.6	08.05.2023
<i>Update README for publishing package to PyPI</i>	
0.2.8	14.12.2023
<i>Fix missing return of test duration when failed to import testcase</i>	

PyTestLog2DB.pdf

Created at 10.04.2024 - 12:34:02

by GenPackageDoc v. 0.41.1

8.13 TestResultWebApp

TestResultWebApp

v. 0.1.3

Thomas Pollerspöck

18.10.2022

CONTENTSCONTENTS

Contents

1	Introduction	1
2	Description	2
2.1	TestResultWebApp Architecture	2
2.2	Import Data	2
2.3	Data Visualization	3
2.3.1	Main menu	3
2.3.2	Dashboard View	3
2.3.3	DataTable View	7
2.3.4	Runtime View	11
2.3.5	Diff View	12
2.4	Developer guidance	14
2.4.1	How to run new TestResultWebApp instance	14
3	Appendix	17
4	History	18

CHAPTER 1. INTRODUCTION

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

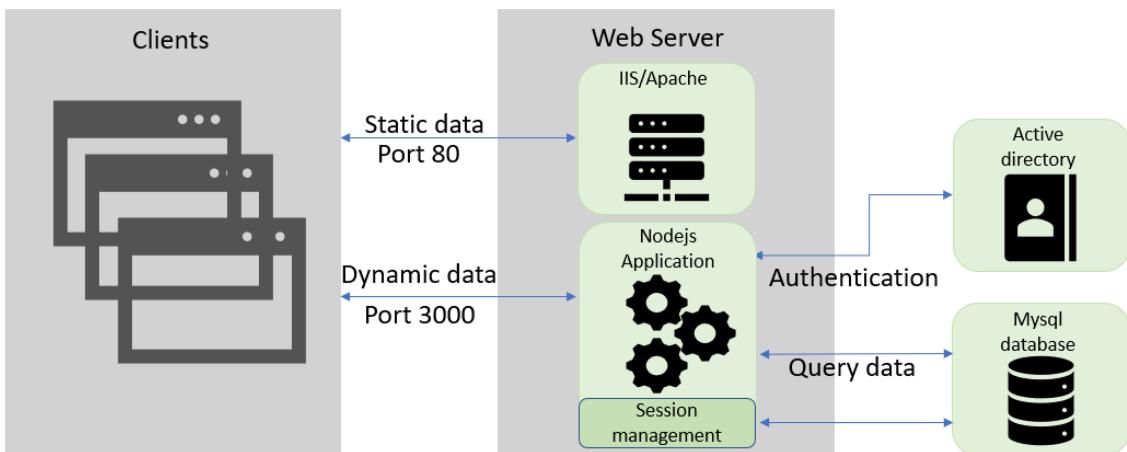
Description

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 [RobotLog2DB](#) which helps to import the Robot Framework result file(s) `output.xml` to the database of the TestResultWebApp.

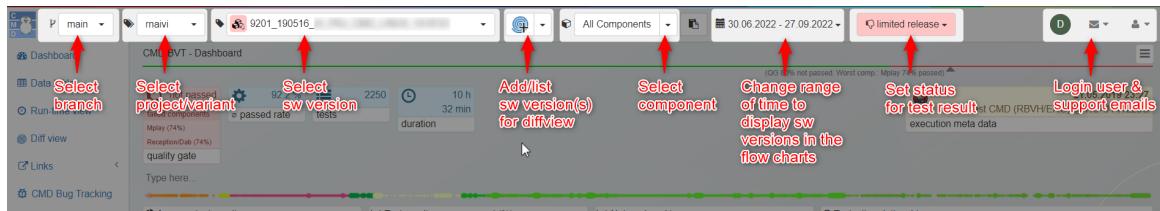
You will need to provide only the Robot Framework result file(s) and database's credential information, that RobotLog2DB will parse the test execution result data and interact with the provided database to import the data.

Please refer [RobotLog2DB's usage](#) and [its document](#) for more detail.

2.3 Data Visualization

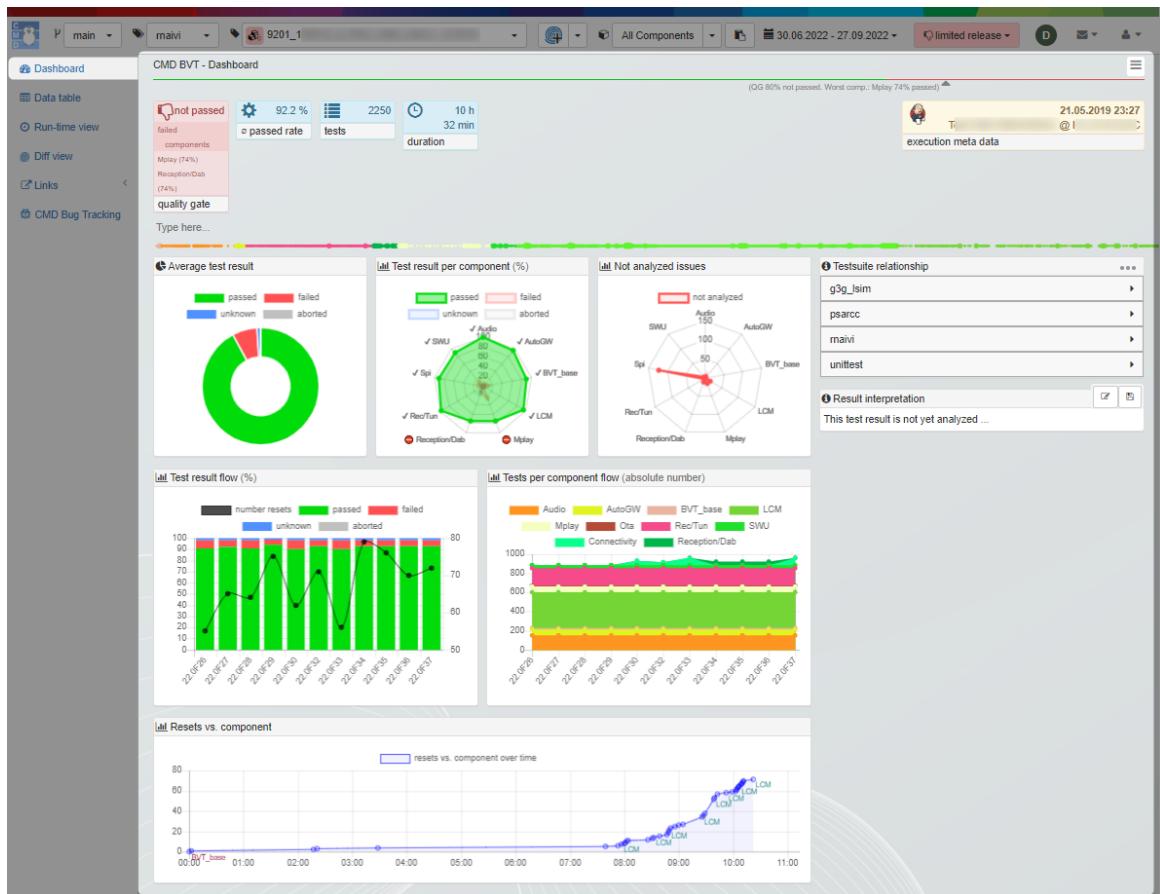
2.3.1 Main menu

The WebApp provides a main menu for selecting a specific **branch**, **project/variant** and **software version** to be displayed.



