

Specification of Robot Framework at Bosch

Thomas Pollerspöck

Nguyen Huynh Tri Cuong

Tran Duy Ngoan

Mai Dinh Nam Son

Tran Hoang Nguyen

June 2022



! Information !

Dear reader,

Information to users.

Contents

1	Code analysis	4
2	Library	5
3	Apertis Pro	6
3.1	How to use DLT	6
3.1.1	dlt-daemon on Apertis	6
3.1.2	Apertis Pro firewall configuration	6
3.1.3	DLTSelfTestApp	6
3.1.4	QConnectDLTLibrary	7
4	Library documentation	9
4.1	GenPackageDoc	10
4.2	PythonExtensionsCollection	35
4.3	RobotframeworkExtensions	70
4.4	JsonPreprocessor	79
4.5	RobotFramework_Testsuites	87
4.6	QConnectBase	105
4.7	RobotResults2RQM	133
4.8	RobotResults2DB	151
4.9	TestResultWebApp	170

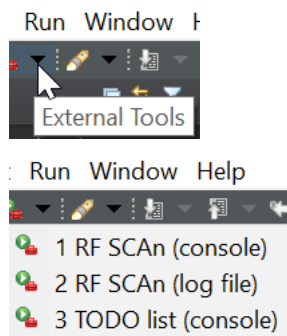
Todo list

1. Code analysis

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

Start Eclipse and select a file or a folder in Project Explorer. A missing selection causes an error.

The External Tools menu provides three preconfigured settings to start the analysis (the numbering within the context menu depends on the overall number of External Tools within your Eclipse configuration).



RF SCAN is the abbreviation for "**R**obot **F**ramework **S**tatic **C**ode **A**nalysis".

1. RF SCAN (console)

Output printed to console

2. RF SCAN (log file)

Output printed to log file: %ROBOTLOGPATH%\Robocop.log

3. TODO list (console)

It is possible to use the string TODO in code comments to indicate that still something is *todo*. But before a release all todo's should be resolved (except there is a good and documented reason to keep the marker). The static code analyser can be used to provide a list of all positions within your code, at which still such a TODO marker is present.

Hint: The console displays a warning regarding rule '0906'. This warning can be ignored.

2. Library

Document for the library

3. Apertis Pro

3.1 How to use DLT

3.1.1 dlt-daemon on Apertis

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

```
systemctl status dlt-daemon
```

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

- Install **dlt-daemon** package

```
sudo apt install dlt-daemon
```

- Start **dlt-daemon** service

```
sudo systemctl start dlt-daemon
```

3.1.2 Apertis Pro firewall configuration

In order to capture DLT log/trace from DLT client(**DLT Viewer**, **DLTConnector**), DLT client has to communicate with Apertis Pro (TCP/IP protocol) via port **3490** (as default).

So that, this connection should be allowed on Apertis Pro target.

Adopt settings of firewall at Apertis Pro:

- Add new rule to allow DLT service at port **3490** (as default)

Edit `/etc/iptables/rules.v4` file to add below line

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

- Restart the firewall with changed parameters

```
sudo systemctl restart iptables.service
```

3.1.3 DLTSelfTestApp

DLTSelfTestApp is an application which will be run on the Apertis Pro target for testing the DLT connection between Robotframework AIO and target.

This package is a part of Robotframework Selftest helpers.

To install **DLTSelfTestApp**, download its debian package on Apertis Pro target then execute the below command.

```
sudo dpkg -i <path/to/dltselftestapp_1.0.0_amd64.deb>
```

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

```
/opt/bosch/rob主fw/dlt/DLTSelfTestApp
```

Welcome log message `Welcome to Robotframework AIO DLTSelfTestApp...` will be sent at application startup.

Then the ping log `ping message from Robotframework AIO DLTSelfTestApp` every 5 seconds.

DLT command injection:

To perform the DLT command injection, use below information:

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

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

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

3.1.4 QConnectDLTLibrary

`QConnectDLTLibrary` is part of Robotframework AIO.

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

Sample Robotframework testcase which are using `QConnectDLTLibrary` to test `DLTSelfTestApp` on Apertis Pro target:

- Robot `Settings` (Setup, Teardown) and used `Variables` , `Keyword`

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

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

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

Open Connection
    ${dlt_config} =    evaluate    json.loads('"'${DLT_CONNECTION_CONFIG}"')    json

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

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

- Sample robot testcase to verify the ping message from `DLTSelfTestApp`

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

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


- Sample robot testcase to verify command injection with DLTSelfTestApp

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

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

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

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

Please refer [QConnectDLTLibrary repository](#) for more details about usage and other example testcase for DLT connection.

4. Library documentation

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

Overview about the included libraries:

GenPackageDoc
Version 0.20.2 (from 05.07.2022)
https://github.com/test-fullautomation/python-genpackagedoc
<i>Documentation builder for Python packages</i>

PythonExtensionsCollection
Version 0.8.3 (from 08.07.2022)
https://github.com/test-fullautomation/python-extensions-collection
<i>Additional Python functions</i>

RobotframeworkExtensions
Version 0.7.3 (from 05.07.2022)
https://github.com/test-fullautomation/robotframework-extensions-collection
<i>Additional Robot Framework keywords</i>

JsonPreprocessor
Version 0.1.3 (from 05.07.2022)
https://github.com/test-fullautomation/python-jsonpreprocessor
<i>Preprocessor for json files</i>

RobotFramework_Testsuites
Version 0.1.2 (from 05.07.2022)
https://github.com/test-fullautomation/robotframework-testsuitesmanagement
<i>Functionality to manage RobotFramework testsuites</i>

QConnectBase
Version 1.1.0 (from 05.07.2022)
https://github.com/test-fullautomation/robotframework-qconnect-base
<i>Robot Framework test library for TCP, SSH, serial connection</i>

RobotResults2RQM
Version 1.1.1 (from 05.07.2022)
https://github.com/test-fullautomation/robotframework-testresult2rqmtool
<i>Imports robot result(s) to IBM Rational Quality Manager (RQM)</i>

RobotResults2DB
Version 1.2.1 (from 05.07.2022)
https://github.com/test-fullautomation/robotframework-testresultwebapptool
<i>Imports robot result(s) to TestResultWebApp database</i>

TestResultWebApp
Version 0.1.0 (from 04.07.2022)
https://github.com/test-fullautomation/testresultwebapp
<i>Web based display of test results</i>

4.1 GenPackageDoc

GenPackageDoc

v. 0.20.2

Holger Queckenstedt

05.07.2022

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	Syntax extensions	9
2.7.2	Examples	9
3	CDocBuilder.py	10
3.1	Class: CDocBuilder	10
3.1.1	Method: Build	10
4	CInterface.py	11
4.1	Class: CInterface	11
4.1.1	Method: GetLaTeXStyles	11
5	CPackageDocConfig.py	12
5.1	Class: CPackageDocConfig	12
5.1.1	Method: PrintConfig	12
5.1.2	Method: PrintConfigKeys	12
5.1.3	Method: Get	12
5.1.4	Method: GetConfig	12
6	CPatterns.py	13
6.1	Class: CPatterns	13
6.1.1	Method: GetHeader	13
6.1.2	Method: GetChapter	14
6.1.3	Method: GetFooter	14

CONTENTS	B
7 CSourceParser.py	15
7.1 Class: CSourceParser	15
7.1.1 Method: ParseSourceFile	15
8 Outlook	17
9 Appendix	20
10 History	21

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.

It is possible to extend the documentation by the content of additional text files. The docstrings and also the additional text files have to be written in rst syntax (rst is the abbreviation for "re structured text", that is a certain markdown dialect).

The documentation is generated in two steps:

1. The rst sources are converted into LaTeX sources
2. The LaTeX sources are converted into a PDF document. This requires a separately installed LaTeX distribution (recommended: MiKTeX), that is **not** part of `GenPackageDoc`.

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

[python-genpackagedoc](#)

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

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

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

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

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

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

Chapter 2

Description

2.1 Repository content

What is the content of the repository python-genpackagedoc?

- Folder GenPackageDoc

Contains the package code.

This folder is specific for the package.

- Folder config

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

This folder is specific for the repository.

- Folder additions

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

- Folder packagedoc

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

This folder is specific for the documentation.

- Repository root folder

- genpackagedoc.py

- Python script to start the documentation build

- setup.py

- Python script to install the package sources. This includes the execution of genpackagedoc.py. Therefore building the documentation is part of the installation process.

- dump_repository_config.py

- Little helper to dump the repository configuration to console

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

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

```
"OUTPUT" : "../build"
```

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

Further details are explained within the json file itself.

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

GenPackageDoc/CPackageDocConfig.py

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

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

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

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

Command line

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

--output

Path and name of folder containing all output files.

--pdfdest

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

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

--configdest

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

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

--strict

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

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	4
2.1	Repository content	4
2.2	Documentation build process	5
2.3	PDF document structure	6
2.4	Examples	7
2.4.1	Example 1: rst file	7
2.4.2	Example 2: Python module	7

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	8
3.1	Class: CDocBuilder	8
3.1.1	Method: Build	8

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",
```

2.7 Syntax aspects

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 it is recommended to start the docstring text within the first column (or rather use the first column as reference for further indentations of rst text).
- In rst also blank lines are part of the syntax!

Please be attentive while typing your documentation in rst format!

2.7.1 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.

2.7.2 Examples

(to be continued)

Chapter 3

CDocBuilder.py

Python module containing all methods to generate tex sources.

3.1 Class: CDocBuilder

```
GenPackageDoc.CDocBuilder
```

Main class to build tex sources out of docstrings of Python modules and separate text files in rst format. Depends on a json configuration file, provided by a `oPackageDocConfig` object (this includes the Repository configuration).

Method to execute: `Build()`

3.1.1 Method: Build

Arguments:

(no arguments)

Returns:

- `bSuccess`

/ *Type*: bool /

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

- `sResult`

/ *Type*: str /

The result of the computation of the method `sMethod`.

Chapter 4

CInterface.py

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

4.1 Class: CInterface

```
GenPackageDoc.CInterface
```

4.1.1 Method: GetLaTeXStyles

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

Arguments:

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

Returns:

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

Chapter 5

CPackageDocConfig.py

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

5.1 Class: CPackageDocConfig

```
GenPackageDoc.CPackageDocConfig
```

5.1.1 Method: PrintConfig

Prints all configuration values to console.

5.1.2 Method: PrintConfigKeys

Prints all configuration key names to console.

5.1.3 Method: Get

Returns the configuration value belonging to a key name.

5.1.4 Method: GetConfig

Returns the complete configuration dictionary.

Chapter 6

CPatterns.py

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

6.1 Class: CPatterns

```
GenPackageDoc.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).

- `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.

Chapter 7

CSourceParser.py

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

7.1 Class: CSourceParser

```
GenPackageDoc.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`
/ *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

Outlook

ToDo list:

- [01]
Introduce `setup.py` including the execution of `genpackagedoc.py` and adding the generated PDF document to the installation.
Introduce `README.rst` and `README.md`.
10.05.2022: Setup process introduced and `README.rst` added
- [02]
Currently it is hard coded, that private functions and methods are skipped. Therefore they are not part of the resulting PDF document.
A configuration switch might be useful to give the user the ability to control this behavior.
09.05.2022: Parameter `'INCLUDEPRIVATE'` added
- [03]
Currently it is implemented that also functions, classes and methods without docstrings are part of the resulting PDF document. They are listed together with the hint, that a docstring is not available.
A configuration switch might be useful to give the user the ability to control this behavior.
09.05.2022: Parameter `'INCLUDEUNDOCUMENTED'` added
- [04]
Currently it is implemented that for Python modules will be searched recursively within the given root folder. Maybe the algorithm also catches modules from which the user does not want `GenPackageDoc` to include them.
A configuration exclude filter can be implemented to skip those files. Or maybe other way round: An include filter includes a subset of available files only.
The same filter mechanism can be extended for the content of Python modules (= include/exclude functions, classes and methods).
- [05]
Currently the configuration parameter for the documentation build process are taken from a json file `packagedoc.config.json`.
It might be helpful to have the possibility to overwrite them in command line (e.g. for redirecting the path to the output folder without changing any code).
31.05.2022: Implemented in v. 0.15.0
- [06]
Introduce text boxes for warnings, errors and informations.
19.05.2022: implemented in v. 0.12.0

- [07]

The documentation build process allows relative paths only (in `packagedoc_config.json`).

Maybe a mechanism is useful to allow absolute paths and paths based on environment variables also.

01.06.2022: implemented in v. 0.16.0
- [08]

Explore further rst syntax elements like the code directive. Some of them produces LaTeX code that requires the include of additional LaTeX packages. Sometimes this causes errors that have to be fixed.

05.05.2022: Python syntax highlighting realized
- [09]

The documentation has to be extended by a set of rst examples (rst best practices).
- [10]

A postprocessing for LaTeX code needs to be implemented:

 - Enable proper line breaks
 - Resolve the ambiguity of labels created automatically when the LaTeX code is generated (for every input file separately)

10.05.2022: Experimental syntax extensions for `newline`, `newpage` and `vspace`

17.05.2022: Postprocessing for rst and tex sources added; "multiply-defined labels" fix.
- [11]

Currently the docstrings of Python modules have to contain a heading for functions, classes and methods. The developer is responsible for that. Maybe it is not necessary to maintain these headings manually. It has to be investigated, if these headings can be added automatically by `GenPackageDoc`.

06.05.2022: Headings are added automatically.
- [12]

Currently the documentation of a single Python module starts at *function* or *class* level. This means it is not possible to provide common information about the Python module itself (placed **before** the content of the first function or class of the module). A way have to be found to add such content.

06.05.2022: Implemented in version 0.4.0
- [13]

The error handling needs to be extended!

17.06.2022: Implemented in version 0.17.0
- [14]

Take over the description of

