RobotframeworkExtensions - v. 0.3.0

Holger Queckenstedt

13.05.2022

Contents

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:

`https://github.com/test-fullautomation/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	0.3.0
Date	13.05.2022
Description	Additional Robot Framework keywords
Package URL	`https://github.com/test-fullautomation/robotframework-extensions-collec`
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*)

Initial version

0.2.0 (*03/2022*)

Setup maintenance

0.3.0 (*05/2022*)

Documentation tool chain switched to `GenPackageDoc`