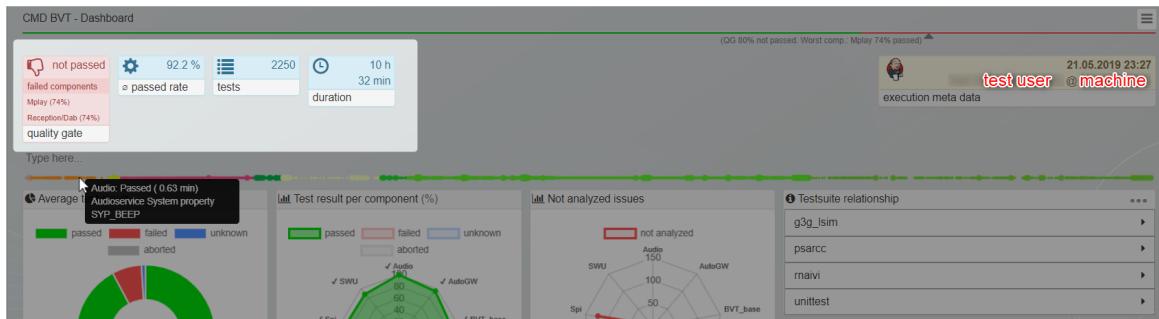
2.3.2 Dashboard View

The Dashboard view does not only provide an overview of test execution results but also shows the correlation between components within the result and relationship with other test execution results.



Result overview

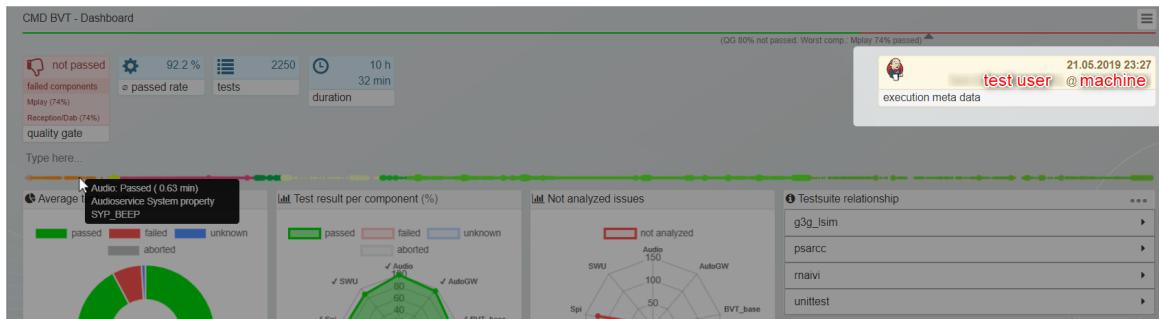
At the top left corner of the Dashboard view you find the test execution result statistics:

CHAPTER 2. DESCRIPTION2.3. DATA VISUALIZATION

Which contains:

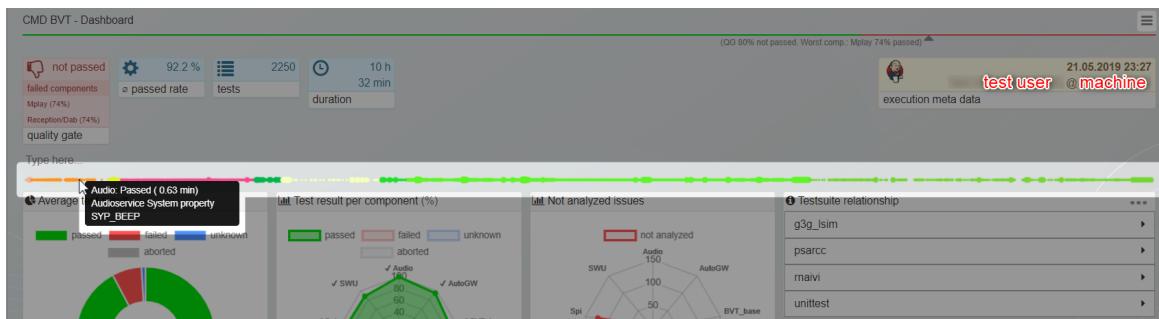
- Overall status (you can define a quality gate).
- Passed rate.
- Total number of executed test cases.
- Test execution duration.

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:



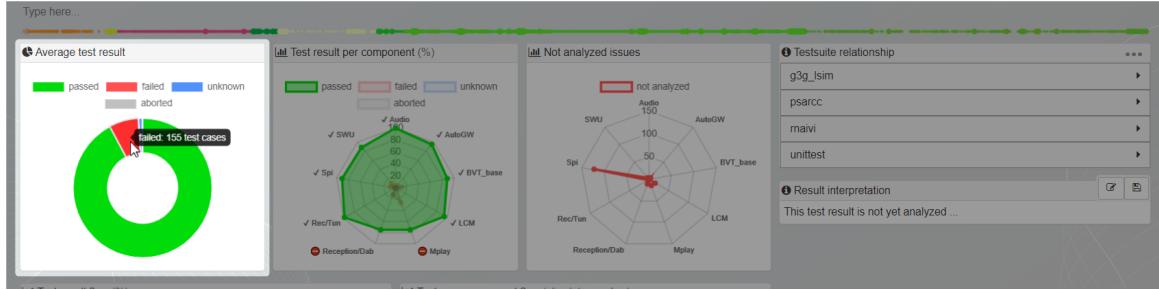
It provides:

- 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.

CHAPTER 2. DESCRIPTION**2.3. DATA VISUALIZATION****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.