```
config/repository-config.json
```

from inside this json file (comment blocks) to the main PDF document.
- [15]

Reference section with useful links
- [16]

History

10.05.2022: History added

- [17]
Debug switch to enable additional output
- [18]
Parse decorators to identify Robot Framework keyword definitions
- [19]
Selftest
- [20]
Introduce a separate folder containing TeX styles - instead of having them hard coded within CPatterns.py.
24.05.2022: implemented in v. 0.13.0

Chapter 9

Appendix

About this package:

Table 9.1: Package setup

Setup parameter	Value
Name	GenPackageDoc
Version	0.20.2
Date	05.07.2022
Description	Documentation builder for Python packages
Package URL	python-genpackagedoc
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 3 - Alpha
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 10

History

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 INCLUDEPRIVATE added</i>	
0.6.0	09.05.2022
<i>Parameter INCLUDEUNDOCUMENTED added</i>	
0.7.0	10.05.2022
<i>Setup process introduced and README.rst 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 newline, newpage and vspace 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 rst and tex sources added; 'multiply-defined labels' fix.</i>	
0.11.0	18.05.2022
<i>Import of tex files enabled</i>	

0.12.0	19.05.2022
<i>Admonitions added, based on LaTeX environment 'tcolorbox'; layout adaptions in titlepage; page numbering fix in TOC</i>	
0.13.0	24.05.2022
<i>LaTeX style definitions moved to separate folder</i>	
0.14.0	27.05.2022
<i>LaTeX compiler check added; control parameter STRICT added to packagedoc.config</i>	
0.15.0	31.05.2022
<i>Command line added; 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; code maintenance; error handling</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; 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>	

GenPackageDoc.pdf

Created at 11.07.2022 - 14:44:45

by GenPackageDoc v. 0.20.2

4.2 PythonExtensionsCollection

PythonExtensionsCollection

v. 0.8.3

Holger Queckenstedt

08.07.2022

Contents

1	Introduction	1
2	Description	2
2.1	Path normalization	2
2.2	File access with CFile	2
3	CFile.py	8
3.1	Class: enFileStatType	8
3.2	Class: CFile	8
3.2.1	Method: Close	9
3.2.2	Method: Delete	9
3.2.3	Method: Write	9
3.2.4	Method: Append	10
3.2.5	Method: ReadLines	11
3.2.6	Method: GetFileInfo	12
3.2.7	Method: CopyTo	13
3.2.8	Method: MoveTo	14
4	CFolder.py	15
4.1	Function: rm_dir_readonly	15
4.2	Class: CFolder	15
4.2.1	Method: Delete	15
4.2.2	Method: Create	16
4.2.3	Method: CopyTo	16
5	CString.py	18
5.1	Class: CString	18
5.1.1	Method: NormalizePath	18
5.1.2	Method: DetectParentPath	19
5.1.3	Method: StringFilter	20
5.1.4	Method: FormatResult	26
6	CUtills.py	28
6.1	Function: PrettyPrint	28
6.2	Class: CTypePrint	29
6.2.1	Method: TypePrint	30

CONTENTS	B
7 Appendix	31
8 History	32

Chapter 1

Introduction

The Python Extensions Collection package extends the functionality of Python by some useful functions that are not available in Python immediately.

This covers for example string operations like normalizing a path or searching for parent directories within a path.

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

Module import

The modules of the Python Extension Collection and their methods can be accessed in the following ways:

CFile

```
from PythonExtensionsCollection.File.CFile import CFile
...
sFile = r"%TMP%\File.txt"
oFile = CFile(sFile)
```

Please consider that oFile is an instance of the class CFile - *and not a file handle*.

CString

```
from PythonExtensionsCollection.String.CString import CString
...
sPath = CString.NormalizePath(sPath)
...
bAck = CString.StringFilter(sString, ...)
...
sResult = CString.FormatResult(sMethod="", bSuccess=True, sResult="")
```

CUtils

```
from PythonExtensionsCollection.Utills.CUtills import *
...
PrettyPrint(oData)
```

Chapter 2

Description

2.1 Path normalization

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 function `CString::NormalizePath()` provides lot's of parameters to influence the result.

2.2 File access with CFile

The motivation for the CFile module contains two main topics:

1. More 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.

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 returns at least `bSuccess` and `sResult`.

- `bSuccess` is `True` in case of no error occurred.
- `bSuccess` is `False` in case of an error occurred.
- `bSuccess` is `None` in case of a very fatal error occurred (exceptions).
- `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 the `Write()` after the `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 a single line from file only 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.

Let's assume the following:

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

This can be implemented in the following way.

Read a subset of file content:

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

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

Close a file handle:

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

This makes sense in case of later again access to this file is needed.

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.

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.

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 it's own file.

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

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

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

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

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

bSuccess, sResult = oFile_1.CopyTo(sFile_2)

del oFile_1
del oFile_2
```

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

In the following example the copying is successful:

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

oFile_2 = CFile(sFile_2)
```

```
listLines_2, bSuccess, sResult = oFile_2.ReadLines()
del oFile_2

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

Chapter 3

CFile.py

3.1 Class: enFileStatiType

```
PythonExtensionsCollection.File.CFile
```

The class `enFileStatiType` defines the following file states:

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

3.2 Class: CFile

```
PythonExtensionsCollection.File.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.

3.2.1 Method: Close

Closes the opened file.

Arguments:

(no args)

Returns:

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

3.2.2 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.

3.2.3 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.

3.2.4 Method: Append

Appends the content of a variable Content to file.

Arguments:

- `Content`
/ *Condition*: required / *Type*: one of: str, list, tuple, set, dict, dict /
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.

3.2.5 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.

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

Arguments:

- `bCaseSensitive`
/ Condition: optional / Type: bool / Default: True /
 - If `True`, the standard filters work case sensitive, otherwise not.
 - This has no effect to the regular expression based filters `sInclRegEx` and `sExclRegEx`.
- `bSkipBlankLines`
/ Condition: optional / Type: bool / Default: False /
 If `True`, blank lines will be skipped, otherwise not.
- `sComment`
/ Condition: optional / Type: str / Default: None /
 In case of a line starts with the string `sComment`, this line is skipped.
- `sStartsWith`
/ Condition: optional / Type: str / Default: None /
 - The criterion of this filter is fulfilled in case of the input string starts with the string `sStartsWith`
 - More than one string can be provided (semicolon separated; logical join: `OR`)
- `sEndsWith`
/ Condition: optional / Type: str / Default: None /
 - The criterion of this filter is fulfilled in case of the input string ends with the string `sEndsWith`
 - More than one string can be provided (semicolon separated; logical join: `OR`)
- `sStartsNotWith`
/ Condition: optional / Type: str / Default: None /
 - The criterion of this filter is fulfilled in case of the input string starts not with the string `sStartsNotWith`
 - More than one string can be provided (semicolon separated; logical join: `AND`)
- `sEndsNotWith`
/ Condition: optional / Type: str / Default: None /
 - The criterion of this filter is fulfilled in case of the input string ends not with the string `sEndsNotWith`
 - More than one string can be provided (semicolon separated; logical join: `AND`)
- `sContains`
/ Condition: optional / Type: str / Default: None /
 - The criterion of this filter is fulfilled in case of the input string contains the string `sContains` at any position

- More than one string can be provided (semicolon separated; logical join: OR)
- `sContainsNot`
/ *Condition*: optional / *Type*: str / *Default*: None /
 - The criterion of this filter is fulfilled in case of the input string does **not** contain the string `sContainsNot` at any position
 - More than one string can be provided (semicolon separated; logical join: AND)
- `sInclRegEx`
/ *Condition*: optional / *Type*: str / *Default*: None /
 - *Include* filter based on regular expressions (consider the syntax of regular expressions!)
 - The criterion of this filter is fulfilled in case of the regular expression `sInclRegEx` matches the input string
 - Leading and trailing blanks within the input string are considered
 - `bCaseSensitive` has no effect
 - A semicolon separated list of several regular expressions is **not** supported
- `sExclRegEx`
/ *Condition*: optional / *Type*: str / *Default*: None /
 - *Exclude* filter based on regular expressions (consider the syntax of regular expressions!)
 - The criterion of this filter is fulfilled in case of the regular expression `sExclRegEx` does **not** match the input string
 - Leading and trailing blanks within the input string are considered
 - `bCaseSensitive` has no effect
 - A semicolon separated list of several regular expressions is **not** supported
- `bLStrip`
/ *Condition*: optional / *Type*: bool / *Default*: False /
If `True`, leading spaces are removed from line before the filters are used, otherwise not.
- `bRStrip`
/ *Condition*: optional / *Type*: bool / *Default*: True /
If `True`, trailing spaces are removed from line before the filters are used, otherwise not.
- `bToScreen`
/ *Condition*: optional / *Type*: bool / *Default*: False /
If `True`, the content read from file is also printed to screen, otherwise not.

3.2.6 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

3.2.7 Method: CopyTo

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

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

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

Arguments:

- `sDestination`
/ *Condition*: required / *Type*: string /
The path to destination file (either incl. file name or without file name)
- `bOverwrite`
/ *Condition*: optional / *Type*: bool / *Default*: False /
 - In case of the destination file already exists and `bOverwrite` is `True`, than the destination file will be overwritten.
 - In case of the destination file already exists and `bOverwrite` is `False` (default), than the destination file will not be overwritten and `CopyTo` returns `bSuccess = False`.

Returns:

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

3.2.8 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 4

CFolder.py

4.1 Function: `rm_dir_readonly`

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

4.2 Class: `CFolder`

```
PythonExtensionsCollection.Folder.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.

4.2.1 Method: `Delete`

Deletes the folder the current class instance contains.

Arguments:

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

Returns:

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

4.2.2 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.

4.2.3 Method: CopyTo

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

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

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

Arguments:

- `sDestination`
/ *Condition*: required / *Type*: string /
The path to destination folder
- `bOverwrite`
/ *Condition*: optional / *Type*: bool / *Default*: False /
 - In case of the destination folder already exists and `bOverwrite` is `True`, than the destination folder will be overwritten.
 - In case of the destination folder already exists and `bOverwrite` is `False` (default), than the destination folder will not be overwritten and `CopyTo` returns `bSuccess = False`.

Returns:

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

Chapter 5

CString.py

5.1 Class: CString

```
PythonExtensionsCollection.String.CString
```

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

5.1.1 Method: NormalizePath

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

Arguments:

- `sPath`
/ Condition: required / Type: str /
 The path to be normalized
- `bWin`
/ Condition: optional / Type: bool / Default: False /
 If `True` then returned path contains masked backslashes as separator, otherwise slashes
- `sReferencePathAbs`
/ Condition: optional / Type: str / Default: None /
 In case of `sPath` is relative and `sReferencePathAbs` (expected to be absolute) is given, then the returned absolute path is a join of both input paths
- `bConsiderBlanks`
/ Condition: optional / Type: bool / Default: False /
 If `True` then the returned path is encapsulated in quotes - in case of the path contains blanks
- `bExpandEnvVars`
/ Condition: optional / Type: bool / Default: True /
 If `True` then in the returned path environment variables are resolved, otherwise not.
- `bMask`
/ Condition: optional / Type: bool / Default: True (requires bWin=True) /
 - If `bWin` is `True` and `bMask` is `True` then the returned path contains masked backslashes as separator.

- If `bWin` is `True` and `bMask` is `False` then the returned path contains single backslashes only
 - this might be required for applications, that are not able to handle masked backslashes.
- In case of `bWin` is `False` `bMask` has no effect.

Returns:

- `sPath`
/ Type: str /
 The normalized path (is `None` in case of `sPath` is `None`)

5.1.2 Method: DetectParentPath

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

Arguments:

- `sStartPath`
/ Condition: required / Type: str /
 The path in which to search for a parent folder
- `sFolderName`
/ Condition: required / Type: str /
 The name of the folder to search for within `sStartPath`. It is possible to provide more than one folder name separated by semicolon
- `sFileName`
/ Condition: optional / Type: str / Default: None /
 The name of a file to search within the detected parent folder

Returns:

- `sDestPath`
/ Type: str /
 Path and name of parent folder found inside `sStartPath`, `None` in case of `sFolderName` is not found inside `sStartPath`. In case of more than one parent folder is found `sDestPath` contains the first result and `listDestPaths` contains all results.
- `listDestPaths`
/ Type: list /
 If `sFolderName` contains a single folder name this list contains only one element that is `sDestPath`. In case of `FolderName` contains a semicolon separated list of several folder names this list contains all found paths of the given folder names. `listDestPaths` is `None` (and not an empty list!) in case of `sFolderName` is not found inside `sStartPath`.
- `sDestFile`
/ Type: str /
 Path and name of `sFileName`, in case of `sFileName` is given and found inside `listDestPaths`. In case of more than one file is found `sDestFile` contains the first result and `listDestFiles` contains all results. `sDestFile` is `None` in case of `sFileName` is `None` and also in case of `sFileName` is not found inside `listDestPaths` (and therefore also in case of `sFolderName` is not found inside `sStartPath`).

