
Robotframework Extensions Collection

Queckenstedt Holger (XC-CT/ECA3)

Mar 23, 2022

CONTENTS

1	Introduction	1
2	Precondition	3
3	Robotframework Extensions Collection	5
	Python Module Index	9

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.

PRECONDITION

The *Robotframework Extensions Collection* requires an installed *Python Extensions Collection*

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

(Last update: 21.03.2022)

ROBOTFRAMEWORK EXTENSIONS COLLECTION

```
class Collection.Collection(sThisModule='Collection.py v. 0.2.2 / 21.03.2022')
```

Bases: object

The Collection module is the interface between the Python Extensions Collection and the Robotframework.

```
ROBOT_AUTO_KEYWORDS = False
```

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

```
ROBOT_LIBRARY_VERSION = '0.2.2'
```

```
normalize_path(sPath=None, bWin=False, sReferencePathAbs=None, bConsiderBlanks=False,  
               bExpandEnvVars=True, bMask=True)
```

Keyword: **normalize_path**

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

Args:

sPath

| *Condition*: required | *Type*: string |

The path to be normalized

bWin

| *Condition*: optional | *Type*: boolean | *Default*: False |

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

sReferencePathAbs

| *Condition*: optional | *Type*: string | *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*: boolean | *Default*: False |

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

bExpandEnvVars

| *Condition*: optional | *Type*: boolean | *Default*: True |

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

bMask

| *Condition:* optional | *Type:* boolean | *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:* string |

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

Library import:

The library containing the keyword definition can be imported in the following way:

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

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
```

pretty_print(oData=None)

Keyword: pretty_print

The pretty_print keyword logs the content of parameters of any Python data type (input: oData).

Simple data types are logged directly. Composite data types are resolved before 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.

Args:

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).

Library import:

The library containing the keyword definition can be imported in the following way:

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

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

PYTHON MODULE INDEX

C

Collection, [5](#)