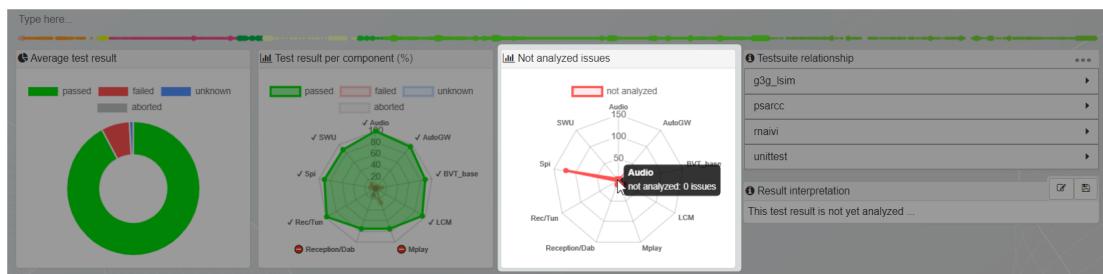
- **Test result per component** chart: provides a fast overview of test result percentages over all components. Here you can quickly see the quality of the system under test. Ideally the spider chart should be fully green. This means that all component tests result in 100% **passed**.

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**

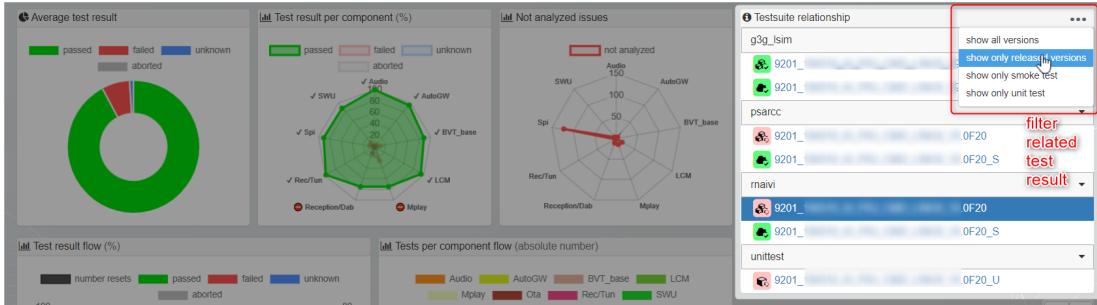
- **Testsuite relationship**: will let you know all the related test results (grouped by project/variant) of the current selected version.

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.

CHAPTER 2. DESCRIPTION**2.3. DATA VISUALIZATION**

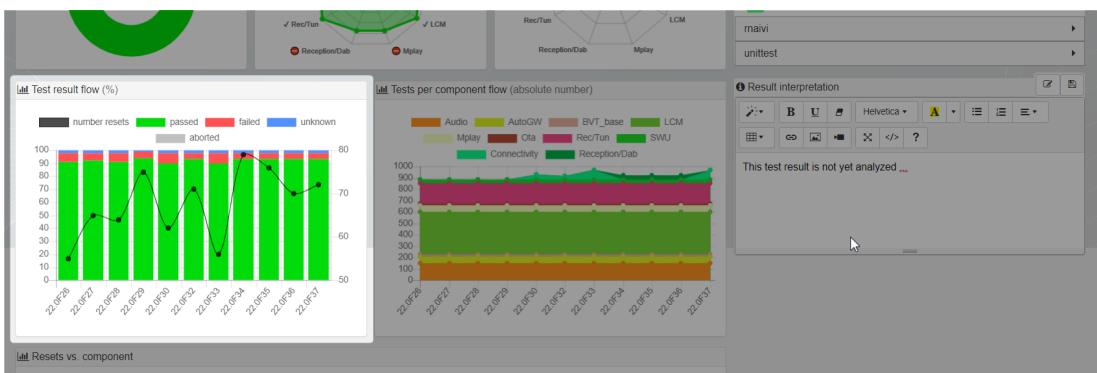
For example: the selected version have been executed for 4 projects/variants: *g3g_lsim*, *psarcc*, *rnaivi* and *unittest*. Each variant (except the *unittest*) has 2 test results (one for the smoke test and one for the whole test execution result).

Then, all the related test execution results will be displayed as below:



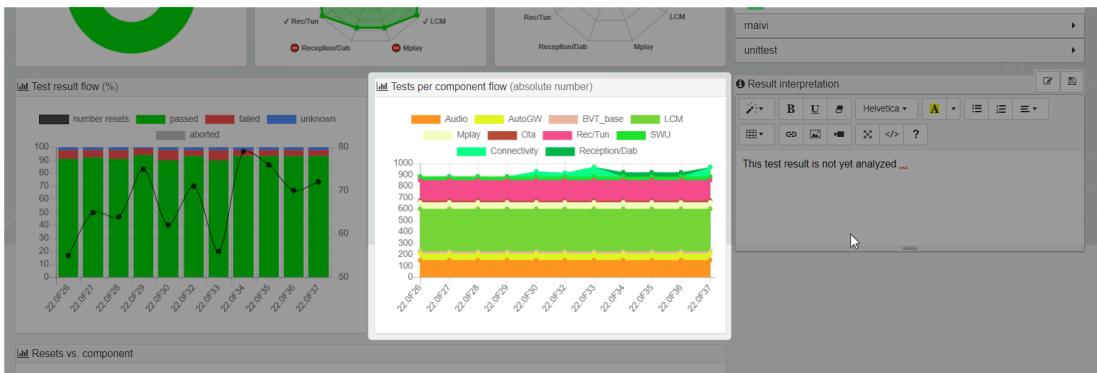
There is also the context menu (...) that allows you to filter the related test results.

- **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.

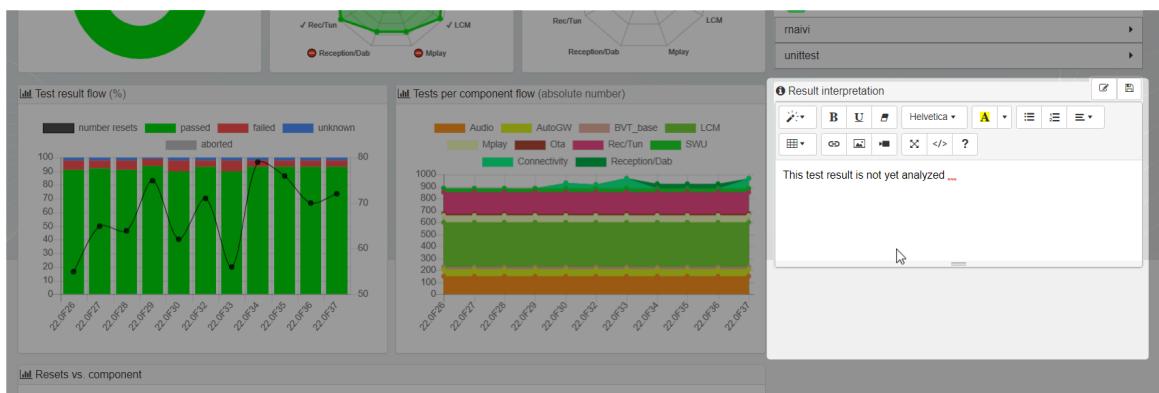
**Notice**

- !
- 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**).