- `listDestFiles`
/ *Type*: list /
Contains all positions of `sFileName` found inside `listDestPaths`.
`listDestFiles` is `None` (and not an empty list!) in case of `sFileName` is `None` and also in case of `sFileName` is not found inside `listDestPaths` (and therefore also in case of `sFolderName` is not found inside `sStartPath`).
- `sDestPathParent`
/ *Type*: str /
The parent folder of `sDestPath`, `None` in case of `sFolderName` is not found inside `sStartPath` (`sDestPath` is `None`).

5.1.3 Method: 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.

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 **not** 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

The following filters are available:

bSkipBlankStrings

- Like already mentioned above 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)

sEndsWith

- The criterion of this filter is fulfilled in case of the input string does **not** end 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: AND)

sContains

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

sContainsNot

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

sInclRegEx

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

sExclRegEx

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

Further arguments:

- `sString`
/ *Condition*: required / *Type*: str /
The input string that has to be investigated.

- `bCaseSensitive`

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

If True, the standard filters work case sensitive, otherwise not.

- `bDebug`

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

If True, additional output is printed to console (e.g. the decision of every single filter), otherwise not.

Returns:

- `bAck`

/ *Type*: bool /

Final statement about the input string `sString` after filter computation

Examples:

1. Returns True:

```
StringFilter(sString      = "Speed is 25 beats per minute",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = None,
             sStartsWith   = "Sp",
             sEndsWith     = None,
             sStartsWithNot = None,
             sEndsWithNot  = None,
             sContains     = "beats",
             sContainsNot  = None,
             sInclRegex    = None,
             sExclRegex    = None)
```

2. Returns False:

```
StringFilter(sString      = "Speed is 25 beats per minute",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = None,
             sStartsWith   = "Sp",
             sEndsWith     = None,
             sStartsWithNot = None,
             sEndsWithNot  = "minute",
             sContains     = "beats",
             sContainsNot  = None,
             sInclRegex    = None,
             sExclRegex    = None)
```

3. Returns True:

```
StringFilter(sString      = "Speed is 25 beats per minute",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = None,
```

```

sStartsWith      = None,
sEndsWith        = None,
sStartsWithNot   = None,
sEndsWithNot     = None,
sContains        = None,
sContainsNot     = "Beats",
sInclRegEx       = None,
sExclRegEx       = None)

```

4. Returns True:

```

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

```

5. Returns False:

```

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

```

6. Returns False:

```

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

```

7. Returns False:

```
StringFilter(sString      = "      ",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = None,
             sStartsWith   = None,
             sEndsWith     = None,
             sStartsWithNot = None,
             sEndsWithNot  = None,
             sContains     = None,
             sContainsNot  = None,
             sInclRegex    = None,
             sExclRegex    = None)
```

8. Returns False:

```
StringFilter(sString      = "# Speed is 25 beats per minute",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = "#",
             sStartsWith   = None,
             sEndsWith     = None,
             sStartsWithNot = None,
             sEndsWithNot  = None,
             sContains     = "beats",
             sContainsNot  = None,
             sInclRegex    = None,
             sExclRegex    = None)
```

9. Returns False:

```
StringFilter(sString      = "    Alpha is not beta; and beta is not
↪ gamma ",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = None,
             sStartsWith   = None,
             sEndsWith     = None,
             sStartsWithNot = None,
             sEndsWithNot  = None,
             sContains     = "    Alpha ",
             sContainsNot  = None,
             sInclRegex    = None,
             sExclRegex    = None)
```

Because blanks around search strings (here " Alpha ") are considered, whereas the blanks around the input string are removed before computation. Therefore " Alpha " cannot be found within the (shortened) input string.

10. This alternative solution returns True:

```
StringFilter(sString      = "    Alpha is not beta; and beta is not
↪ gamma ",
             bCaseSensitive = True,
             bSkipBlankStrings = True,
             sComment      = None,
```

```

sStartsWith      = None,
sEndsWith        = None,
sStartsWithNot   = None,
sEndsWithNot     = None,
sContains        = None,
sContainsNot     = None,
sInclRegEx       = r"\s{3}Alpha",
sExclRegEx       = None)

```

11. Returns True:

```

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

```

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

A semicolon that shall be part of the search string, has to be masked with '\;'

The meaning of "beta\; not" in the following example is: The criterion is fulfilled in case of "beta; not" can be found.

That's **not** True. Therefore the method returns False:

```

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

```

5.1.4 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 6

CUtils.py

6.1 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

Example call:

```
PrettyPrint(oData)
```

(with oData is a Python variable of any type)

The output can e.g. look like this:

```
[DICT] (3/1) > {K1} [STR] : 'Val1'
[DICT] (3/2) > {K2} [LIST] (4/1) > [INT] : 1
[DICT] (3/2) > {K2} [LIST] (4/2) > [STR] : 'A'
[DICT] (3/2) > {K2} [LIST] (4/3) > [INT] : 2
[DICT] (3/2) > {K2} [LIST] (4/4) > [TUPLE] (2/1) > [INT] : 9
[DICT] (3/2) > {K2} [LIST] (4/4) > [TUPLE] (2/2) > [STR] : 'Z'
[DICT] (3/3) > {K3} [INT] : 5
```

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

For example the meaning of the fifth line of output

```
[DICT] (3/2) > {K2} [LIST] (4/4) > [TUPLE] (2/1) > [INT] : 9
```

is:

- The type of input parameter (oData) is dict

- The dictionary contains 3 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 9

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

Arguments:

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

Returns:

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

6.2 Class: CTypePrint

```
PythonExtensionsCollection.Utills.CUtils
```

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.

6.2.1 Method: `TypePrint`

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

Arguments:

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

Returns:

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

Chapter 7

Appendix

About this package:

Table 7.1: Package setup

Setup parameter	Value
Name	PythonExtensionsCollection
Version	0.8.3
Date	08.07.2022
Description	Additional Python functions
Package URL	python-extensions-collection
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 8

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
<i>Documentation rebuild with GenPackageDoc v. 0.13.0; code maintenance</i>	
0.5.0	31.05.2022
<i>Adapted to GenPackageDoc v. 0.15.0</i>	
0.6.0	02.06.2022
<i>Documentation rebuild with GenPackageDoc v. 0.16.0; code maintenance</i>	
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>	

PythonExtensionsCollection.pdf

Created at 11.07.2022 - 14:44:54

by GenPackageDoc v. 0.20.2

4.3 RobotframeworkExtensions

RobotframeworkExtensions

v. 0.7.3

Holger Queckenstedt

05.07.2022

Contents

1	Introduction	1
2	Collection.py	2
2.1	Class: <code>Collection</code>	2
2.1.1	Method: <code>pretty_print</code>	2
2.1.2	Method: <code>normalize_path</code>	3
3	Appendix	6
4	History	7

Chapter 1

Introduction

The *Robotframework Extensions Collection* extends the functionality of the Robotframework by some keywords providing features, that are implemented in the *Python Extensions Collection*.

The goal behind these extensions is to have certain functionality available in both: pure Python applications and Robotframework.

The *Robotframework Extensions Collection* requires an installed *Python Extensions Collection*, that can be found in this repository:

[python-extensions-collection](#)

Chapter 2

Collection.py

The Collection module is the interface between the Python Extensions Collection and the Robotframework. This library containing the keyword definitions, can be imported in the following way:

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

2.1 Class: Collection

```
RobotframeworkExtensions.Collection
```

Module main class

2.1.1 Method: 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 logging.

The output contains for every parameter: the value, the type and counter values (in case of composite data types).

The trace level for output is `INFO`.

The output is also returned as list of strings.

Arguments:

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

Returns:

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

Example:

Variable of Python type `list`:

```

set_test_variable    @{aItems}    String
...                  ${25}
...                  ${True}
...                  ${None}

```

Call of `pretty_print` keyword:

```

rf.extensions.pretty_print    ${aItems}

```

Output:

```

INFO - [LIST] (4/1) > [STR] : 'String'
INFO - [LIST] (4/2) > [INT] : 25
INFO - [LIST] (4/3) > [BOOL] : True
INFO - [LIST] (4/4) > [NONE] : None

```

2.1.2 Method: `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)

Example 1:

Variable containing a path with:

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

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

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

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

Usage of the normalize_path keyword:

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

Result (content of sPath):

```
C:/subfolder3
```

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

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

Result (content of sPath):

```
C:\\subfolder3
```

The masking of backslashes can be deactivated:

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

Result (content of sPath):

```
C:\subfolder3
```

Example 2:

Variable containing a path of a local network resource:

```
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

Appendix

About this package:

Table 3.1: Package setup

Setup parameter	Value
Name	RobotframeworkExtensions
Version	0.7.3
Date	05.07.2022
Description	Additional Robot Framework keywords
Package URL	robotframework-extensions-collection
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 4

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
<i>Documentation rebuild with GenPackageDoc v. 0.13.0; code maintenance</i>	
0.5.0	31.05.2022
<i>Adapted to GenPackageDoc v. 0.15.0</i>	
0.6.0	02.06.2022
<i>Documentation rebuild with GenPackageDoc v. 0.16.0; code maintenance</i>	
0.7.0	28.06.2022
<i>PythonExtensionsCollection updated to version 0.8.0</i>	

RobotframeworkExtensions.pdf

Created at 11.07.2022 - 14:45:13

by GenPackageDoc v. 0.20.2

4.4 JsonPreprocessor

JsonPreprocessor

v. 0.1.3

Mai Dinh Nam Son

05.07.2022

Contents

1	Introduction	1
2	Description	2
3	CJsonPreprocessor.py	3
3.1	Class: CSyntaxType	3
3.2	Class: CPythonJSONDecoder	3
3.2.1	Method: custom_scan_once	3
3.3	Class: CJsonPreprocessor	3
3.3.1	Method: jsonLoad	4
4	Appendix	5
5	History	6

Chapter 1

Introduction

Json Preprocessor

!!! add introduction - if wanted !!

Chapter 2

Description

The json preprocessor:



TO BE CONTINUED

Chapter 3

CJsonPreprocessor.py

3.1 Class: CSyntaxType

```
JsonPreprocessor.CJsonPreprocessor
```

3.2 Class: CPythonJSONDecoder

```
JsonPreprocessor.CJsonPreprocessor
```

Method: PythonJSONDecoder Add python data types and syntax to json. True, False and None will be accepted as json syntax elements.

Args: json.JSONDecoder (*object*) Decoder object provided by `json.loads`

3.2.1 Method: custom_scan_once

3.3 Class: CJsonPreprocessor

```
JsonPreprocessor.CJsonPreprocessor
```

CJsonPreprocessor extends the syntax of json.

Features are

- Allow c/c++-style comments within json files.
// single line or part of single line and /* */ multiline comments are possible
- Allow to import json files into json files
"[import]" : "relative/absolute path", imports another json file to exactly this location.
%envariable% and \${envariable} can be used, too.
- Allow use of variables within json files
In any place the syntax \${basenode.subnode. ... nodename} allows to reference an already existing variable.
Example:

```
{
  "basenode" : {
    subnode : {
      "myparam" : 5
    },
  },
  "myVar" : "${basenode.subnode.myparam}"
}
```

- Allow python data types True, False and None

3.3.1 Method: jsonLoad

Method: jsonLoad This function is the entry point of JsonPreprocessor.

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

Args: jFile (*string*) relative/absolute path to main json file.

%envvariable% and \${envvariable} can be used, too in order to access environment variables.

Returns: oJson (*dict*) preprocessed json file(s) as data structure

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	JsonPreprocessor
Version	0.1.3
Date	05.07.2022
Description	Preprocessor for json files
Package URL	python-jsonpreprocessor
Author	Mai Dinh Nam Son
Email	son.maidinhnam@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 5

History

0.1.0	01/2022
<i>Initial version</i>	

JsonPreprocessor.pdf

Created at 11.07.2022 - 14:45:21

by GenPackageDoc v. 0.20.2

4.5 RobotFramework_Testsuites

RobotFramework_Testsuites

v. 0.1.2

Mai Dinh Nam Son

05.07.2022

Contents

1	Introduction	1
2	Description	2
3	CConfig.py	3
3.1	Class: dotdict	3
3.2	Class: CConfig	3
3.2.1	Method: loadCfg	3
3.2.2	Method: updateCfg	3
3.2.3	Method: verifyRbfwVersion	3
3.2.4	Method: versioncontrol_error	4
4	COnFailureHandle.py	5
4.1	Class: COnFailureHandle	5
4.1.1	Method: is_noney	5
4.1.2	Method: register_keyword_run_on_failure	5
5	CSetup.py	6
5.1	Class: CSetupKeywords	6
5.1.1	Method: testsuite_teardown	6
5.1.2	Method: testcase_setup	6
5.1.3	Method: testcase_teardown	6
5.1.4	Method: update_config	6
5.2	Class: CGeneralKeywords	6
6	CStruct.py	7
6.1	Class: CStruct	7
7	Event.py	8
7.1	Class: Event	8
7.1.1	Method: trigger	8
8	ScopeEvent.py	9
8.1	Class: ScopeEvent	9
8.1.1	Method: trigger	9
8.2	Class: ScopeStart	9

CONTENTS	B
8.3 Class: ScopeEnd	9
9 __init__.py	10
9.1 Function: on	10
9.2 Function: dispatch	10
9.3 Function: register_event	10
10 LibListener.py	11
10.1 Class: LibListener	11
11 __init__.py	12
11.1 Class: RobotFramework.Testsuites	12
11.1.1 Method: get_keyword_tags	12
11.1.2 Method: get_keyword_documentation	12
11.1.3 Method: failure_occurred	12
11.2 Class: CTestsuitesCfg	12
12 version.py	13
12.1 Function: robfwaio_version	13
13 Appendix	14
14 History	15

Chapter 1

Introduction

RobotFramework.Testsuites

!!! add introduction - if wanted !!

Chapter 2

Description

The RobotFramework AIO testsuites management:
TO BE CONTINUED

Chapter 3

CConfig.py

3.1 Class: dotdict

```
RobotFramework.Testsuites.Config.CConfig
```

3.2 Class: CConfig

```
RobotFramework.Testsuites.Config.CConfig
```

Defines the properties of configuration Holds the identified config files. Level1 is highest priority, Level4 is lowest priority.

(remaining content needs to be fixed and restored)

3.2.1 Method: loadCfg

3.2.2 Method: updateCfg

staticmethod updateParams: This method updates preprocessor, global or local params base on ROBFW local config or any json config file according to purpose of specific testsuite.

Args: sUpdateCfgFile: str

Returns: None

3.2.3 Method: verifyRbfwVersion

Validate the current robotframework version with maximum and minimum version (if provided in the configuration file). In case the current version is not between min and max version, then the execution of testsuite is terminated with "unknown" state Method: bValidateMinVersion -----

Validate current version with required minimum version. Method: bValidateMaxVersion -----

Validate current version with required maximum version. Method: bValidateSubVersion -----

Validate the format of provided sub version and parse it into sub tuple for version comparison. Method: tupleVersion -----

Return a tuple which contains the (major, minor, patch) version.

(remaining content needs to be fixed and restored)

3.2.4 Method: `versioncontrol_error`

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

Chapter 4

COnFailureHandle.py

4.1 Class: COnFailureHandle

```
RobotFramework.Testsuites.Keywords.COnFailureHandle
```

4.1.1 Method: is_noney

4.1.2 Method: register_keyword_run_on_failure

TBD

Chapter 5

CSetup.py

5.1 Class: CSetupKeywords

```
RobotFramework.Testsuites.Keywords.CSetup
```

Definition setup keywords Method: testsuite_setup -----

5.1.1 Method: testsuite_teardown

5.1.2 Method: testcase_setup

5.1.3 Method: testcase_teardown

5.1.4 Method: update_config

5.2 Class: CGeneralKeywords

```
RobotFramework.Testsuites.Keywords.CSetup
```

Definition setup keywords Method: get_config -----

oConfigParams: is the dictionary consist of some configuration params which are return to user from
get_config_params keyword Method: load_json -----

Chapter 6

CStruct.py

6.1 Class: CStruct

```
RobotFramework.Testsuites.Utils.CStruct
```

Constructor `__init__` creates the given attributes dynamically at runtime.

Args:

Attributes to be created with the initial value

Returns:

Accessible attributes

Chapter 7

Event.py

7.1 Class: Event

```
RobotFramework.Testsuites.Utils/Events.Event
```

7.1.1 Method: trigger

Chapter 8

ScopeEvent.py

8.1 Class: ScopeEvent

```
RobotFramework.Testsuites.Utils/Events.ScopeEvent
```

8.1.1 Method: trigger

8.2 Class: ScopeStart

```
RobotFramework.Testsuites.Utils/Events.ScopeEvent
```

8.3 Class: ScopeEnd

```
RobotFramework.Testsuites.Utils/Events.ScopeEvent
```

Chapter 9

`__init__.py`

9.1 **Function:** `on`

9.2 **Function:** `dispatch`

9.3 **Function:** `register_event`

Chapter 10

LibListener.py

10.1 Class: LibListener

```
RobotFramework.Testsuites.Utls.LibListener
```

Define some hook methods

Chapter 11

`__init__.py`

11.1 Class: `RobotFramework.Testsuites`

```
RobotFramework.Testsuites.__init__
```

`RobotFramework.Testsuites` is the Bosch testing library for Robot Framework. `RobotFramework.Testsuites` control peripheral devices, tools and target under testing. Method: `run_keyword`

11.1.1 Method: `get_keyword_tags`

11.1.2 Method: `get_keyword_documentation`

11.1.3 Method: `failure_occurred`

11.2 Class: `CTestsuitesCfg`

```
RobotFramework.Testsuites.__init__
```

Chapter 12

version.py

12.1 Function: robfwaio_version

Return testsuitemanagement version as Robot framework AIO version

Chapter 13

Appendix

About this package:

Table 13.1: Package setup

Setup parameter	Value
Name	RobotFramework-Testsuites
Version	0.1.2
Date	05.07.2022
Description	Functionality to manage RobotFramework testsuites
Package URL	robotframework-testsuitesmanagement
Author	Mai Dinh Nam Son
Email	son.maidinhnam@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 14

History

0.1.0	06/2022
<i>Initial version</i>	

RobotFramework-Testsuites.pdf*Created at 11.07.2022 - 14:45:29**by GenPackageDoc v. 0.20.2*

4.6 QConnectBase

QConnectBase

v. 1.1.0

Nguyen Huynh Tri Cuong

05.07.2022

Contents

1	Introduction	1
2	Description	2
3	__init__.py	3
3.1	Class: ConnectionManager	3
4	connection_base.py	4
4.1	Class: BrokenConnError	4
4.2	Class: ConnectionBase	4
4.2.1	Method: is_precondition_pass	4
4.2.2	Method: error_instruction	4
4.2.3	Method: quit	4
4.2.4	Method: connect	5
4.2.5	Method: disconnect	5
4.2.6	Method: send_obj	5
4.2.7	Method: read_obj	5
4.2.8	Method: wait_4_trace	5
4.2.9	Method: wait_4_trace_continuously	6
4.2.10	Method: create_and_activate_trace_queue	6
4.2.11	Method: deactivate_and_delete_trace_queue	6
4.2.12	Method: activate_trace_queue	6
4.2.13	Method: deactivate_trace_queue	7
4.2.14	Method: check_timeout	7
4.2.15	Method: pre_msg_check	7
4.2.16	Method: post_msg_check	7
5	connection_manager.py	8
5.1	Class: InputParam	8
5.1.1	Method: get_attr_list	8
5.2	Class: ConnectParam	8
5.2.1	Method: add_connection	8
5.2.2	Method: remove_connection	9
5.2.3	Method: get_connection_by_name	9
5.2.4	Method: disconnect	9

CONTENTS

B

5.2.5	Method: <code>connect</code>	9
5.2.6	Method: <code>connect_named_args</code>	9
5.2.7	Method: <code>connect_unnamed_args</code>	9
5.2.8	Method: <code>send_command</code>	10
5.2.9	Method: <code>send_command_named_args</code>	10
5.2.10	Method: <code>send_command_unnamed_args</code>	10
5.2.11	Method: <code>verify</code>	10
5.2.12	Method: <code>verify_named_args</code>	10
5.2.13	Method: <code>verify_unnamed_args</code>	10
5.3	Class: <code>TestOption</code>	11
6	<code>constants.py</code>	12
6.1	Class: <code>SocketType</code>	12
6.2	Class: <code>String</code>	12
7	<code>qlogger.py</code>	13
7.1	Class: <code>ColorFormatter</code>	13
7.2	Class: <code>QFileHandler</code>	13
7.2.1	Method: <code>get_config_supported</code>	13
7.3	Class: <code>QDefaultFileHandler</code>	13
7.3.1	Method: <code>get_config_supported</code>	14
7.4	Class: <code>QConsoleHandler</code>	14
7.5	Class: <code>QLogger</code>	14
7.5.1	Method: <code>set_handler</code>	14
8	<code>serial_base.py</code>	15
8.1	Class: <code>SerialConfig</code>	15
8.1.1	Method: <code>disconnect</code>	15
8.1.2	Method: <code>quit</code>	15
8.2	Class: <code>SerialClient</code>	15
9	<code>raw_tcp.py</code>	16
9.1	Class: <code>RawTCPBase</code>	16
10	<code>ssh_client.py</code>	17
10.1	Class: <code>AuthenticationType</code>	17
10.2	Class: <code>SSHConfig</code>	17
10.2.1	Method: <code>close</code>	17
11	<code>tcp_base.py</code>	18
11.1	Class: <code>TCPConfig</code>	18
11.1.1	Method: <code>quit</code>	18
11.1.2	Method: <code>connect</code>	18
11.1.3	Method: <code>disconnect</code>	18
11.2	Class: <code>TCPBaseServer</code>	18

CONTENTS

C

11.2.1 Method: <code>connect</code>	19
11.2.2 Method: <code>disconnect</code>	19
11.3 Class: <code>TCPBaseClient</code>	19
11.3.1 Method: <code>disconnect</code>	19
12 <code>utils.py</code>	20
12.1 Class: <code>Singleton</code>	20
12.2 Class: <code>DictToClass</code>	20
12.3 Class: <code>Utils</code>	20
12.3.1 Method: <code>get_all_sub_classes</code>	20
12.3.2 Method: <code>is_valid_host</code>	21
12.3.3 Method: <code>execute_command</code>	21
12.3.4 Method: <code>kill_process</code>	21
12.3.5 Method: <code>caller_name</code>	21
12.3.6 Method: <code>load_library</code>	21
12.3.7 Method: <code>is_ascii_or_unicode</code>	21
12.4 Class: <code>Job</code>	21
12.4.1 Method: <code>stop</code>	21
12.4.2 Method: <code>run</code>	21
12.5 Class: <code>ResultType</code>	21
13 Appendix	23
14 History	24

Chapter 1

Introduction

QConnectBase

TO BE CONTINUED

Chapter 2

Description

QConnectBase

TO BE CONTINUED

Chapter 3

`__init__.py`

3.1 Class: ConnectionManager

```
QConnectBase.__init__
```

Class to manage all connections.

Chapter 4

connection_base.py

4.1 Class: BrokenConnError

```
QConnectBase.connection_base
```

4.2 Class: ConnectionBase

```
QConnectBase.connection_base
```

Base class for all connection classes. Method: `is_supported_platform` -----
 Check if current platform is supported.

Returns: True if platform is supported.
 False if platform is not supported.

4.2.1 Method: `is_precondition_pass`

Check for precondition.

Returns: True if passing the precondition.
 False if failing the precondition.

4.2.2 Method: `error_instruction`

Get the error instruction.

Returns: Error instruction string.

4.2.3 Method: `quit`

>> This method MUST be overridden in derived class << Abstract method for quitting the connection.

Args: `is_disconnect_all`: Determine if it's necessary to disconnect all connections.

Returns: None.

4.2.4 Method: connect

>> This method MUST be overridden in derived class << Abstract method for quitting the connection.

Args: device: Determine if it's necessary to disconnect all connections.

files: Determine if it's necessary to disconnect all connections.

test_connection: Determine if it's necessary to disconnect all connections.

Returns: None.

4.2.5 Method: disconnect

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

Args: device: Device name.

Returns: None.

4.2.6 Method: send_obj

Wrapper method to send message to a tcp connection.

Args: obj: Data to be sent. cr: Determine if it's necessary to add newline character at the end of command.

Returns: None

4.2.7 Method: read_obj

Wrapper method to get the response from connection.

Returns: Responded message.

4.2.8 Method: wait_4_trace

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

Args: search_obj : 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 : Determine if 'fetch block' feature is used.

end_of_block_pattern : The end of block pattern.

filter_pattern : Regular expression object to filter message line by line.

timeout : Optional timeout parameter specified as a floating point number in the unit 'seconds'.

fct_args: Optional list of function arguments passed to be sent.

Returns: None : If no trace message matched to the specified regular expression and a timeout occurred.

<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'.

4.2.9 Method: wait_4_trace_continuously

Getting trace log continuously without creating a new trace queue.

Args: trace_queue: Queue to store the traces.

timeout: Timeout for waiting a matched log.

fct.args: Arguments to be sent to connection.

Returns: None : If no trace message matched to the specified regular expression and a timeout occurred.

match object : If a trace message has matched to the specified regular expression, a match object is returned as the result. The complete trace message can be accessed by the 'string' attribute of the match object. For access to groups within the regular expression, use the group() method. For more information, refer to Python documentation for module 're'.

4.2.10 Method: create_and_activate_trace_queue

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

Args: search_element : 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 : Determine if 'fetch block' feature is used.

end_of_block_pattern : The end of block pattern.

regex_line_filter_pattern : Regular expression object to filter message line by line.

Returns: trq_handle, trace_queue: the handle and search object

4.2.11 Method: deactivate_and_delete_trace_queue

Deactivate trace queue and delete.

Args: trq_handle: Trace queue handle.

trace_queue: Trace queue object.

Returns: None.

4.2.12 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.

Args: search_obj : 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 : 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 : Determine if 'fetch block' feature is used

end_of_block_pattern : The end of block pattern

line_filter_pattern : Regular expression object to filter message line by line.

Returns: <int> : Handle to deactivate the message filter.

4.2.13 Method: deactivate_trace_queue

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

Args: handle : Integer object returned by ActivateTraceQ() method.

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

4.2.14 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.

Args: msg: Responded message for checking.

Returns: None.

4.2.15 Method: pre_msg_check

>> This method will be override in derived class << Pre-checking message when receiving it from connection.

Args: msg: received message to be checked.

Returns: None.

4.2.16 Method: post_msg_check

>> This method will be override in derived class << Post-checking message when receiving it from connection.

Args: msg: received message to be checked.

Returns: None.

Chapter 5

connection_manager.py

5.1 Class: InputParam

```
QConnectBase.connection_manager
```

5.1.1 Method: get_attr_list

5.2 Class: ConnectParam