CHAPTER 2. DESCRIPTION2.3. DATA VISUALIZATION**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 interpretation 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.

CHAPTER 2. DESCRIPTION

2.3. DATA VISUALIZATION

The screenshot shows a data table interface for test results. The table has columns: Result, Name, Component, tcid, fid, and issue. The 'Name' column contains file paths like 'D:/JenkinsClient/workspace/man_bvt_AIv_cmd_fs/ai_audio_tmtest/components/AMCommandPlugin/tests/testcases/connect/UTS_02_027.tml'. The 'Component' column shows 'Audio' repeated for most rows. The 'tcid' column contains IDs such as TC1458, TC1412, TC1460, TC1415, TC1421, TC3005, TC1440, TC1483, and TC1482. The 'fid' column contains 'SWF-5093' repeated. The 'issue' column contains ticket numbers like 'RTC-122345'. A 'users's comment' section is highlighted in blue, containing two entries from users Ashok and Manohara. The table shows 10 of 2250 entries.

Besides, you can also:

- determine how many test cases (entries) are displayed in the table page.

CHAPTER 2. DESCRIPTION2.3. DATA VISUALIZATION

CMD BVT - data table				
(QG 80% not passed. Worst comp.: Mplay 74% passed)				
<input type="button" value="clear all filters"/> <input type="button" value="not passed 176"/> <input type="button" value="not analyzed 176"/> <input type="button" value="no test 1655"/> <input type="button" value="no fid 170"/>				
Show 10 entries	Name	Component	tcid	fid
Results 25 / 100	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_027.tml			
	INVALID sink connection	Audio	TC1458	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_028.tml			
	Connect to AUX_2	Audio	TC1412	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_030.tml			
	change source from AUX_2 to TUNER_AM and sink internal Amp	Audio	TC1480	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_032.tml			
	change source from AUX_2 to TUNER_FM and sink internal Amp	Audio	TC1415	SWF-5093
	S 0118.17.1534 (Ashok ())			RTO-122346
	The RTC linked is not for the issue, but since there are some missing traces , we need to enable the audio stack traces to further analyze the issue.			
	S 0212.17.1549 (Manohara ())			
	test comment			
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_037.tml			
	change source from AUX_2 to MEDIA_PLAYER and sink internal Amp	Audio	TC1421	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_053.tml			
	change source from AUX_2 to AUX_2 and sink internal Amp	Audio	TC3006	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_054.tml			
	INVALID sink connection	Audio	TC1440	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_055.tml			
	Connect to TUNER_AM	Audio	TC1483	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_059.tml			
	change source from TUNER_AM to TUNER_FM and sink internal Amp	Audio	TC1483	SWF-5093
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_064.tml			
	change source from TUNER_AM to MEDIA_PLAYER and sink internal Amp	Audio	TC1462	SWF-5093
<input type="button" value="Copy"/>	<input type="button" value="Excel"/>			
Showing 1 to 10 of 2250 entries				
Previous <input type="button" value="1"/> <input type="button" value="2"/> <input type="button" value="3"/> <input type="button" value="4"/> <input type="button" value="5"/> ... <input type="button" value="225"/> Next				

- apply filters to display such as not **passed** test cases.

CMD BVT - data table				
(QG 60% not passed. Worst comp.: Mplay 74% passed)				
<input type="button" value="clear all filters"/> <input type="button" value="not passed 176"/> <input type="button" value="not analyzed 176"/> <input type="button" value="no test 1655"/> <input type="button" value="no fid 170"/>				
Show 10 entries	Name	Component	tcid	fid
Results 10 / 176	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_automotive_idai_gateway\tests\Results\CMD\AudioTC\html\Audio_Negative_Boundary.tml			
	GATEWAY_AUDIO_24_A Requesting Mute State to allocated channel 1 and state ON, wait on signal MuteState update	AutomotiveGateway		
	GATEWAY_AUDIO_28_A Requesting Mute State to allocated channel 3 and state ON, wait on signal MuteState update	AutomotiveGateway		
	GATEWAY_AUDIO_24_B Requesting Mute State to allocated channel 4 and state ON, wait on signal MuteState update	AutomotiveGateway		
	GATEWAY_AUDIO_25_B Requesting Mute State to allocated channel 6 and state ON, wait on signal MuteState update	AutomotiveGateway		
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_lcm_generic\tests\TML\Other_Features\Cvm_Handling\SPM_CvmEvent_ResetHandling.tml	LCM	TC8314	SWF-794
	Testing the high voltage reset handling	LCM	TC8314	SWF-794
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_lcm_generic\tests\TML\Other_Features\CyclicResetHandling\SPM_CyclicResetHandling_FD28_DevWup_Reduces_ResetCounter.tml	LCM	TC8256	SWF-7256
	Testing cyclic reset handling of LCM when /dev/wup reduces the reset counter by one on intended reset	LCM	TC8256	SWF-7256
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_lcm_generic\tests\TML\Other_Features\SpmCoreFi_Test\SPM_ResetCounter_Set.tml	LCM	TC3468	SWF-784
	Set ResetCounter	LCM	TC3468	SWF-784
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_lcm_generic\tests\TML\Other_Features\SpmCoreFi_Test\SPM_UpdateLockStates.tml	LCM	TC8309	SWF-7256
	Testing new FI UpdateLockStates of LCM	LCM	TC8309	SWF-7256
	D:\JenkinsClient\workspace\man_bvt_AVI_cmd_fs\ai_lcm_generic\tests\TML\Other_Features\Startup\SPM_StartupMultipleSyncBlocksTests.tml	LCM	TC6178	SWF-784
	Testing Startup Synchronization with multiple SPM_U32_SYNC_CONNECTED SYNC BLOCKS to start the process	LCM	TC6178	SWF-784
	Testing Startup Synchronization with multiple SPM_U32_SYNC_UP SYNC BLOCKS to start the process	LCM	TC6177	SWF-784
<input type="button" value="Copy"/>	<input type="button" value="Excel"/>			
Showing 1 to 10 of 176 entries				
Previous <input type="button" value="1"/> <input type="button" value="2"/> <input type="button" value="3"/> <input type="button" value="4"/> <input type="button" value="5"/> ... <input type="button" value="18"/> Next				

CHAPTER 2. DESCRIPTION

2.3. DATA VISUALIZATION

- search for the specific test case for reference.

CMW BVT - data table				
(2) 50% not passed, 100% errors - Missing 75% passes clear all filters not passed 116 not analyzed 116 no file 100 no FA 100 PFR 100				
Search: Media				
Result	Name	Component	tcid	fid
✓	D:\jenkins\client\workspace\man_btl_AVI_cmd_Navi_audio_imTest\components\AVCommandPlugins\tests\testcases\connectUTS_02_037.tst			
✓	change source from AUX_IN_TO_MEDIA_PLAYER and sink internal Amp	Audio	TC1421	SWF-6993
✓	D:\jenkins\client\workspace\man_btl_AVI_cmd_Navi_audio_imTest\components\AVCommandPlugins\tests\testcases\connectUTS_02_044.tst			
✓	change source from TUNER_AM to MEDIA_PLAYER and sink internal Amp	Audio	TC1462	SWF-6993
✓	D:\jenkins\client\workspace\man_btl_AVI_cmd_Navi_audio_imTest\components\AVCommandPlugins\tests\testcases\connectUTS_02_114.tst			
✓	change source from TUNER_FM to MEDIA_PLAYER and sink internal Amp	Audio	TC295	SWF-6993
✓	D:\jenkins\client\workspace\man_btl_AVI_cmd_Navi_automatic_Media_gateway_imTest\bin\test\binaries\CMS\Media\ChrmMedia_Overall.tst			
✓	GATEWAY_MEDIAPLAYER_01 MediaPlayer_Test_Verification for Play method	AutomotiveGateway		
✓	GATEWAY_MEDIAPLAYER_02 MediaPlayer_Test_Verification for Pause method	AutomotiveGateway		
✓	GATEWAY_MEDIAPLAYER_03 MediaPlayer_Test_Verification for Pause method without Play	AutomotiveGateway		
✓	GATEWAY_MEDIAPLAYER_04 MediaPlayer_Test_Verification for Stop method	AutomotiveGateway		
✓	GATEWAY_MEDIAPLAYER_05 MediaPlayer_Test_Verification for Stop method without Play	AutomotiveGateway		
✓	GATEWAY_MEDIAPLAYER_06 MediaPlayer_Test_Verification for Next method	AutomotiveGateway		
✓	GATEWAY_MEDIAPLAYER_07 MediaPlayer_Test_Verification for Previous method	AutomotiveGateway		

- get more information about test environment, configurations.

CMD BVT - data table				
Test Log				
Test Case Details				
Show 10 entries				Search: <input type="text"/>
Result	Name	Component	tcid	fid
D/JenkinsClient/workspace/man_bvt_AV1_cmd_fs/ai_audio_tmtest/components/AMCommandPlugin/tests/testcases/connect/UTS_02_027.tml				
<pre>> [Test started: 21.05.2019 23:30:02] > [Test tool configuration: 'rnaivi'] * Test tool name.....: TML Framework * Test tool version...: TESTVERSION 09.05.2019 / V 1.14.21.8 (1341) * Project name.....: 03g * LogFile encoding...: UTF-8 * Python version....: 2.7.0 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on Windows * Test file.....: D:/JenkinsClient/workspace/man_bvt_AV1_cmd_fs/ai_audio_tmtest/components/AMCommandPlugin/tests/testcases/connect/UTS_02_027.tml * Log file path.....: D:/JenkinsClient/workspace/man_bvt_AV1_cmd_fs/BVT/LogFile/Audio * Log file mode.....: 8/7/a' * Ctrl file path...: C:/AutoTest/ctrlFiles * Config file.....: 2:D:/JenkinsClient/workspace/man_bvt_AV1_cmd_fs/ai_sw/test/testsuites/BVT/config/test_config_rnaivi.xml * UUID.....: 91e8ba12-5ed6-4cde-8ac9-c5c802a22539 > [Test file header] * Author.....: * Project.....: * Source file date...: * Source file version: / / * Keyword.....: * Short description..: > [Test execution] * User account.....: Test_CMD (RBH/ENG) * Computer name.....: HC-UT41223C > [Test requirements] * Document management: * Test environment...: > [Test done: 21.05.2019 23:30:13]</pre>	Audio	TC1458	SWF-8093	
<input checked="" type="checkbox"/> <input type="button"/> INVAL link connection				
D/JenkinsClient/workspace/man_bvt_AV1_cmd_fs/ai_audio_tmtest/components/AMCommandPlugin/tests/testcases/connect/UTS_02_028.tml				
<input checked="" type="checkbox"/> <input type="button"/> Connect to AUX_2		Audio	TC1412	SWF-8093
D/JenkinsClient/workspace/man_bvt_AV1_cmd_fs/ai_audio_tmtest/components/AMCommandPlugin/tests/testcases/connect/UTS_02_030.tml				
<input checked="" type="checkbox"/> <input type="button"/> change source from AUX_2 to TUNER_AM and sink internal Amp		Audio	TC1480	SWF-8093

- get traceback information for not passed test case(s). A 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.

CHAPTER 2. DESCRIPTION

2.3. DATA VISUALIZATION

Result	Name	Component	tcid	fid	Issue
	D:\JenkinsClient\workspace\man_bt_AV_CMD_fsl_bt_mediaplayer\components\TMLTests\Bluetooth\0_1_TC_MP_BT_PLAY_PAUSE.tml	MediaPlayer	13380		
	BT_playback_PLAY Select an item	Sharmini Sundaramurthi (RBE) [EC03]			
	is not connected while performing the test. Please pair and connect a Bluetooth device and execute the tests.				
	BT_stop_BT_STOP Select an item	Sengutivel Nachimuthu (REIEU) [C06]			
	failed Due to unavailable of media source				
		Add comment			
	D:\JenkinsClient\workspace\man_bt_AV_CMD_fsl_bt_mediaplayer\components\TMLTests\Bluetooth\0_2_TC_MP_BT_NEXT.tml	MediaPlayer	13386		
	D:\JenkinsClient\workspace\man_bt_AV_CMD_fsl_bt_mediaplayer\components\TMLTests\Bluetooth\0_3_TC_MP_BT_RPT_LIST.tml	MediaPlayer	13386	20979	
	D:\JenkinsClient\workspace\man_bt_AV_CMD_fsl_bt_mediaplayer\components\TMLTests\Bluetooth\0_4_TC_MP_BT_RPT_OFF.tml	MediaPlayer	13386	20979	
	BT_NEXT_bt_RPT_OFF_check	MediaPlayer	13386	20979	

- copy data table or export them to an excel file.