```
QConnectBase.connection_manager
```

Class for storing parameters for connect action. Class: SendCommandParam =====

```
QConnectBase.connection_manager
```

Class for storing parameters for send command action. Class: VerifyParam =====

```
QConnectBase.connection_manager
```

Class for storing parameters for verify action. Class: ConnectionManager =====

```
QConnectBase.connection_manager
```

Class to manage all connections. Method: quit -----

Quit connection manager.

Returns: None.

5.2.1 Method: add_connection

Add a connection to managed dictionary.

Args: name: connection's name.

conn: connection object.

Returns: None.

5.2.2 Method: `remove_connection`

Remove a connection by name.

Args: `connection_name`: connection name.

Returns: None.

5.2.3 Method: `get_connection_by_name`

Get an exist connection by name.

Args: `connection_name`: connection's name.

Returns: Connection object.

5.2.4 Method: `disconnect`

Keyword for disconnecting a connection by name.

Args: `connection_name`: Name of connection.

Returns: None.

5.2.5 Method: `connect`

Keyword for making a connection.

Args: `args`: Non-Keyword Arguments.

`kwargs`: Keyword Arguments.

Returns: None.

5.2.6 Method: `connect_named_args`

Making a connection with name arguments.

Args: `kwargs`: Dictionary of arguments.

Returns: None.

5.2.7 Method: `connect_unnamed_args`

Making a connection.

Args: `connection_name`: Name of connection.

`connection_type`: Type of connection.

`mode`: Connection mode.

`config`: Configuration for connection.

Returns: None.

5.2.8 Method: `send_command`

Keyword for sending command to a connection.

Args: args: Non-Keyword Arguments.

kwargs: Keyword Arguments.

Returns: None.

5.2.9 Method: `send_command_named_args`

Send command to a connection with name arguments.

Args: args: Dictionary of arguments.

Returns: None.

5.2.10 Method: `send_command_unnamed_args`

Send command to a connection.

Args: connection_name: connection's name.

command: command.

Returns: None.

5.2.11 Method: `verify`

Keyword uses to verify a pattern from connection response after sending a command.

Args: args: Non-Keyword Arguments.

kwargs: Keyword Arguments.

Returns: match_res: matched string.

5.2.12 Method: `verify_named_args`

Verify a pattern from connection response after sending a command with named arguments.

Args: kwargs: Dictionary of arguments.

Returns: match_res: matched string.

5.2.13 Method: `verify_unnamed_args`

Verify a pattern from connection response after sending a command.

Args: connection_name: connection's name.

search_obj: search pattern.

timeout: timeout for waiting result.

fetch_block: use fetch block feature.

end_of_block_pattern: pattern for detecting the end of block.

filter_pattern: line filter pattern.

fct_args: command to be sent.

Returns: match_res: matched string.

5.3 Class: TestOption

```
QConnectBase.connection_manager
```


Chapter 6

constants.py

6.1 Class: SocketType

```
QConnectBase.constants
```

6.2 Class: String

```
QConnectBase.constants
```

Chapter 7

qlogger.py

7.1 Class: ColorFormatter

`QConnectBase.qlogger`

Custom formatter class for setting log color. Method: `format` -----
Set the color format for the log.

Args: record: log record.

Returns: Log with color formatter.

7.2 Class: QFileHandler

`QConnectBase.qlogger`

Handler class for user defined file in config. Method: `get_log_path` -----
Get the log file path for this handler.

Args: config: connection configurations.

Returns: Log file path.

7.2.1 Method: `get_config_supported`

Check if the connection config is supported by this handler.

Args: config: connection configurations.

Returns: True if the config is supported.
False if the config is not supported.

7.3 Class: QDefaultFileHandler

```
QConnectBase.qlogger
```

Handler class for default log file path. Method: `get_log_path` -----

Get the log file path for this handler.

Args: `logger_name`: name of the logger.

Returns: Log file path.

7.3.1 Method: `get_config_supported`

Check if the connection config is supported by this handler.

Args: `config`: connection configurations.

Returns: True if the config is supported.

False if the config is not supported.

7.4 Class: `QConsoleHandler`

```
QConnectBase.qlogger
```

Handler class for console log. Method: `get_config_supported` -----

Check if the connection config is supported by this handler.

Args: `config`: connection configurations.

Returns: True if the config is supported.

False if the config is not supported.

7.5 Class: `QLogger`

```
QConnectBase.qlogger
```

Logger class for QConnect Libraries. Method: `get_logger` -----

Get the logger object.

Args: `logger_name`: Name of the logger.

Returns: Logger object.

7.5.1 Method: `set_handler`

Set handler for logger.

Args: `config`: connection configurations.

Returns: None if no handler is set. Handler object.

Chapter 8

serial_base.py

8.1 Class: SerialConfig

```
QConnectBase.serialclient.serial_base
```

Class to store the configuration for Serial connection. Class: SerialSocket =====

```
QConnectBase.serialclient.serial_base
```

Class for handling serial connection. Method: connect -----

Connect to serial port.

Returns: None.

8.1.1 Method: disconnect

Disconnect serial port.

Args: _device: unused.

Returns: None.

8.1.2 Method: quit

Quit serial connection.

Returns: None

8.2 Class: SerialClient

```
QConnectBase.serialclient.serial_base
```

Serial client class. Method: connect -----

Connect to the Serial port.

Returns: None.

Chapter 9

raw_tcp.py

9.1 Class: RawTCPBase

```
QConnectBase.tcp/raw.raw_tcp
```

Base class for a raw tcp connection. Class: RawTCPServer =====

```
QConnectBase.tcp/raw.raw_tcp
```

Class for a raw tcp connection server. Class: RawTCPClient =====

```
QConnectBase.tcp/raw.raw_tcp
```

Class for a raw tcp connection client.

Chapter 10

ssh_client.py

10.1 Class: AuthenticationType

`QConnectBase.tcp/ssh.ssh_client`

10.2 Class: SSHConfig

`QConnectBase.tcp/ssh.ssh_client`

Class to store the configuration for SSH connection. Class: SSHClient =====

`QConnectBase.tcp/ssh.ssh_client`

SSH client connection class. Method: connect -----

Implementation for creating a SSH connection.

Returns: None.

10.2.1 Method: close

Close SSH connection.

Returns: Method: quit -----

Quit and stop receiver thread.

Returns: None.

Chapter 11

tcp_base.py

11.1 Class: TCPConfig

```
QConnectBase.tcp.tcp_base
```

Class to store configurations for TCP connection. Class: TCPBase =====

```
QConnectBase.tcp.tcp_base
```

Base class for a tcp connection. Method: close -----

Close connection.

Returns: None.

11.1.1 Method: quit

Quit connection.

Args: is_disconnect_all: Determine if it's necessary for disconnect all connection.

Returns: None.

11.1.2 Method: connect

>> Should be override in derived class. Establish the connection.

Returns: None.

11.1.3 Method: disconnect

>> Should be override in derived class. Disconnect the connection.

Returns: None.

11.2 Class: TCPBaseServer

```
QConnectBase.tcp.tcp_base
```

Base class for TCP server. Method: `accept_connection` -----

Wrapper method for handling accept action of TCP Server.

Returns: None.

11.2.1 Method: connect

11.2.2 Method: disconnect

11.3 Class: TCPBaseClient

```
QConnectBase.tcp.tcp_base
```

Base class for TCP client. Method: `connect` -----

11.3.1 Method: disconnect

Chapter 12

utils.py

12.1 Class: Singleton

`QConnectBase.utils`

Class to implement Singleton Design Pattern. This class is used to derive the `TTFisClientReal` as only a single instance of this class is allowed.

Disabled pyLint Messages: R0903: Too few public methods (%s/%s) Used when class has too few public methods, so be sure it's really worth it.

This base class implements the Singleton Design Pattern required for the `TTFisClientReal`. Adding further methods does not make sense.

12.2 Class: DictToClass

`QConnectBase.utils`

Class for converting dictionary to class object. Method: `validate` -----

12.3 Class: Utils

`QConnectBase.utils`

Class to implement utilities for supporting development. Method: `get_all_descendant_classes` -----

Get all descendant classes of a class

Args: `cls`: Input class for finding descendants.

Returns: Array of descendant classes.

12.3.1 Method: `get_all_sub_classes`

Get all children classes of a class

Args: cls: Input class for finding children.

Returns: Array of children classes.

12.3.2 Method: is_valid_host

12.3.3 Method: execute_command

12.3.4 Method: kill_process

12.3.5 Method: caller_name

Get a name of a caller in the format module.class.method

Args: skip: 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: An empty string is returned if skipped levels exceed stack height

12.3.6 Method: load_library

Load native library depend on the calling convention.

Args: path: library path. is_stdcall: determine if the library's calling convention is stdcall or cdecl.

Returns: Loaded library object.

12.3.7 Method: is_ascii_or_unicode

Check if the string is ascii or unicode

Args: str_check: string for checking codecs: encoding type list

Returns: True : if checked string is ascii or unicode False : if checked string is not ascii or unicode

12.4 Class: Job

```
QConnectBase.utils
```

12.4.1 Method: stop

12.4.2 Method: run

12.5 Class: ResultType

```
QConnectBase.utils
```

Result Types. Class: ResponseMessage =====

```
QConnectBase.utils
```

Response message class Method: get_json -----

Convert response message to json Returns: Response message in json format Method: get_data -----

Get string data result Returns: String result Method: create_from_string -----

Chapter 13

Appendix

About this package:

Table 13.1: Package setup

Setup parameter	Value
Name	QConnectBase
Version	1.1.0
Date	05.07.2022
Description	Robot Framework test library for TCP, SSH, serial connection
Package URL	robotframework-qconnect-base
Author	Nguyen Huynh Tri Cuong
Email	cuong.nguyenuynhtri@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 14

History

1.0.0	2022
<i>Initial version</i>	

QConnectBase.pdf

Created at 11.07.2022 - 14:45:39

by GenPackageDoc v. 0.20.2

4.7 RobotResults2RQM

RobotResults2RQM

v. 1.1.1

Tran Duy Ngoan

05.07.2022

Contents

1	Introduction	1
2	Description	2
3	CRQM.py	3
3.1	Function: <code>get_xml_tree</code>	3
3.2	Class: <code>CRQMClient</code>	3
3.2.1	Method: <code>login</code>	3
3.2.2	Method: <code>verifyProjectName</code>	4
3.2.3	Method: <code>disconnect</code>	4
3.2.4	Method: <code>userURL</code>	4
3.2.5	Method: <code>integrationURL</code>	4
3.2.6	Method: <code>webIDfromResponse</code>	5
3.2.7	Method: <code>webIDfromGeneratedID</code>	5
3.2.8	Method: <code>getResourceByID</code>	5
3.2.9	Method: <code>getAllByResource</code>	5
3.2.10	Method: <code>getAllBuildRecords</code>	5
3.2.11	Method: <code>addTeamAreaNode</code>	6
3.2.12	Method: <code>createTestcaseTemplate</code>	6
3.2.13	Method: <code>createTCERTemplate</code>	6
3.2.14	Method: <code>createExecutionResultTemplate</code>	6
3.2.15	Method: <code>createBuildRecordTemplate</code>	7
3.2.16	Method: <code>createConfigurationTemplate</code>	7
3.2.17	Method: <code>createTSERTemplate</code>	7
3.2.18	Method: <code>createTestsuiteResultTemplate</code>	7
3.2.19	Method: <code>createResource</code>	8
3.2.20	Method: <code>createBuildRecord</code>	8
3.2.21	Method: <code>createConfiguration</code>	8
3.2.22	Method: <code>updateResourceByID</code>	8
3.2.23	Method: <code>linkListTestcase2Testplan</code>	8
3.2.24	Method: <code>linkListTestcase2Testsuite</code>	9
4	<code>__init__.py</code>	10
5	<code>robot2rqm.py</code>	11

CONTENTS

B

5.1	Function: <code>get_from_tags</code>	11
5.2	Function: <code>convert_to_datetime</code>	11
5.3	Function: <code>process_suite_metadata</code>	11
5.4	Function: <code>process_metadata</code>	11
5.5	Function: <code>process_suite</code>	12
5.6	Function: <code>process_test</code>	12
5.7	Function: <code>RobotResults2RQM</code>	12
5.8	Class: <code>Logger</code>	12
5.8.1	Method: <code>log</code>	13
5.8.2	Method: <code>log_warning</code>	13
5.8.3	Method: <code>log_error</code>	13
6	Appendix	14
7	History	15

Chapter 1

Introduction

RobotResults2RQM

!!! add introduction - if wanted !!

Chapter 2

Description

The RobotResults2RQM:
TO BE CONTINUED

Chapter 3

CRQM.py

3.1 Function: get_xml_tree

Parse xml object from file.

Args: file_name : path to file or file-like object. bdd_validation : if True, validate against a DTD referenced by the document.

Returns: oTree : xml etree object

3.2 Class: CRQMClient

```
RobotResults2RQM.CRQM
```

CRQMClient class uses RQM REST APIs to get, create and update resources (testplan, testcase, test result, ...) on RQM - Rational Quality Manager

Resource 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.

Note: When the authentication is successful, the JSESSIONID from cookies will be stored as header for later POST method.

Returns: True if successful, False otherwise.

3.2.2 Method: verifyProjectName

Verify the project name by searching it in project-areas XML response.

Note: The found project ID will be stored and:

- required for team-areas request (project name cannot be used)
- used for all later request urls instead of project name

Returns:

- True if the authentication is successful.
- False if the authentication is failed.

3.2.3 Method: disconnect

Disconnect from RQM 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.

Args: plan_id : testplan ID of RQM project for importing result(s).

build_name (optional) : the Build Record for linking result(s). Set it to None if not be used, the empty name " " may lead to error.

config_name (optional) : the Test Environment for linking result(s). Set it to None if not be used, the empty name " " may lead to error.

createmissing (optional) : in case this argument is set to True, the testcase without tcid information will be created on RQM.

updatetestcase (optional) : in case this argument is set to True, the information of testcase on RQM will be updated bases on robot testfile.

suite_id (optional) : testsuite ID of RQM project for importing result(s).

Returns: None.

3.2.4 Method: userURL

Return interaction URL of provided userID

Args: userID : the user ID

Returns: userURL : the interaction URL of provided userID

3.2.5 Method: integrationURL

Return interaction URL of provided resource and ID.

Note: ID can be internalID (contains only digits) or externalID.

Args: resourceType : the RQM resource type (e.g: "testplan", "testcase", ...)

id (optional) : 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 (optional) : force to return the url of resource as internal ID.

Returns: integrationURL : interaction URL of provided resource and ID.

3.2.6 Method: webIDfromResponse

Get internal ID (number) from response of POST method.

Note: Only executionresult has response text. Other resources has only response header.

Args: response : the response from POST method.

tagID : tag name which contains ID information.

Returns: resultId : internal ID (as number).

3.2.7 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.

Args: resourceType : the RQM resource type.

generateID : the Slug ID which is returned in Content-Location from POST response.

Returns: webID : web ID (number).

3.2.8 Method: getResourceByID

Return data of provided resource and ID by GET method

Args: resourceType : the RQM resource type.

id : ID of resource.

Returns: res : response data of GET request.

3.2.9 Method: getAllByResource

Return all entries of provided resource by GET method.

Note: This method will try to fetch all entries in all pages of resource.

Args: resourceType : the RQM resource type.

Returns: dReturn : a dictionary which contains response status, message and data.

3.2.10 Method: getAllBuildRecords

Get all available build records of project on RQM and store them into dBuildVersion property. Method: getAllConfigurations -----

Get all available configurations of project on RQM and store them into dConfiguration property. Method: getAllTeamAreas -----

Get all available team-areas of project on RQM and store them into dTeamAreas property.

Example: { 'teamA' : '{host}/qm/process/project-areas/{project-id}/team-areas/{teamA-id}', 'teamB' : '{host}/qm/process/project-areas/{project-id}/team-areas/{teamB-id}' }

3.2.11 Method: addTeamAreaNode

Append team-area node which contains URL to given team-area into xml template

Note: team-area information is case-casesensitive

Args: root : xml root object.

sTeam : team name to be added.

Returns: root : xml root object with addition team-area node.

3.2.12 Method: createTestcaseTemplate

Return testcase template from provided information.

Args: testCaseName : testcase name.

sDescription (optional) : testcase description.

sComponent (optional) : component which testcase is belong to.

sFID (optional) : function ID(requirement ID) for linking.

sTeam (optional) : team name for linking.

sRobotFile (optional) : link to robot file on source control.

sTestType (optional) : test type information.

sASIL (optional) : ASIL information.

sOwnerID (optional) : user ID of testcase owner.

sTCtemplate (optional) : existing testcase template as xml string. If not provided, template file under *RQM_templates* is used as default.

Returns: xml template as string.

3.2.13 Method: createTCERTemplate

Return testcase execution record template from provided information

Args: testCaseID : testcase ID.

testCaseName : testcase name.

testplanID : testplan ID for linking.

confID (optional) : configuration - Test Environment for linking.

sTeam (optional) : team name for linking.

sOwnerID (optional) : user ID of testcase owner.

Returns: xml template as string.

3.2.14 Method: createExecutionResultTemplate

Return testcase execution result template from provided information

Args: testCaseID : testcase ID.

testCaseName : testcase name.

testplanID : testplan ID for linking.

TCERID : testcase execution record (TCER) ID for linking.

resultState : testcase result status.

startTime : testcase start time.
 endTime (optional) : testcase end time.
 duration (optional) : testcase duration.
 testPC (optional) : test PC which executed testcase.
 testBy (optional) : user ID who executed testcase.
 lastlog (optional) : traceback information (for Failed testcase).
 buildrecordID (optional) : Build Record ID for linking.
 sTeam (optional) : team name for linking.
 sOwnerID (optional) : user ID of testcase owner.

Returns: xml template as string.

3.2.15 Method: createBuildRecordTemplate

Return build record template from provided build name

Args: buildName : Build Record name.

Returns: xml template as string.

3.2.16 Method: createConfigurationTemplate

Return configuration - Test Environment template from provided configuration name

Args: buildName : configuration - Test Environment name.

Returns: xml template as string.

3.2.17 Method: createTSERTemplate

Return testsuite execution record (TSER) template from provided configuration name

Args: testsuiteID : testsuite ID.

testsuiteName : testsuite name.
 testplanID : testplan ID for linking.
 confID (optional) : configuration - Test Environment ID for linking.
 sOwnerID (optional) : user ID of testsuite owner.

Returns: xml template as string.

3.2.18 Method: createTestsuiteResultTemplate

Return testsuite execution result template from provided configuration name

Args: testsuiteID : testsuite ID.

testsuiteName : testsuite name.
 TSERID : testsuite execution record (TSER) ID for linking.
 ITCER : list of testcase execution records (TCER) for linking.
 ITCResults : list of testcase results for linking.
 startTime (optional) : testsuite start time.
 endTime (optional) : testsuite end time.
 duration (optional) : testsuite duration.
 sOwnerID (optional) : user ID of testsuite owner.

Returns: xml template as string.

3.2.19 Method: createResource

Create new resource with provided data from template by POST method.

Args: resourceType : resource type.

content: xml template as string.

Returns: returnObj: a response dictionary which contains status, ID, status_code and error message.
 { 'success' : False, 'id': None, 'message': ", 'status_code': " }

3.2.20 Method: createBuildRecord

Create new build record.

Note: Tool will check if build record is already existing or not (both on RQM and current execution).

Args: sBuildSWVersion : build version - Build Record name.

forceCreate (optional) : if True, force to create new build record without existing verification.

Returns: returnObj: a response dictionary which contains status, ID, status_code and error message.
 { 'success' : False, 'id': None, 'message': ", 'status_code': " }

3.2.21 Method: createConfiguration

Create new configuration - test environment.

Note: Tool will check if configuration is already existing or not (both on RQM and current execution).

Args: sConfigurationName : configuration - Test Environment name.

forceCreate (optional) : if True, force to create new Test Environment without existing verification.

Returns: returnObj: a response dictionary which contains status, ID, status_code and error message.
 { 'success' : False, 'id': None, 'message': ", 'status_code': " }

3.2.22 Method: updateResourceByID

Update data of provided resource and ID by PUT method.

Args: resourceType : resource type.

id : resource id.

content : xml template as string.

Returns: res : response object from PUT request.

3.2.23 Method: linkListTestcase2Testplan

Link list of test cases to provided testplan ID.

Args: testplanID : testplan ID to link given testcase(s).

lTestcases : list of testcase(s) to be linked with given testplan. If None (as default), lTestcaseIDs property will be used as list of testcase.

Returns: res : response object which contains status and error message. { 'success' : False, 'message': " }

3.2.24 Method: linkListTestcase2Testsuite

Link list of test cases to provided testsuite ID

Args: testsuiteID : testsuite ID to link given testcase(s).

lTestcases : list of testcase(s) to be linked with given testsuite. If None (as default), lTestcaseIDs property will be used as list of testcase.

Returns: res : response object which contains status and error message. { 'success' : False, 'message': " " }

Chapter 4

`--init--.py`

RobotResults2RQM package provides an interface to import robot result(s) from output.xml result file to IBM Rational Quality Manager (RQM).

Chapter 5

robot2rqm.py

5.1 Function: get_from_tags

Extract testcase information from tags.

Example: TCID-xxxx, FID-xxxx, ...

Args: lTags : list of tag information.

reInfo : regex to get the expected info (ID) from tag info.

Returns: lInfo : list of expected information (ID)

5.2 Function: convert_to_datetime

Convert time string to datetime.

Args: time : string of time.

Returns: dt : datetime object

5.3 Function: process_suite_metadata

Try to find metadata information from all suite levels.

Note: Metadata at top suite level has a highest priority.

Args: suite : Robot suite object.

default_metadata: initial Metadata information for updating.

Returns: dMetadata : dictionary of Metadata information.

5.4 Function: process_metadata

Extract metadata from suite result bases on DEFAULT_METADATA

Args: metadata : Robot metadata object.

default_metadata: initial Metadata information for updating.

Returns: dMetadata : dictionary of Metadata information.

5.5 Function: process_suite

process robot suite for importing to RQM.

Args: RQMClient : RQMClient object.

suite : Robot suite object.

Returns: None.

5.6 Function: process_test

process robot test for importing to RQM.

Args: RQMClient : RQMClient object.

test : Robot test object.

Returns: None.

5.7 Function: RobotResults2RQM

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

Args: args : Argument parser object:

- outputfile : path to the output file or directory with output files to be imported.
- host : RQM host url.
- project : RQM project name.
- user : user for RQM login.
- password : user password for RQM login.
- testplan : RQM testplan ID.
- recursive : if True, then the path is searched recursively for log files to be imported.
- createmissing : if True, then all testcases without fcid are created when importing.
- updatetestcase : if True, then testcases information on RQM will be updated bases on robot testfile.
- dryrun : if True, then just check the RQM authentication and show what would be done.

Returns: None.

5.8 Class: Logger