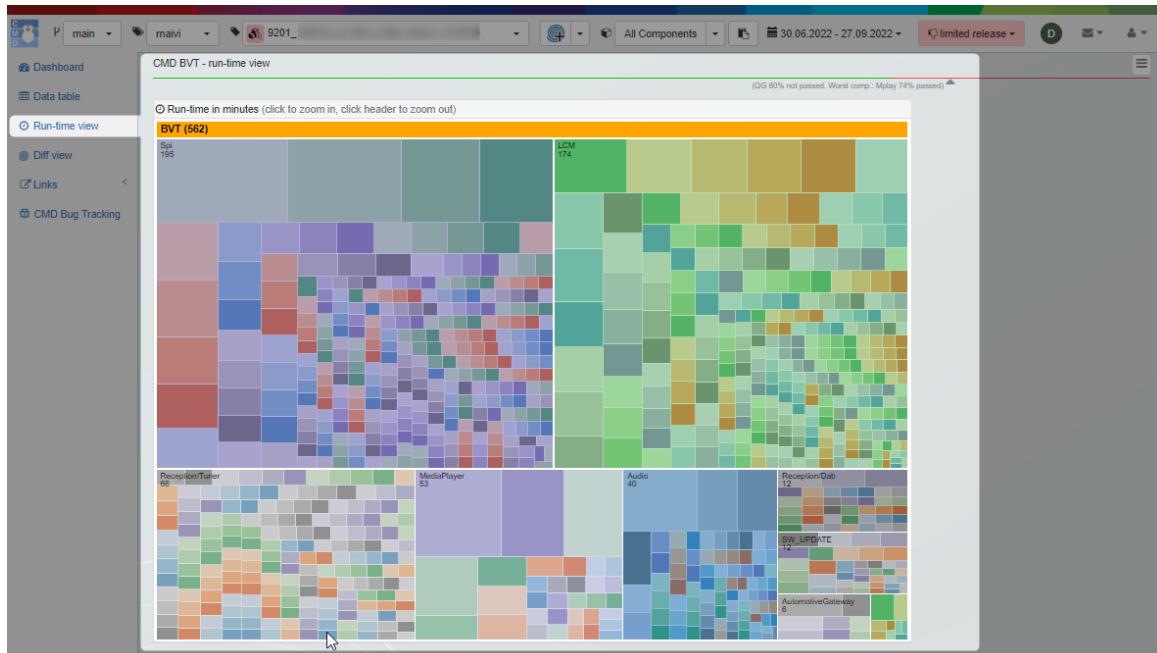
D:\JenkinsClient\workspace\man_bvt_AIV_Cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_055.tml				
	Connect to TUNER_AM	Audio	TC1483	SWF-8093
D:\JenkinsClient\workspace\man_bvt_AIV_Cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_059.tml	change source from TUNER_AM to TUNER_FM and sink internal Amp	Audio	TC1483	SWF-8093
D:\JenkinsClient\workspace\man_bvt_AIV_Cmd_fs\ai_audio_tmtest\components\AMCommandPlugin\tests\testcases\connect\UTS_02_064.tml	change source from TUNER_AM to MEDIA_PLAYER and sink internal Amp	Audio	TC1482	SWF-8093

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.



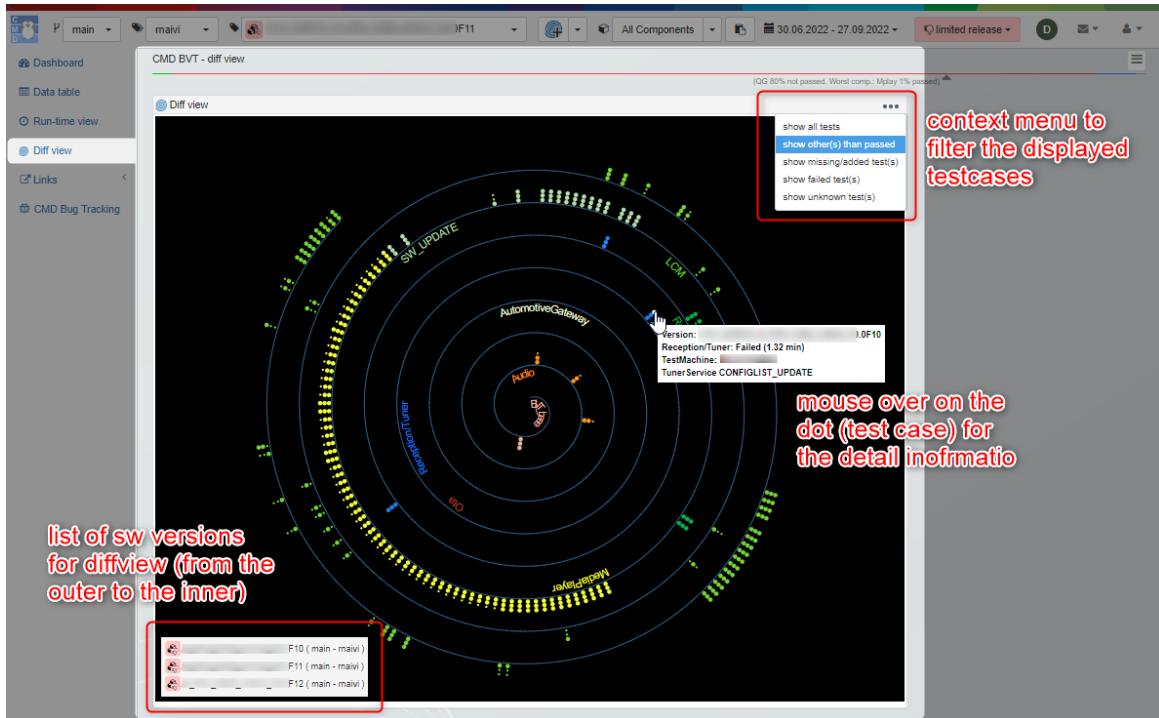
2.3.5 Diff View

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 spiral 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:

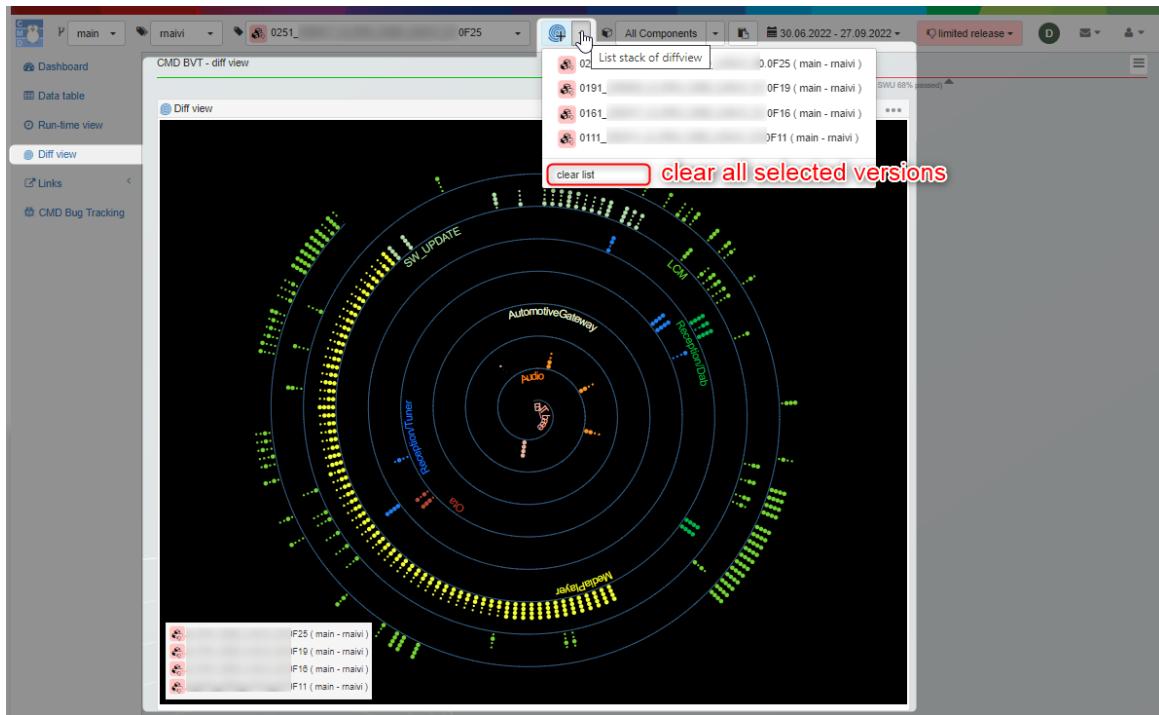
- see a test case result change (e.g from **passed** to **failed**).
- quickly find new or removed test case(s).
- recognize unstable test case(s)/component(s).

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 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.

CHAPTER 2. DESCRIPTION2.3. DATA VISUALIZATION

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.

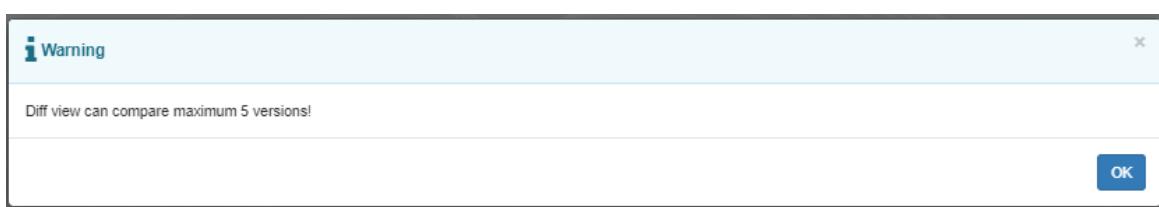
The dots which present the test cases along the spiral line (small dot is **passed** and bigger one for other status) are interactable. It means that you can:

- move the 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 environment: 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](#) should be installed.
- [MySQL server](#) should be installed.
- A cloned package's resource from [Github repo](#)

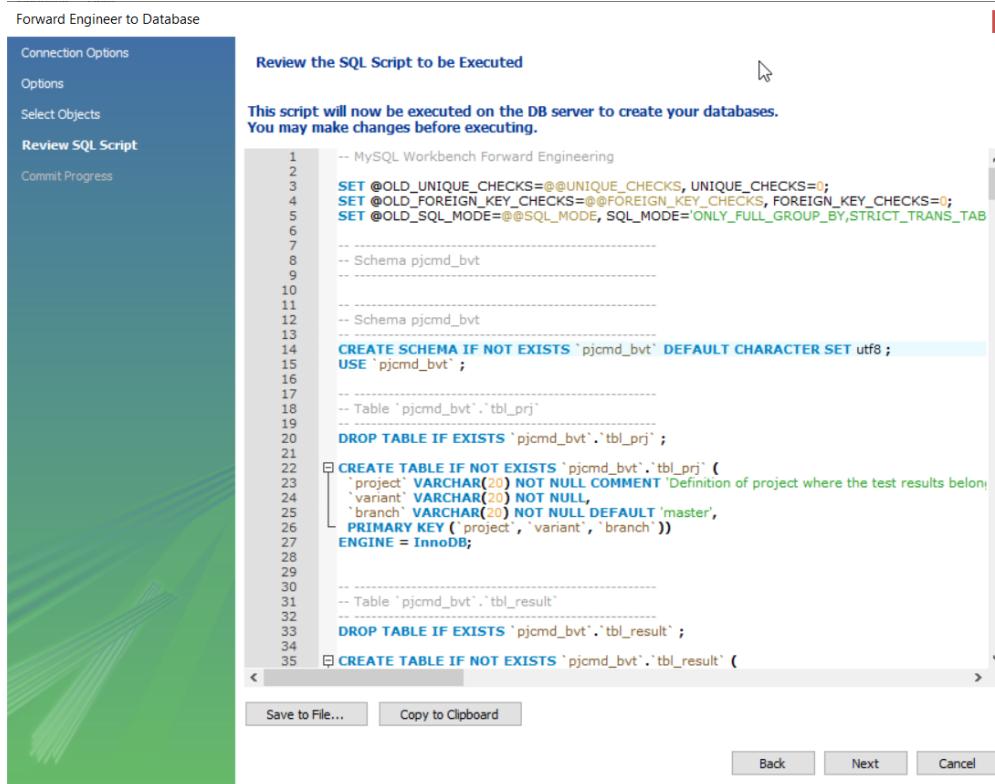
```
git clone https://github.com/test-fullautomation/testresultwebapp.git
```

- 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 **Forward Engineer to Database** feature of MySQL Workbench.

```
EER Diagram > Database > Forward Engineer... > your schema
```

CHAPTER 2. DESCRIPTION2.4. DEVELOPER GUIDANCE**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 [RobotLog2DB 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:

CHAPTER 2. DESCRIPTION2.4. DEVELOPER GUIDANCE

```
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/dashboard/` 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/dashboard/")));

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

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

Chapter 4

History

0.1.0	07/2022
<i>Initial version</i>	
0.1.1	09/2022
<i>Update README file and package's document</i>	
0.1.2	05/10/2022
<i>Fix findings with package's document</i>	

TestResultWebApp.pdf

*Created at 10.04.2024 - 12:34:04
by GenPackageDoc v. 0.41.1*

Chapter 9

Appendix

9.1 Installed Python Modules

This chapter contains a list of installed Python modules together with their version numbers.

Based on: Python 3.9.2 (default, Mar 3 2021, 22:31:41) [Clang 11.1.0]

Identified 198 packages.

aenum : 3.1.15	elementpath : 4.4.0	mobly : 1.12.2
alabaster : 0.7.16	entrypoint2 : 1.1	msgpack : 1.0.8
altgraph : 0.17.4	et-xmlfile : 1.1.0	msl-loadlib : 0.10.0
appdirs : 1.4.4	exceptiongroup : 1.2.0	mss : 9.0.1
Appium-Python-Client : 3.1.1	filelock : 3.13.4	mypy-extensions : 1.0.0
argparse-addons : 0.12.0	fonttools : 4.51.0	mysqlclient : 2.2.4
astroid : 3.1.0	func_timeout : 4.3.5	networkx : 3.2.1
attrs : 23.2.0	futures : 3.0.5	nodeenv : 1.8.0
Babel : 2.14.0	GenPackageDoc : 0.41.1	numpy : 1.26.4
bcrypt : 4.1.2	h11 : 0.14.0	odxtools : 6.7.1
beautifulsoup4 : 4.12.3	httpserver : 1.1.0	opencv-python-headless : 4.9.0.80
bitstruct : 8.19.0	identify : 2.5.35	openpyxl : 3.1.2
black : 24.3.0	idna : 3.6	ordereddict : 1.1
bokeh : 3.4.0	imageio : 2.34.0	outcome : 1.3.0.post0
certifi : 2024.2.2	imagesize : 1.4.1	packaging : 24.0
cffi : 1.16.0	importlib_metadata : 7.1.0	paho-mqtt : 2.0.0
cfgv : 3.4.0	importlib_resources : 6.4.0	pandas : 2.2.1
chardet : 5.2.0	imutils : 0.5.4	pandoc : 2.3
charset-normalizer : 3.3.2	iniconfig : 2.0.0	paramiko : 3.4.0
click : 8.1.7	isort : 5.13.2	path : 16.14.0
colorama : 0.4.6	jeepney : 0.8.0	path.py : 12.5.0
configparser : 6.0.1	Jinja2 : 3.1.3	pathspec : 0.12.1
contourpy : 1.2.1	JsonPreprocessor : 0.4.0	pickleDB : 0.9.2
cryptography : 42.0.5	jsonschema : 4.21.1	pillow : 10.3.0
cycler : 0.12.1	jsonschema-specifications : 2023.12.1	platformdirs : 4.2.0
dasbus : 1.7	kitchen : 1.2.6	pluggy : 1.4.0
decorator : 5.1.1	kiwisolver : 1.4.5	plumbum : 1.8.2
deprecation : 2.1.0	lazy_loader : 0.4	ply : 3.11
dill : 0.3.8	lxml : 5.2.1	portpicker : 1.6.0
distlib : 0.3.8	markdown-it-py : 3.0.0	pre-commit : 3.7.0
docopt : 0.6.2	markdownify : 0.12.1	pre-commit-hooks : 4.6.0
docutils : 0.20.1	MarkupSafe : 2.1.5	psutil : 5.9.8
doipclient : 1.1.1	matplotlib : 3.8.4	pycairo : 1.26.0
dotdict : 0.1	mccabe : 0.7.0	pycparser : 2.22
dotmap : 1.3.30	mdurl : 0.1.2	pyfranca : 0.4.1
doxypy : 0.8.8.7	MicroserviceBase : 0.1.1	Pygments : 2.17.2
EasyProcess : 1.1	MicroserviceClewareSwitch : 0.1.2	PyGObject : 3.44.1

pyinstaller : 6.5.0	robotframework-excellib : 2.0.1	snowballstemmer : 2.2.0
pyinstaller-hooks-contrib : 2024.3	robotframework-extensions-collection : 0.10.0	sortedcontainers : 2.4.0
pylint : 3.1.0	robotframework-ftplib : 1.9	soupsieve : 2.5
PyNaCl : 1.5.0	robotframework-imagecompare : 0.2.0	Sphinx : 7.2.6
pypandoc_binary : 1.13	robotframework-listenerlibrary : 1.0.3	sphinxcontrib-applehelp : 1.0.8
pyparsing : 3.1.2	robotframework-mqtlibrary : 0.7.1.post3	sphinxcontrib-devhelp : 1.0.6
pyrsistent : 0.20.0	robotframework-pythonlibcore : 4.4.1	sphinxcontrib-htmlhelp : 2.0.5
pyscreenshot : 3.1	robotframework-qconnect-base : 1.1.3	sphinxcontrib-jsmath : 1.0.1
pyserial : 3.4	robotframework-qconnect-winapp : 1.0.3	sphinxcontrib-qthelp : 1.0.7
pyshark : 0.6	robotframework-requests : 0.9.7	sphinxcontrib-serializinghtml : 1.1.10
PySocks : 1.7.1	robotframework-roboocop : 5.0.4	tabulate : 0.9.0
pytest : 8.1.1	robotframework-robotlog2db : 1.4.1	termcolor : 2.4.0
pytest-mock : 3.14.0	robotframework-robotlog2rqm : 1.2.3	tifffile : 2024.2.12
PyTestLog2DB : 0.2.8	robotframework-sshlibrary : 3.8.0	tk : 0.1.0
python-can : 4.3.1	robotframework-testsuitesmanagement : 0.7.6	tomli : 2.0.1
python-dateutil : 2.9.0.post0	robotframework-tidy : 4.11.0	tomlkit : 0.12.4
PythonExtensionsCollection : 0.15.1	rpds-py : 0.18.0	tornado : 6.4
pyttsx : 1.1	ruamel.yaml : 0.18.6	trio : 0.25.0
pytz : 2024.1	ruamel.yaml.clib : 0.2.8	trio-websocket : 0.11.1
PyYAML : 6.0.1	ruff : 0.3.5	typing_extensions : 4.11.0
pyzmq : 25.1.2	schedule : 1.2.1	tzdata : 2024.1
QLed : 1.3.1	scikit-image : 0.22.0	urllib3 : 2.2.1
referencing : 0.34.0	scipy : 1.13.0	virtualenv : 20.25.1
requests : 2.31.0	scp : 0.14.5	Wave : 0.0.2
rich : 13.7.1	selenium : 4.19.0	wrapt : 1.16.0
rich-click : 1.7.4	six : 1.16.0	wsproto : 1.2.0
robotframework : 7.0	sniffio : 1.3.1	xmleschema : 3.2.1
robotframework-appiumlibrary : 2.0.0		xmldict : 0.13.0
robotframework-dbus : 0.1.3		xyzservices : 2024.4.0
robotframework-dependencylibrary : 4.0.1		zipp : 3.18.1

Additionally required Python packages can be installed in this way:

1. Windows:

```
"%RobotPythonPath%/python.exe" -m pip install --proxy <proxy address> <packagename>
```

2. Linux:

```
"${RobotPythonPath}/python3" -m pip install --proxy <proxy address> <packagename>
```

The full path and name of the Python interpreter is required in these command lines because the **RobotFramework AIO** installer does not modify the environment of the computer (except the setup of some environment variables).

The proxy address is an option and depends on the conditions under which your company grants the access to the internet.