```
RobotResults2RQM.robot2rqm
```

Logger class for logging message. Method: config -----

Configure Logger class.

Args: output_console : write message to console output.
output_logfile : path to log file output.
indent : offset indent.
dryrun : if set, a prefix as 'dryrun' is added for all messages.

Returns: None.

5.8.1 Method: log

Write log message to console/file output.

Args: msg : message to write to output.
color : color style for the message.
indent : offset indent.

Returns: None.

5.8.2 Method: log_warning

Write warning message to console/file output.

Args: msg : message to write to output.

Returns: None.

5.8.3 Method: log_error

Write error message to console/file output.

Args: msg : message to write to output.
fatal_error : if set, tool will terminate after logging error message.

Returns: None.

Chapter 6

Appendix

About this package:

Table 6.1: Package setup

Setup parameter	Value
Name	RobotResults2RQM
Version	1.1.1
Date	05.07.2022
Description	Imports robot result(s) to IBM Rational Quality Manager (RQM)
Package URL	robotframework-testresult2rqmtool
Author	Tran Duy Ngoan
Email	Ngoan.TranDuy@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 7

History

0.1.0	07/2022
<i>Initial version</i>	

RobotResults2RQM.pdf

Created at 11.07.2022 - 14:45:49

by GenPackageDoc v. 0.20.2

4.8 RobotResults2DB

RobotResults2DB

v. 1.2.1

Tran Duy Ngoan

05.07.2022

Contents

1	Introduction	1
2	Description	2
3	CDataBase.py	3
3.1	Class: CDataBase	3
3.1.1	Method: disconnect	3
3.1.2	Method: nCreateNewFile	4
3.1.3	Method: vCreateNewHeader	4
3.1.4	Method: nCreateNewSingleTestCase	5
3.1.5	Method: nCreateNewTestCase	5
3.1.6	Method: vCreateTags	6
3.1.7	Method: vSetCategory	6
3.1.8	Method: vUpdateStartEndTime	6
3.1.9	Method: arGetCategories	6
3.1.10	Method: vCreateAbortReason	6
3.1.11	Method: vCreateReanimation	7
3.1.12	Method: vCreateCCRdata	7
3.1.13	Method: vFinishTestResult	7
3.1.14	Method: vUpdateEvtbls	7
3.1.15	Method: vUpdateFileEndTime	7
3.1.16	Method: vUpdateResultEndTime	7
3.1.17	Method: bExistingResultID	8
4	__init__.py	9
5	robot2db.py	10
5.1	Function: is_valid_uuid	10
5.2	Function: get_from_tags	10
5.3	Function: get_branch_from_swversion	10
5.4	Function: format_time	11
5.5	Function: process_suite_metadata	11
5.6	Function: process_metadata	11
5.7	Function: process_suite	11
5.8	Function: process_test	11

CONTENTS	B
5.9 Function: <code>process_config_file</code>	12
5.10 Function: <code>validate_config</code>	12
5.11 Function: <code>normailze_path</code>	12
5.12 Function: <code>truncate_string</code>	12
5.13 Function: <code>RobotResults2DB</code>	13
5.14 Class: <code>Logger</code>	13
5.14.1 Method: <code>log</code>	14
5.14.2 Method: <code>log_warning</code>	14
5.14.3 Method: <code>log_error</code>	14
6 Appendix	15
7 History	16

Chapter 1

Introduction

RobotResults2DB

!!! add introduction !!

Chapter 2

Description

The RobotResults2DB:

TO BE CONTINUED

Chapter 3

CDataBase.py

3.1 Class: CDataBase

RobotResults2DB.CDataBase

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

Connect to the database with provided authentication and db info.

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

user : user name for database authentication.

passwd : user password for database authentication.

database : database name.

charset (optional): the connection character set.

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

Returns: None.

3.1.1 Method: disconnect

Disconnect from TestResultWebApp's database. Method: cleanAllTables -----

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

Args: _tbl_prj_project : project information.

_tbl_prj_variant : variant information.

_tbl_prj_branch : branch information.

_tbl_test_result_id : UUID of test result.

_tbl_result_interpretation : result interpretation.

_tbl_result_time_start : test result start time.

_tbl_result_time_end : test result end time.

_tbl_result_version_sw_target : software version information.

`_tbl_result_version_sw_test` : test version information.
`_tbl_result_version_target` : hardware version information.
`_tbl_result_jenkinsurl` : jenkinsurl in case test result is executed by jenkins.
`_tbl_result_reporting_qualitygate` : qualitygate information for reporting.

Returns: `_tbl_test_result_id`: test_result_id.

3.1.2 Method: nCreateNewFile

Create new file entry in `tbl_file` table.

Args: `_tbl_file_name` : file name information.

`_tbl_file_tester_account` : tester account information.
`_tbl_file_tester_machine` : test machine information.
`_tbl_file_time_start` : test file start time.
`_tbl_file_time_end` : test file end time.
`_tbl_test_result_id` : UUID of test result for linking to `tbl_result` table.

`_tbl_file_origin` : **origin (test framework) of test file.** Deafult is "ROBFW"

Returns: `iInsertedID`: ID of new entry.

3.1.3 Method: vCreateNewHeader

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

Args: `_tbl_file_id` : file ID information.

`_tbl_header_testtoolconfiguration_testtoolname` : test tool name.
`_tbl_header_testtoolconfiguration_testtoolversionstring` : test tool version.
`_tbl_header_testtoolconfiguration_projectname` : project name.
`_tbl_header_testtoolconfiguration_logfileencoding` : encoding of logfile.
`_tbl_header_testtoolconfiguration_pythonversion` : Python version info.
`_tbl_header_testtoolconfiguration_testfile` : test file name.
`_tbl_header_testtoolconfiguration_logfilepath` : path to log file.
`_tbl_header_testtoolconfiguration_logfilemode` : mode of log file.
`_tbl_header_testtoolconfiguration_ctrlfilepath` : path to control file.
`_tbl_header_testtoolconfiguration_configfile` : path to configuration file.
`_tbl_header_testtoolconfiguration_confname` : configuration name.
`_tbl_header_testfileheader_author` : file author.
`_tbl_header_testfileheader_project` : project information.
`_tbl_header_testfileheader_testfiledate` : file creation date.
`_tbl_header_testfileheader_version_major` : file major version.
`_tbl_header_testfileheader_version_minor` : file minor version.
`_tbl_header_testfileheader_version_patch` : file patch version.
`_tbl_header_testfileheader_keyword` : file keyword.
`_tbl_header_testfileheader_shortdescription` : file short description.
`_tbl_header_testexecution_useraccount` : tester account who run the execution.
`_tbl_header_testexecution_computername` : machine name which is executed on.

`_tbl_header_testrequirements_documentmanagement` : requirement management information.
`_tbl_header_testrequirements_testenvironment` : requirement environment information.
`_tbl_header_testbenchconfig_name` : testbench configuration name.
`_tbl_header_testbenchconfig_data` : testbench configuration data.
`_tbl_header_preprocessor_filter` : preprocessor filter information.
`_tbl_header_preprocessor_parameters` : preprocessor parameters definition.

Returns: None.

3.1.4 Method: `nCreateNewSingleTestCase`

Create single testcase entry in `tbl_case` table immediately.

Args: `_tbl_case_name` : test case name.

`_tbl_case_issue` : test case issue ID.
`_tbl_case_tcid` : test case ID (used for testmanagement tool).
`_tbl_case_fid` : test case requirement (function) ID.
`_tbl_case_testnumber` : order of test case in file.
`_tbl_case_repeatcount` : test case repeatition count.
`_tbl_case_component` : component which test case is belong to.
`_tbl_case_time.start` : test case start time.
`_tbl_case_result_main` : test case main result.
`_tbl_case_result.state` : test case completion state.
`_tbl_case_result.return` : test case result code (as integer).
`_tbl_case_counter_resets` : counter of target reset within test case execution.
`_tbl_case_lastlog` : traceback information when test case is failed.
`_tbl_test_result_id` : UUID of test result for linking to file in `tbl_result` table.
`_tbl_file_id` : test file ID for linking to file in `tbl_file` table.

Returns: `iInsertedID`: ID of new entry.

3.1.5 Method: `nCreateNewTestCase`

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

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

Args: `_tbl_case_name` : test case name.

`_tbl_case_issue` : test case issue ID.
`_tbl_case_tcid` : test case ID (used for testmanagement tool).
`_tbl_case_fid` : test case requirement (function) ID.
`_tbl_case_testnumber` : order of test case in file.
`_tbl_case_repeatcount` : test case repeatition count.
`_tbl_case_component` : component which test case is belong to.
`_tbl_case_time.start` : test case start time.
`_tbl_case_result_main` : test case main result.
`_tbl_case_result.state` : test case completion state.
`_tbl_case_result.return` : test case result code (as integer).

`_tbl_case_counter_resets` : counter of target reset within test case execution.
`_tbl_case_lastlog` : traceback information when test case is failed.
`_tbl_test_result_id` : UUID of test result for linking to file in `tbl_result` table.
`_tbl_file_id` : test file ID for linking to file in `tbl_file` table.

Returns: None.

3.1.6 Method: vCreateTags

Create tag entries.

Args: `_tbl_test_result_id` : UUID of test result.
`_tbl_usr_result_tags` : user tags information.

Returns: None.

3.1.7 Method: vSetCategory

Create category entry.

Args: `_tbl_test_result_id` : UUID of test result.
`tbl_result_category_main` : category information.

Returns: None.

3.1.8 Method: vUpdateStartTimeEndTime

Create start-end time entry.

Args: `_tbl_test_result_id` : UUID of test result.
`_tbl_result_time_start` : result start time.
`_tbl_result_time_end` : result end time.

Returns: None.

3.1.9 Method: arGetCategories

Get existing categories.

Returns: `arCategories` : list of existing categories.

3.1.10 Method: vCreateAbortReason

Create abort reason entry.

Args: `_tbl_test_result_id` : UUID of test result.
`_tbl_abort_reason` : abort reason.
`_tbl_abort_message` : detail message of abort.

Returns: None.

3.1.11 Method: vCreateReanimation

Create reanimation entry.

Args: `_tbl_test_result_id` : UUID of test result.

`_tbl_num_of_reanimation` : counter of target reanimation during execution.

Returns: None.

3.1.12 Method: vCreateCCRdata

Create CCR data per test case.

Args: `_tbl_test_case_id` : test case ID.

`lCCRdata` : list of CCR data.

Returns: None.

3.1.13 Method: vFinishTestResult

Finish upload:

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

3.1.14 Method: vUpdateEvtbls

Call `update_evtbls` stored procedure. Method: `vUpdateEvtbl` -----

Call `update_evtbl` stored procedure to update provided `test_result_id`. Method: `vEnableForeignKeyCheck`

Switch `foreign_key_checks` flag. Method: `sGetLatestFileID` -----

Get latest file ID from `tbl_file` table.

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

Returns: `sFileID` : file ID.

3.1.15 Method: vUpdateFileEndTime

Update test file end time.

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

`sEndtime` : end time information.

Returns: None.

3.1.16 Method: vUpdateResultEndTime

Update test result end time.

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

`sEndtime` : end time information.

Returns: None.

3.1.17 Method: bExistingResultID

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

Args: sResultID : test result UUID to be verified.

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

Chapter 4

`__init__.py`

RobotResults2DB package provides an interface to import robot result(s) from output.xml result file to TestResultWebApp database.

TestResultWebApp is available in Github as below repo: <https://github.com/test-fullautomation/TestResultWebApp>

Chapter 5

robot2db.py

5.1 Function: is_valid_uuid

Verify the given UUID is valid or not.

Args: uuid_to_test : UUID to be verified.
version (optional): UUID version.

Returns: True if the given UUID is valid.

5.2 Function: get_from_tags

Extract testcase information from tags.

Example: TCID-xxxx, FID-xxxx, ...

Args: lTags : list of tag information.
reInfo : regex to get the expected info (ID) from tag info.

Returns: lInfo : list of expected information (ID)

5.3 Function: get_branch_from_swversion

Get branch name from software version information.

Convention of branch information in suffix of software version:

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

Args: sw_version : software version.

Returns: branch_name : branch name.

5.4 Function: format_time

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

Args: stime : string of time.

Returns: TestResultWebApp's time format.

5.5 Function: process_suite_metadata

Try to find metadata information from all suite levels.

Note: Metadata at top suite level has a highest priority.

Args: suite : Robot suite object.

default_metadata: initial Metadata information for updating.

Returns: dMetadata : dictionary of Metadata information.

5.6 Function: process_metadata

Extract metadata from suite result bases on DEFAULT_METADATA

Args: metadata : Robot metadata object.

default_metadata: initial Metadata information for updating.

Returns: dMetadata : dictionary of Metadata information.

5.7 Function: process_suite

Process to the lowest suite level (test file):

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

Args: suite : Robot suite object.

_tbl_test_result_id : UUID of test result for importing.

root_metadata : metadata information.

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

Returns: None.

5.8 Function: process_test

Process test case data and create new test case record.

Args: db : database object.

test : Robot test object.

file_id : file ID for mapping.

test_result_id : test result ID for mapping.

metadata.info : metadata information.

test_number : order of test case in file.

Returns: None.

5.9 Function: process_config_file

Parse information from configuration file:

- 'component': :

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

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

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

Args: config_file : path to configuration file.

Returns: dConfig : configuration object.

5.10 Function: validate_config

Validate the json configuration base on given schema.

Default schema just supports "component", "variant" and "version_sw" :

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

Args: dConfig : json configuration object to be verified.

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

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

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

5.11 Function: normailze_path

Normalize path file.

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

Returns: sNPath : string of normalized path file.

5.12 Function: truncate_string

Truncate input string before importing to database.

Args: sString : input string for truncation.

iMaxLength : max length of string to be allowed.

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

Returns: content : string after truncation.

5.13 Function: RobotResults2DB

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

Flow to import Robot results to database:

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

Args: args : Argument parser object:

- outfile : path to the output file or directory with output files to be imported.
- server : server which hosts the database (IP or URL).
- user : user for database login.
- password : password for database login.
- database : database name.
- recursive : if True, then the path is searched recursively for log files to be imported.
- dryrun : if True, then just check the RQM authentication and show what would be done.
- UUID : UUID used to identify the import and version ID on TestResultWebApp.
- variant : variant name to be set for this import.
- versions : metadata: Versions (Software;Hardware;Test) to be set for this import.
- config : configuration json file for component mapping information.

Returns: None.

5.14 Class: Logger

```
RobotResults2DB.robot2db
```

Logger class for logging message Method: config -----

Configure Logger class.

Args: output_console : write message to console output.

output_logfile : path to log file output.

indent : offset indent.

dryrun : if set, a prefix as 'dryrun' is added for all messages.

Returns: None.

5.14.1 Method: log

Write log message to console/file output.

Args: msg : message to write to output.

color : color style for the message.

indent : offset indent.

Returns: None.

5.14.2 Method: log_warning

Write warning message to console/file output.

Args: msg : message to write to output.

Returns: None.

5.14.3 Method: log_error

Write error message to console/file output.

Args: msg : message to write to output.

fatal_error : if set, tool will terminate after logging error message.

Returns: None.

Chapter 6

Appendix

About this package:

Table 6.1: Package setup

Setup parameter	Value
Name	RobotResults2DB
Version	1.2.1
Date	05.07.2022
Description	Imports robot result(s) to TestResultWebApp database
Package URL	robotframework-testresultwebapptool
Author	Tran Duy Ngoan
Email	Ngoan.TranDuy@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 7

History

0.1.0	07/2022
<i>Initial version</i>	

RobotResults2DB.pdf

Created at 11.07.2022 - 14:45:57

by GenPackageDoc v. 0.20.2

4.9 TestResultWebApp

TestResultWebApp

v. 0.1.0

Thomas Pollerspöck

04.07.2022

Contents

1	Introduction	1
1.1	Impressions	2
2	Description	6
3	Appendix	7
4	History	8

Chapter 1

Introduction

This package provides a generic web based open source application which allows processing displaying of test result data in a sophisticated way.

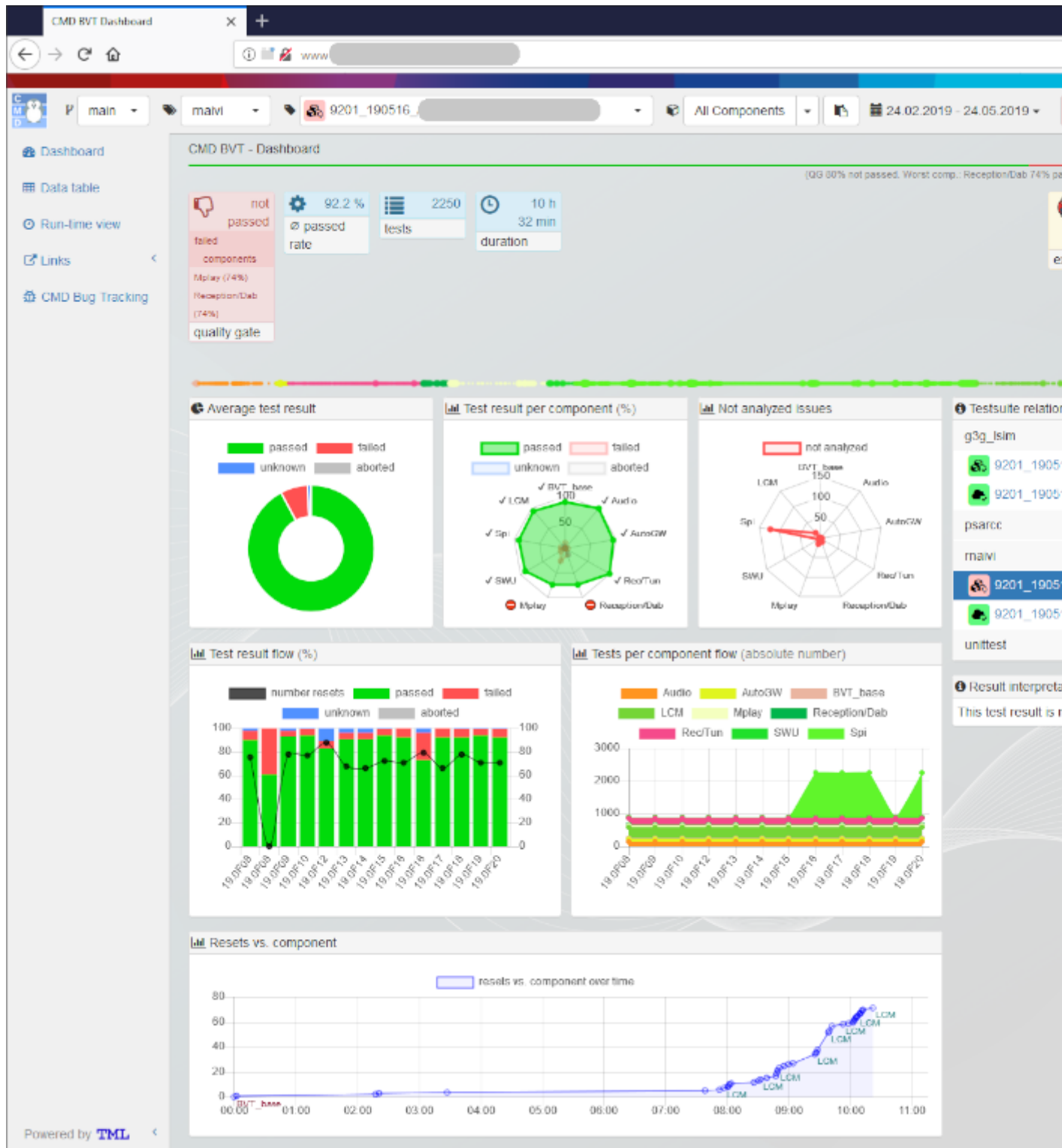
TO BE CONTINUED

(Place the content of

HowTo_curl_test_TODO_convertto_rst.txt and

HowTo_NewWebApp_instance_TODO_convertto_rst.txt here)

1.1 Impressions



The screenshot displays the CMD BVT Dashboard in a web browser. The dashboard has a sidebar with navigation links: Dashboard, Data table, Run-time view, Links, and CMD Bug Tracking. The main content area is titled "CMD BVT - data table" and shows a table of test results. The table has columns for Result, Name, Component, and tcid. The results are filtered to show 10 entries. The first entry is "INVALID sink connection" with a green checkmark. The second entry is "Connected to AUX_2" with a green checkmark. The third entry is "change source from AUX_2 to TUNER_AM and sink internal Amp" with a green checkmark. The fourth entry is "change source from AUX_2 to TUNER_FM and sink internal Amp" with a green checkmark. The fifth entry is "change source from AUX_2 to MEDIA_PLAYER and sink internal Amp" with a green checkmark. The sixth entry is "change source from AUX_2 to AUX_2 and sink internal Amp" with a green checkmark. The seventh entry is "INVALID sink connection" with a green checkmark. The eighth entry is "Connect to TUNER_AM" with a green checkmark. The ninth entry is "change source from TUNER_AM to TUNER_FM and sink internal Amp" with a green checkmark. The tenth entry is "change source from TUNER_AM to TUNER_FM and sink internal Amp" with a green checkmark. The table also shows a summary of test results: 100% not passed, Worst comp.: ReceptionTab 74%. There are buttons for "clear all filters" and "not passed 178".

Dashboard

Data table

Run-time view

Links

CMD Bug Tracking

CMD BVT Dashboard

main

main

9201_190516

All Components

24.02.2019 - 24.05.2019

CMD BVT - data table

(QG 80% not passed, Worst comp.: ReceptionTab 74%)

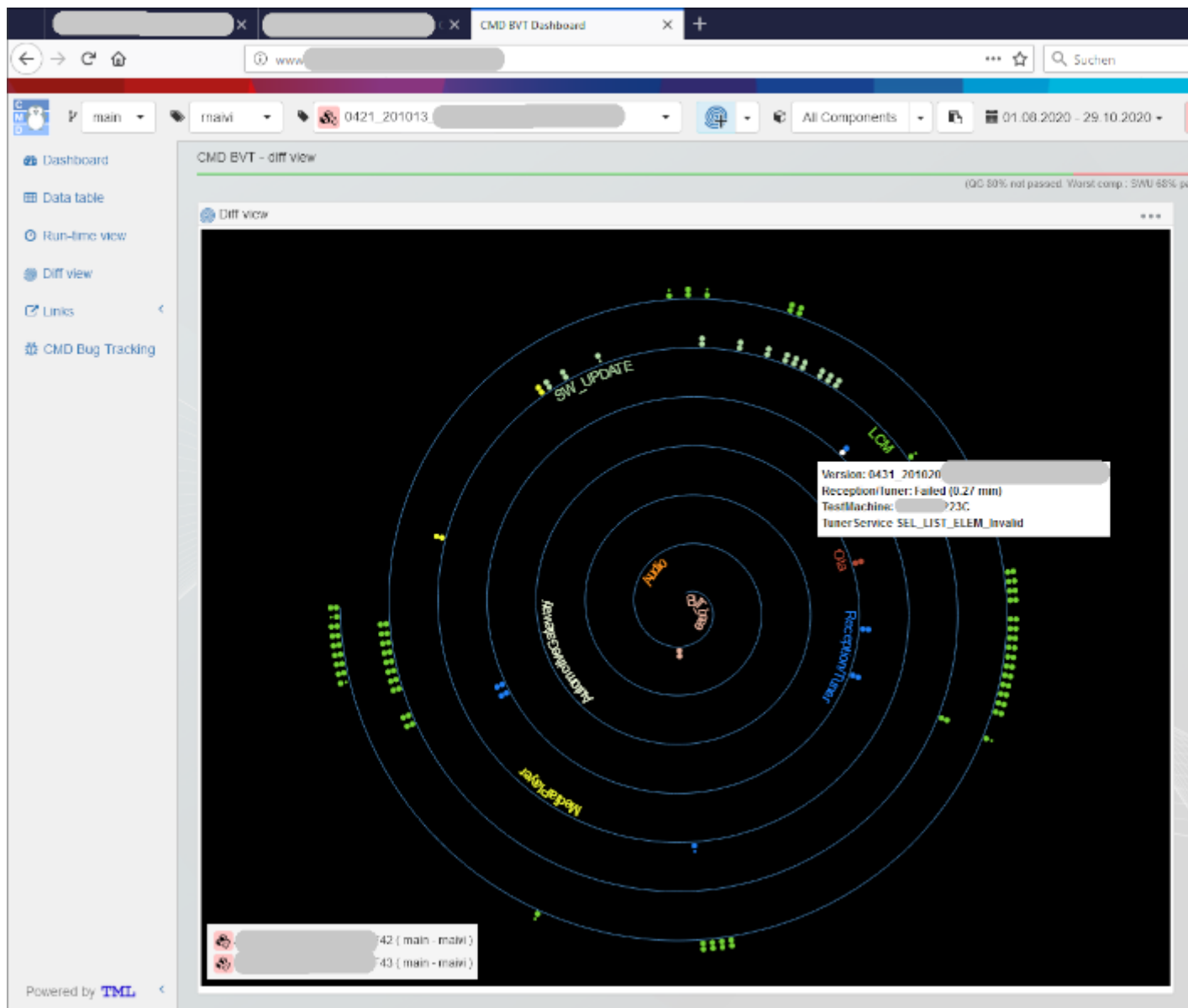
clear all filters

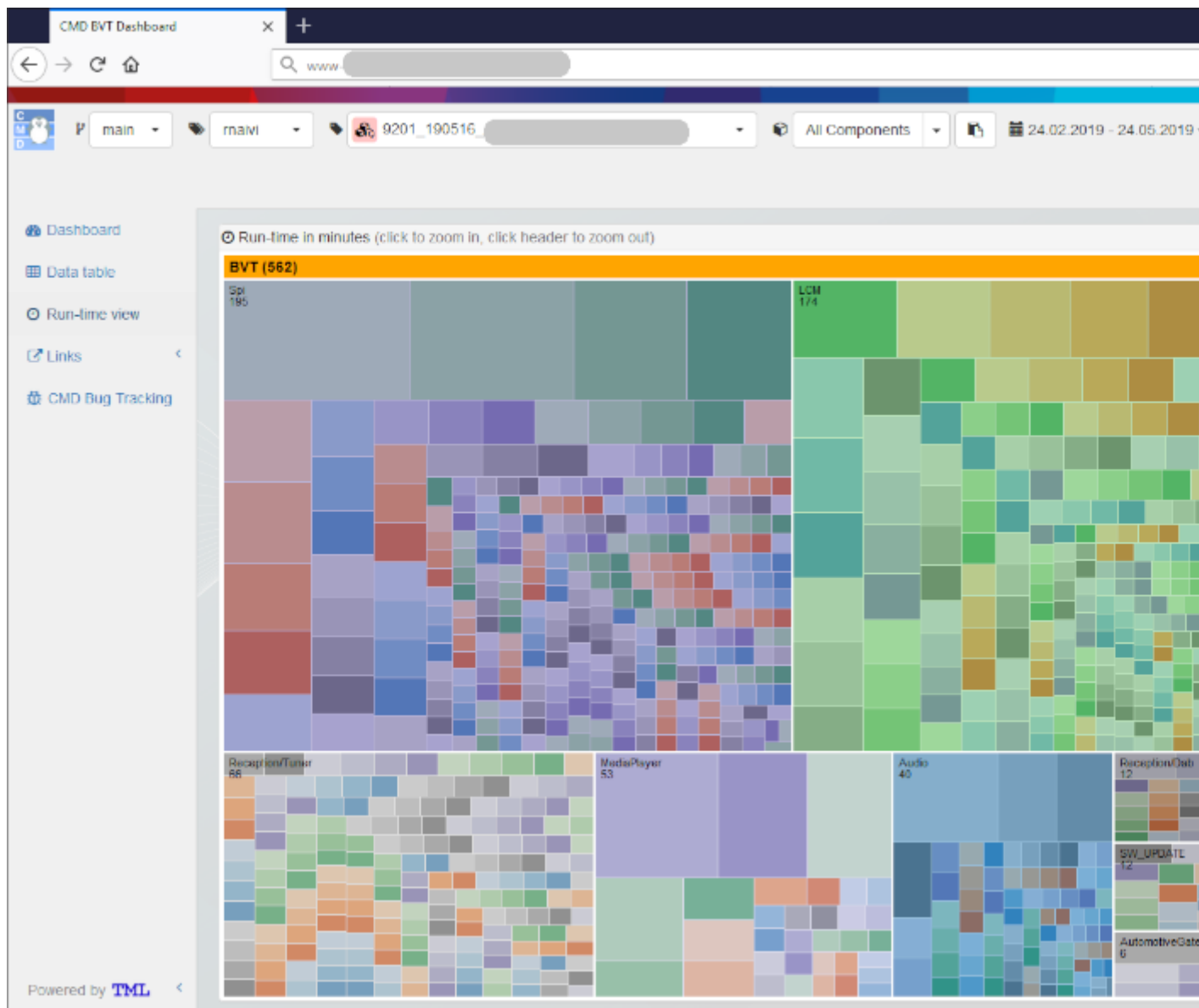
not passed 178

Show 10 entries

Result	Name	Component	tcid
✓	INVALID sink connection	Audio	TC1
✓	Connected to AUX_2	Audio	TC1
✓	change source from AUX_2 to TUNER_AM and sink internal Amp	Audio	TC1
✓	change source from AUX_2 to TUNER_FM and sink internal Amp	Audio	TC1
✓	change source from AUX_2 to MEDIA_PLAYER and sink internal Amp	Audio	TC1
✓	change source from AUX_2 to AUX_2 and sink internal Amp	Audio	TC3
✓	INVALID sink connection	Audio	TC1
✓	Connect to TUNER_AM	Audio	TC1
✓	change source from TUNER_AM to TUNER_FM and sink internal Amp	Audio	TC1
✓	change source from TUNER_AM to TUNER_FM and sink internal Amp	Audio	TC1

Powered by TML





Chapter 2

Description

The TestResultWebApp:
TO BE CONTINUED

Chapter 3

Appendix

About this package:

Table 3.1: Package setup

Setup parameter	Value
Name	TestResultWebApp
Version	0.1.0
Date	04.07.2022
Description	Web based display of test results
Package URL	testresultwebapp
Author	Thomas Pollerspöck
Email	Thomas.Pollerspoeck@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 4

History

0.1.0	07/2022
<i>Initial version</i>	

TestResultWebApp.pdf

Created at 11.07.2022 - 14:46:06

by GenPackageDoc v. 0.20.2
